VISITOR-HOSTER: Towards An Intelligent Electronic Secretary [1]

Katia P. Sycara
katia+@cs.cmu.edu
(412)268-8825

Dajun Zeng
zeng+@cs.cmu.edu
(412)268-8815

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

1 Introduction

The ubiquity of network-based information resources have given impetus for the development of intelligent software agents that will be able to (1) extract task-relevant information automatically or with little help from human users from various on-line information resources, (2) resolve the potential conflicts among acquired knowledge from different information resources, and (3) more importantly, collectively solve tasks requested by human users effectively without interrupting/bothering users too much.

In this preliminary report, we briefly present an implemented intelligent information system aimed at helping a human secretary organize a visit in an academic environment. The task of hosting a visitor involves arranging the visitor's schedule with faculty that match the interests that the visitor has expressed in his/her visit request. In our system, called VISITOR-HOSTER, various information agents are utilized to retrieve task-related information from several real-world heterogeneous data resources, such as internet-based `finger`, on-line electronic library, etc.

[1] This paper appears in the Proceedings of the CIKM-94 (International Conference on Information and Knowledge Mangement) Workshop on Intelligent Information Agents.

VISITOR-HOSTER is part of the PLEIADES project at Carnegie Mellon University. The broader goal of PLEIADES is to characterize and develop distributed agent-based architectures that are composed of *negotiating and learning agents* and apply them to tackle information and activity management problems for everyday use.

2 Software Agent Architecture

We developed a layered architecture in which *task-specific* software agents help users perform tasks by communicating with each other and/or querying and exchanging information with *information-specific* software agents, which provide intelligent access to a heterogeneous collection of databases. For example, the meeting scheduling module could be a task-specific agent, which will manage and update a particular user's appointment and meeting agenda. The general-purposed `finger` service module, which can extract useful information from the network `finger` utility given user's login name and IP addresses, can be viewed as an information-specific software agent. Although the boundary between these two types of software agents is quite arbitrary and vague, we make the distinction that typically task-specific agents access other agents (either task-specific or information-specific ones), whereas information agents (usually) accesses only information sources, such as library database records.

This architecture is mainly motivated by the following considerations:

- *sharability:* Many users can share information-specific software agents or task-specific agents. Typically, user applications will access several agents in parallel and one software agent can serve different application programs. The behavior of a software agent, essentially, could be easily described in a *server-client* model.

- *modularity and reuseability:* Although software agents will be operating on behalf of their *patrons*—human users, pieces of code can be copied from one user to another without modifications or with little adaptation according to particular users' preferences or idiosyncrasies. One of the basic ideas behind the distributed agent-based approach is that software agents will be kept simple for ease of maintenance, initialization and customization.

- *flexibility:* software agents can interact in new configurations "on-demand", depending on the information requirements of a particular decision making task.

3 Scenario: Organize a Visit

To examine the proposed software agent architecture, we chose organizing visits in an academic environment as a testbed for several reasons. First, hosting a visitor in an academic institution is a typical mundane secretarial activity of practical interest. Second, organizing a visit involves multi-stage decision makings among different agents. For example, a software agent should be able to retrieve relevant personnel information about potential meeting attendees and then set up meeting location and time between the visitor and attendees. Third, the task of organizing a visit does not really involve any deep specialized knowledge, which makes this application domain to be a good illustrative example for agent-based architectures. A different variation of the hosting visitor task has also been explored by Kautz and his colleagues at Bell Labs [KSC94].

A visitor hosting agent should have the following capabilities:

- It should automate information retrievals in terms of finding personnel information of potential meeting attendees. It should be able to access various on-line public databases and information resources at the disposal of the visit organizer. The system should also integrate the results obtained from various databases, clarify ambiguities (e.g., the synonyms for certain entities) and resolve the conflicts which might arise from inconsistency between information resources. Some possible information resources that are common to a modern university are: networking finger, on-line library, on-line phone-book, etc.

- It should create and manage schedule for visitors. It is also preferable if the meeting location and equipment can be managed in a coherent way.

- It should possess a graphical user interface which can interact with the users. The GUI is applied for getting input from the user, presenting acquired information, asking for user confirmation as well as advising the user of the state of the system and its progress.

To achieve the above requirements, we implemented the VISITOR-HOSTER system in the proposed layered architecture. Our prototype system focuses on the information resources accessible at Carnegie-Mellon University (we are planning to access other internet-based resources in the near future). The currently available components of our implemented system are:

- Information-Specific Agents

  1. `Finger` agent, which heuristically parses the retrieved information from remotely residing `finger` data bases. The possible types of information that can be acquired in this way include: work title, research interests, work and home phone numbers, vacation plan, etc.
  2. `Who's-Who` agent, which accesses on-line CMU who's who database through http-based queries. The fields in the database include: name, title, affiliation, campus office, campus phone number, home address and E-mail address.
  3. `Faculty Interests` agent, which can be used to retrieve information about the faculty members in the School of Computer Science at CMU with respect to their research interests.
  4. `Computer-Science-Directory` agent, which can get the information about phone number, office number, home address, etc. for all the members of the School of Computer Science at CMU, including faculty members, staffs and students.

- Task-Specific Agents

  1. `Host-Visitor` agent, which accepts input from the user concerning the information about the visitor and intended specification of possible meeting candidates, and initiates other related personnel information agents and scheduling agents.
  2. `Scheduling` agent, which takes the responsibility of maintaining a visitor's meeting schedule, coordinating among different meeting requests meanwhile taking into consideration possible meeting preferences of meeting attendees. For example, user A might prefer to meet with the visitor in the afternoon although meeting in the morning is also admissible.
  3. `Personnel Finder` agent, which coordinates all personnel information agents through a `Database-Mapper`, in which mapping functionality from available knowledge to information assistants containing desired information is provided. After answers from information agents get collected, `Personnel Finder` will try to

resolve conflict heuristically [2] and merge them together to get a coherent picture about meeting candidates.

4. `Interface` agent, which takes care of presenting acquired information from task or information specific agents to human users. It also handles the input from users. Separating interface functionalities from agent functionality helps increase the system modularity and makes it possible to enