Real-Time Scheduling for Multi-Agent Call Center Automation

Yong Wang, Qiang Yang and Zhong Zhang

Information Service Agents Lab
School of Computing Science
Simon Fraser University
Burnaby B.C. V5A 1S6 Canada
{ywangc,qyang,zzhang}@cs.sfu.ca

Abstract. In a call center, service agents with different capabilities are available for solving incoming customer problems at any time. To supply quick response and better problem solution to customers, it is necessary to schedule customer problems to appropriate service agents efficiently. We developed SANet, a service agent network for call center, which integrates multiple service agents including both software agents and human agents, and employs a broker to schedule customer problems to service agents for better solutions according to their changing capabilities and availability. This paper describes the real-time scheduling method in SANet as well as its architecture. There are two phases in our scheduling method. One is problem-type learning. The broker is trained to learn the problem types and hence can decide the type of incoming problems automatically. The other is the scheduling algorithm based on problem types, capabilities and availability of service agents. We highlight an application in which we apply SANet to a call center problem for a cable-TV company. Finally, we support our claims via experimental results and discuss related works.

1 Introduction

Our research is motivated by a realistic problem in call center environments. To solve customers' problems, many telecommunications companies, such as Cable-TV and telephone companies, maintain large call centers that are aimed at providing real-time solutions to their customers. In a Cable-TV call center environment, for example, a customer may phone in to ask about a solution to his fuzzy-picture problem. A human agent is selected for answering the question and interactively diagnoses the source of the problem. In this environment, the human agent's expertise is distributed and changing. They are not always available either due to time shifts, or changing interests and training. A solution is to create a number of software agents which can provide subsets of the expertise that human agents can provide, and cater to customers through a network of human and software agents. From the agent research point of view, a call center is a multi-agent system which includes customer service agents and an agent who schedules customer problems to service agents. Using agent development techniques, we have developed SANet — a service agent network for call center automation. SANet integrates both human and software service agents in providing customer service in real time, and employs a broker to schedule customer problems to service agents for better solutions according to the nature of the input problems as well as the changing capabilities and availability of service agents on the network. This paper mainly describes how SANet schedules the incoming customer problems to appropriate service agents, which includes a problem-type learning method and a problem scheduling algorithm based on the problem types, the capabilities and availability of service agents. As we will see, SANet is a flexible network on which service agents can be added or deleted at any time and their capabilities to solve problems can be tracked and utilized.

In this paper, we first describe the architecture of SANet. Then we present the problem-type learning method and the problem scheduling algorithm. Also we present an application in which we simulate a call center environment of a cable-TV company. Finally, we demonstrate through the experimental results that the proposed architecture and algorithms can provide high-quality service in real time.

2 The SANet Architecture

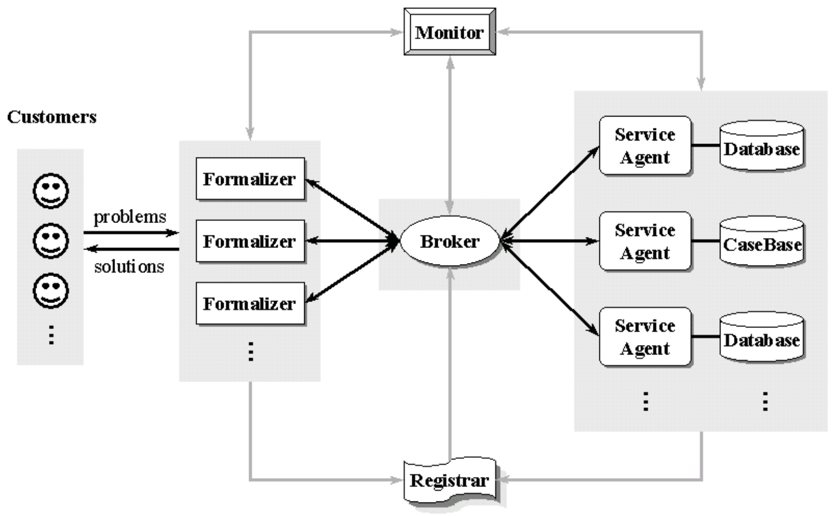

Fig. 1. The architecture of SANet

Figure 1 shows the architecture of SANet. There are two kinds of service agents on SANet. One is the software agent. A software agent is capable of solving one or more types of problems, and is associated with a solution database or case base. The other is the live agent. A live agent can be a human agent who is an expert in one or some specific areas. It may also be an integrated system of a human agent and a software agent. In this case, software agents act as a decision-support assistant to help human solve problems. Obviously, different service agents have different problem-solving capabilities. Their availability also change with time. Because problems in a call center environment may come in many manners, through telephone, fax, letter, e-mail etc. The formalizer agent transforms an unstructured incoming problem into a structure problem format using a set of attribute/value pairs. Any service agent or formalizer can be easily added to or deleted from the network by sending a registrar agent to the broker agent. The registrar is a mobile agent which carries information about the added/deleted agent, such as its most current capability, to the broker. The broker is an important agent on SANet, which is responsible for selecting the appropriate agents to solve problems according to the problem type, and capabilities and availability of the agents. The broker is trained to learn problem types and hence can decide the type of a given problem automatically. There are some subbrokers in the broker agent. Each subbroker corresponds to one type of problem and is created or deleted by the broker agent. Figure 2 shows the architecture of the broker agent.

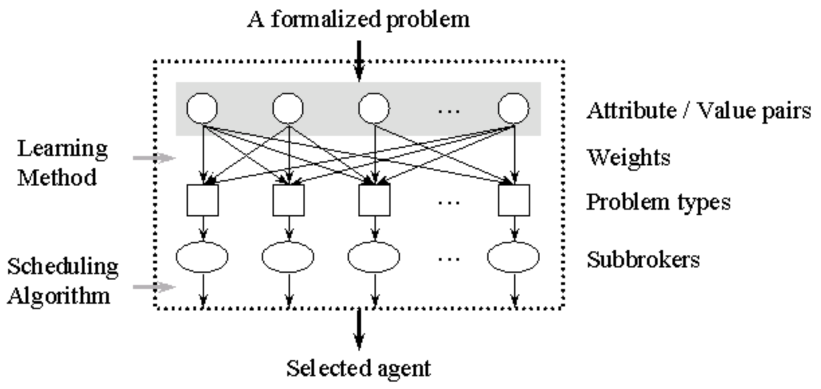

Fig. 2. The architecture of the broker agent

Each subbroker maintains a table of availability and capabilities of the agents which can solve the corresponding problem types. The broker maintains a table showing the correspondence between problem types and the subbrokers, a list of registered agents and a list of registered formalizers. When a formalized problem comes in, the broker has two steps to select an agent for this problem:

Step 1: decide the problem type using the result of problem type learning and then pass the problem to the corresponding subbroker.

Step 2: the subbroker selects an appropriate agent for the problem by using a scheduling algorithm.

In the next subsection, we discuss the learning method and the scheduling algorithm in details.

As soon as an agent receives a problem, it sends a "not available" message to the broker and then the broker can inform both the monitor and the subbroker. After an agent finishes working on a problem, it updates the quality of solving this type of problem by the following quality updating formula:

$$q'(t) = \frac{100q(t) + q(p)}{100 + 1}$$

where $q(t) \in [0, 1]$ is the average quality of solving the problems of type t before solving the problem p, $q'(t) \in [0, 1]$ is the average quality after solving p and $q(p) \in [0, 1]$ is the quality of solving p. One simple way to decide $q(p)$ is:

$$q(p) = \begin{cases} 1, & \text{if } p \text{ is solved;} \\ 0, & \text{if } p \text{ is not solved.} \end{cases}$$

Another way is to ask the user to provide a feedback ranking score on the quality of the solution provided by the agent.

An important feature of our SANet system is its ability to continuously update the capabilities and availability of different agents in real time. After a problem is solved, the solution and the quality at this time are added to the problem package. Then the agent sends back the problem package and the new quality to the subbroker, along with an "available" message to the broker. When a subbroker receives an availability changing message of an agent, it updates its availability table. When a subbroker receives a problem with a new quality returning from an agent, it updates its capability table. If the problem was not solved, then the subbroker delivers it again. Otherwise, the subbroker sends it back to the broker and the broker then relays it to the user through the corresponding formalizer. The monitor agent monitors the system performance and shows the changing capabilities and availability of service agents on real time.

3 Matching Problems with Software Agents Through Learning

In order to pass a given problem to an appropriate subbroker, the broker has to decide what type the problem is of. We can relay this task to the outside end users, but this will impose an extra requirement that an end user have to know the problem type *a priori.* This will not only burden the users but also, more importantly, force a user to choose a problem type when in fact the user may not know exactly what the problem type is. Furthermore, the situation is made more complex by the dynamic nature of the capabilities and availability of the distributed agents. Obviously, we wish to alleviate such a burden, making automatic the decision-making process of problem types in the broker agent.

As discussed before, after a problem is input, the formalizer will convert it into a standard representation form — a set of attribute/value pairs. Essentially, the broker will decide the problem type based on these attribute/value pairs. Therefore, we can view the relationship between attribute/value pairs and problem types as that shown in Figure 2. From the figure, it can be seen that each problem type is connected to a set of attribute/value pairs. We say that these attribute/value pairs decide this problem type. They distinguish a problem type from another. However, in real applications, not all attribute/value pairs have equal importance in this decision-making process. We thus consider assigning different weights to the connections between attribute/value pairs and problem types.

In [16], a learning model is applied to a network which is very similar to the one shown in Figure 2. The learning model aims to track its users' preferences dynamically, such that whenever the users' preferences change, the changes will be captured by the learning model and reflected in the following usage session. In our broker agent, we have the goal of learning problem types in the broker agent, given the attribute/value pairs. With the two-level architecture as shown above, the learning algorithm is a variation of the perception learning algorithm.

In particular, the training process of a problem type in the broker can be described as follows. After an input problem is formalized into a set of attribute/value pairs, these pairs will be fed into the network shown in Figure 2. The ranking scores for individual problem types will be computed. During the training, an expert critiques whether the highest-ranking problem type is the correct one. This will, in turn, be taken by the learning policy to update the relevant weights. In our broker agent, the training process is off-line assisted by domain experts. Our experiments show that after learning, the broker agent will choose, on behave of the user, the most appropriate problem type to be sent to the corresponding subbroker.

For the sake of simplicity, we do not discuss the learning model in detail here. Interested readers are referred to [16]. In Section 6, we will discuss the experiments we have conducted in order to demonstrate that the learning model fulfills our desired goal.

4 SANet's Agent Scheduling Algorithm

This section describes SANet's scheduling method for the subbroker to deliver problems to agents. The problems a subbroker may receive can be divided into three kinds:

new-problem A new-problem is one delivered from the broker.

problem-with-solution A problem-with-solution is one returned from an agent which solved the problem.

problem-without-solution A problem-without-solution is one returned by an agent which did not solve the problem.

Besides the capability table, the subbroker also maintains a problem queue. Each problem which can not be sent to any agent at current time will be put into the queue for future delivery. To select an agent to solve the problem, we define a capability function for each problem type.

Definition 1 *The capability function for a problem type t is denoted by $c(a) = c_t(a, q(a), t(a))$, where a is an agent which is capable to solve the problems of type t, $q(a) \in [0,1]$ is the quality for this agent to solve the problems, and $t(a) \in [0,\infty]$ is the solving time. Let α be a user-defined factor. For agents a and a',*

1. *If $q(a) \geq q(a')$ and $t(a) \leq t(a')$, then $c(a) \geq c(a')$.*
2. *If $q(a) \geq q(a')$ and $t(a) > t(a')$, then if $(q(a) - q(a'))\alpha \geq (t(a) - t(a'))$ then $c(a) > c(a')$ else $c(a) < c(a')$.*
3. *If $q(a) < q(a')$ and $t(a) \geq t(a')$, then $c(a) < c(a')$.*
4. *If $q(a) < q(a')$ and $t(a) < t(a')$, then if $(q(a') - q(a))\alpha \geq (t(a') - t(a))$ then $c(a) < c(a')$ else $c(a) > c(a')$.*

α acts like a weight to make $|q(a) - q(a')|$ and $|t(a) - t(a')|$ comparable. For example, it can be defined as $\alpha = \max\{(t(a) - t(a'))\}$.

We now describe the problem scheduling algorithm. We assume that we are given N problems, each associated with a problem type $i = 1, \ldots, K$. There are M agents A_l. We allow a problem to be of multiple types; in that case we assign a problem to a subbroker based on one of the types selected by the broker. Similarly, agents can have overlapping capabilities that are also changing with time. Also assume that there are N_i agents initially available to problem of type i, where each agent j has a capability defined by a value c_{ij} which is a real number between zero and one. We wish is to assign problems P_k to agents A_l such that the total sum of the capability value is maximized.

We adopt a greedy algorithm, extended to handle concurrent agent-based problem solving. The greedy scheduling method is shown as follows:

Algorithm SANet Scheduling Algorithm

Input: M agents $\{A_l, l = 1, \ldots, M\}$, K problem types, Agent capabilities where agent A_l has capability $\{c_{li}, i = 1, \ldots, K\}$, N problems P_{ij} where the problem is the j^{th} problem of type i, and a number N which is the maximum time a problem is allowed to be returned by agents without solution.

Output: An assignment of Agent A_l to Problem P_{ij}.

```
1. Partition problems $P_{ij}$ into $K$ typed groups, one for each sub-broker $B_i$;
2. concurrently for each $B_i$ $i := 1$ to $K$ do
3.     Set a queue of problems $Q_i$ sorted by come-in time;
4.     loop until $Q_i$ is empty do
```

```
5.    for every problem $P_{ij}$ in the queue $Q_i$, do
6.      if all agents have looked at $P_{ij}$, then
          give it to a designated human agent.
        else
          Let $S_{ij}$ be the set of available agents which haven't worked on $P_{ij}$;
7.    Let $P_{im}$ be the first problem in $Q_i$ with non-empty $S_{im}$,
      and $n$ be the time $P_{im}$ is already returned by agents without solution.
8.    find an agent $A_l$ with maximal capability from suggested agents
      in $S_{im}$, if any; otherwise, find $A_l$ with maximal capability in $S_{im}$;
      then send $P_{im}$ to $A_l$;
9.    if $A_l$ is not available and $n < N$, then
        return $P_{im}$ to $B_i$ without working on it.
      else
        $A_l$ must work on $P_{im}$;
        update capability for $A_l$ for problem type $i$;
        update and return the problem package to $B_i$;
```

This algorithm is designed to have the following two properties. First, we can ensure that no problem will wait forever before it is solved by an available agent. This is ensured in **step 9** of the algorithm, which commits a problem to an agent. Second, we claim that this greedy algorithm returns solutions with good quality and solving time. We verify this point experimentally in the next section.

5 SANet in a Cable-TV Call Center Environment

We have applied SANet to a simulated call-center environment of a local cable-TV company. In this application, a software service agent is a case-based reasoning (CBR) system that can answer typical questions or frequently asked questions automatically by retrieving the most similar case in the case base. A live agent is a staff in the call center with an assistant CBR system. At this point, the system can solve 16 types of problems. Each problem type has one to ten cases in one or more case bases. To input a problem, the user has to answer some questions. The question list is:

Q1 What's the problem type?
Q2 Which channels have the problems?
Q3 Is the account an active cable account?
Q4 Is the problem affecting more than 1 outlet?
Q5 What does the picture on the screen look like?
Q6 Is the customer in an affected area?

These questions are optional to answer. If the user answered question 1, which means the user knows the problem type, then the broker just passes this problem to the corresponding subbroker. Otherwise, the broker has to decide the problem type. In other words, the user has two choices — let the problem type be decided by himself or the broker. For instance, the user may have answered question 1, 2 and 3. Then the formalizer creates a problem package such as the following:

Problem ID	111
Description	the set of q/a pairs
Solution	Null
Suggested Agents	ϕ
Agent/Time List	ϕ

The set of q/a pairs is:

Q1	Poor reception
Q2	Channels 2 to 6
Q3	Yes

If the problem is sent to a live agent, the solution part may be filled by a staff who solved this problem. If it is routed to a CBR agent, a CBR system will then solve this problem and the solution part is filled by the most similar case in the case base. In this case, when the problem returns to the user, the solution part looks like the following:

Case Name	Poor reception low band
Q1	Poor reception
Q2	Channels 2 to 6
Description	Channels 2 to 6 have poor reception
Solution	Sometimes caused by a loose connection.
	1. Check channel outage note pad;
	2. Check for a possible loose connection;
	3. Try unplug power cord for television;
	4. Try hook cable direct to Television;

Also, the agent/time list will be filled. This list provides all agents which have worked on this problem and the solving time. The quality of solution can then be provided by the end customer based on his or her satisfaction with the solution. This feedback value is used to update the capability matrix for each agent. According to the solution quality, the end customer can also feedback a evaluation value of the problem-type decision to the broker to adjust the learning result of problem types.

This application system was developed by using Java JDK 1.1.6 and ASDK V1.0.3 (IBM's aglets software development kit).

6 Experiments

This section describes the experiments we have conducted and discusses the experiment results. We have done two kinds of experiments. One is for the problem-type learning in broker and the other is for the performance of SANet.

6.1 Experiment for the Problem-Type Learning

We introduce a learning model into our broker agent with the hope that it would automatically choose the most appropriate problem type on behalf the user. In the following we demonstrate our experiments with the learning model.

The learning process is off-line and mainly occurs in the broker agent. We create 20 problem types with each type being associated maximally with 20 attributes. In the experiment, on average each attribute can have 2 to 5 values. Accordingly, we also create a set of 100 user queries or problems represented by attribute/value pairs. The ranking of a problem type is between 0.0 and 1.0.

We feed these queries one by one into the learning model in the broker agent. The learning model then computes the rankings for the problem types and compares them with the desired ones specified by the queries. If there is a discrepancy between the computed ranking and desired ranking of a problem type, the learning process will be triggered and the corresponding weights will be updated. In the experiment, the learning process takes five rounds. Therefore, there is a total of 500 learning data points.

In Figure 3, we show the average error convergence chart for all the 100 queries after five learning rounds, where the X-axis represents the learning process which is composed of 500 learning data points while the Y-axis represents the average error among the 100 problem types in all the queries after each learning data point is learned. The error is defined as the absolute distance between the computed ranking and desired ranking of a problem type in individual queries. It can be seen that along the learning process, the average error for all the problem types tends to approximate zero and will be stabilized at learning rounds 4 and 5. We also see that because of the interactions among different queries, the learning error could not be zero no matter how long the learning process undertakes.

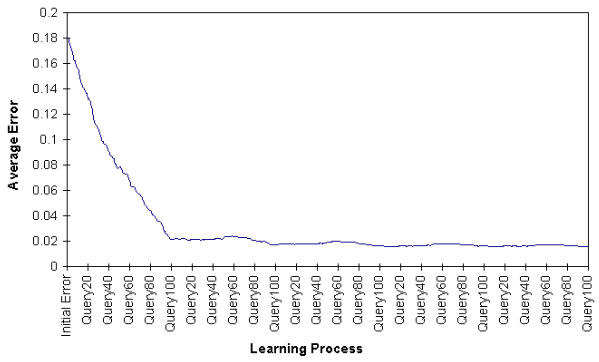

Fig. 3. Average Error Convergence Chart for 100 Queries

6.2 Experiment for the Performance of SANet

We have also conducted experiments to verify the performance of SANet. The application domain of the experiment is the call center of a cable TV company, as described above. In our experiment, we have 10 software agents which are CBR systems. Each software agent is associated with a case base in which there are 5 to 33 cases and 1 to 16 problem types. The solving time is set to $[n_c/300]$ seconds where n_c is the number of cases in the case base. We set different qualities to the agents for the experiment. Setting 1 sets higher qualities to the agents with lower solving time. Setting 2 sets lower qualities to the agents with lower solving time. Setting 3 is a mixture of Settings 1 and 2. Table 1 shows the different settings of our experiment. For each problem type, there are four agents which are able to solve this type of problems.

Table 1. Settings of experiments

Agent	Case Number	Type Number	Solving Time	Quality		
				S_1	S_2	S_3
A_1	5	5	0.01	1.0	0.1	0.1
A_2	6	2	0.02	0.9	0.2	1.0
A_3	6	2	0.02	0.8	0.3	0.2
A_4	6	6	0.02	0.7	0.4	0.9
A_5	10	1	0.03	0.6	0.5	0.3
A_6	11	7	0.03	0.5	0.6	0.8
A_7	16	7	0.05	0.4	0.7	0.4
A_8	18	10	0.06	0.3	0.8	0.7
A_9	21	8	0.07	0.2	0.9	0.5
A_{10}	33	16	0.11	0.1	1.0	0.6

S_i refers to Setting $i (i = 1, 2, 3)$

Initially, 100 random problems are created for the experiment. Then we send some of them to the broker 100 times. Each time, the first n problems are sent to the broker, where n varies from 1 to 100. For each problem, we calculate the solving time which begins at the time it is sent to the broker and ends at the time it is returned with the solution. When a problem is solved, we get the quality of the agent which solved this problem. Then, for n problems, we calculate the average solving time and the average quality. Figures 4 and 5 show the results.

These results highlight the following three main headlines.

- The system is stable for different settings.
  In our experiment, we used three different and typical settings as described above. We got very close results for these settings.
- The average solving time is nearly linear with the problem size.
  We observe that when the solving time of each agent for any problem is the same, the average solving time is linear to the problem number. Compared to this case, our result for the average solving time is reasonable.

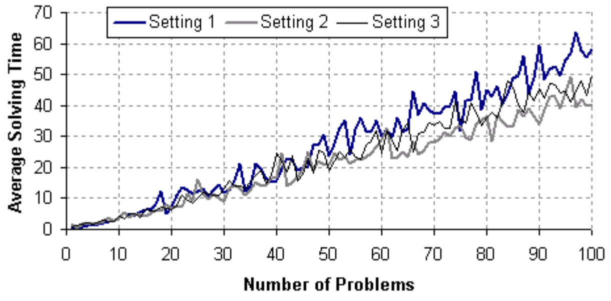

Fig. 4. Average Solving Time

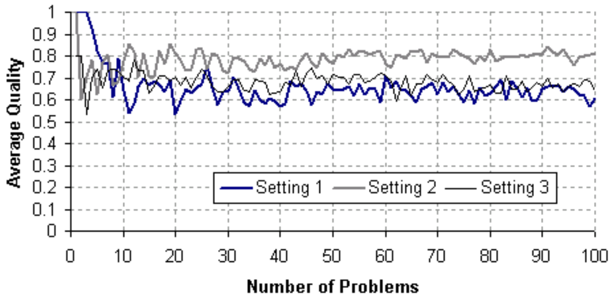

Fig. 5. Average Solving Quality

– The average solving quality is high and remains stable when the number of problems increases.
  These results show even when we have a large number of problems, the higher average solving quality can still be obtained.

7 Related work

Our system is closely related to *information gathering systems* in which there is usually an information broker. In such systems, the information broker is to gather information and find solutions from heterogeneous resources on a network of resources. Currently, many of them have been developed or are under development. Among them are BIG [9], Disco [15], Garlic [13], HERMES [1], InfoSleuth [12], Infomaster [6], Information Manifold [11,10], SIMS [2,3], TSIMMIS [4,7]. These systems take a user query and translate it into sub-queries that can be executed by various information agents attached to information sources. Most of them mainly focus on the integration of different knowledge representations. Therefore they do not take scheduling problem and the availability and the capabilities of resources or agents as main issues. But in our application domain, scheduling is an important problem because the primary concern in a a call center is to provide high quality service at a low service cost. The other major difference from information gathering systems is that, in our SANet system, a mechanism is designed so that after solving each problem, the quality and availability of the agent are updated and used by the broker to assign problems to agents in a better way.

Our work is also closely related to scheduling problems. In the past, scheduling has been widely studied by researchers [5, 8, 14, 17]. The problem is to schedule jobs to processors under certain constraints. A novelty in our work is the assumption that processor capability and availability are a function of time.

8 Conclusions and Future Work

This paper describes SANet — a service agent network designed for call center automation. We have focused on its scheduling method for customer problems. Our contributions are mainly the following:

- Availability and capabilities of service agents are considered.
- The capability of each agent to solve a type of problem is maintained by the broker agent. In SANet, the broker can learn and use the newest capabilities to select appropriate agents for the customer problems.
- As a network, the agents can be easily added and deleted at any time. This makes the network very flexible and easy to be maintained. The broker scheduling algorithm maintains the overall competency of the network by intelligently choosing the right software agents for a problem.

In our future work, we are considering to add a manager agent to manage the agents on the network. For example, if all agents on the network are very busy and many problems are still coming in, then the management agent should add some appropriate agents automatically. Some other efficient task-scheduling methods need to be studied and compared. Also, decomposing a problem into some sub-problems and then integrating the solutions returned by agents are left as our future work.

References

1. S. Adali and V. S. Subrahmanian. Amalgamating knowledge bases, ii - distributed mediators. *Journal of Intelligent and Cooperative Information Systems*, 3(4):349–383, 1994.
2. 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):127–158, 1993.
3. Y. Arens, C. A. Knoblock, and C.-N. Hsu. Query processing in the sims information mediator. In A. Tate, editor, *Advanced Planning Technology*. AAAI Press, Menlo Park, CA, 1996.
4. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The tsimmis project: Integration of heterogeneous information sources. In *Proceedings of IPSJ Conference*, pages 7–18, 1994.

5. J. Dey, J. Kurose, and D. Towsley. On-line processor scheduling for a class of iris real-time tasks. *IEEE Trans. Computers*, 45(7), July 1996.
6. O. M. Duschka and M. R. Genesereth. Infomaster - an information integration tool. In *Proceedings of International Workshop: Intelligent Information Integration, during the 21st German Annual Conference on Artificial Intelligence*, 1997.
7. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The tsimmis approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 1997.
8. J. Jonsson and J. Vasell. Evaluation and comparison of task allocation and scheduling methods for distributed real-time systems. Technical report, CTH, Dept. of Computer Engineering, Computer Architecture Laboratory (CAL), MicroMulti-Processor Group (MMP), 1996.
9. V. R. Lesser, B. Horling, F. Klassner, A. Raja, T. A. Wagner, and S. X. Zhang. Big: A resource-bounded information gathering agent. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, January 1998.
10. A. Y. Levy, A. Rajaraman, and J. J. Ordille. Query-answering algorithms for information agents. In *Proceedings of the 13th National Conference on Artificial Intelligence, AAAI-96*, pages 40–47, 1996.
11. A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieva*, 1995.
12. M. Nodine, B. Perry, and A. Unruh. Experience with the infosleuth agent architecture. In *Proceedings of AAAI-98 Workshop on Software Tools for Developing Agents*, 1998.
13. M. T. Roth and P. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *Proceedings of the 23rd VLDB Conference*, Athens, Greece, 1997.
14. P. Sparaggis and D. Towsley. Optimal routing and scheduling of customers with deadlines. *Probability in the Engineering and Informational Sciences*, 8(1), January 1994.
15. A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of disco. In *Proceedings of International Conference on Distributed Computing Systems*, 1996.
16. Z. Zhang and Q. Yang. Towards lifetime maintenance of case based indexes for continual case based reasoning. In F. Giunchiglia, editor, *Artificial Intelligence: Methodology, Systems, and Applications. 8th International Conference, AIMSA'98. Sozopol, Bulgaria, September 1998. Proceeedings*, volume 1480 of *Lecture Notes in Artificial Intelligence*, pages 489–500. Springer, 1998.
17. Y. C. Zhuang, C. K. Shieh, and T. Y. Liang. Centralized load balance on distributed shared memory systems. In *Proc. of the Fourth Workshop on Compiler Techniques for High-Performance Computing (CTHPC'98)*, pages 166–174, Mar. 1998.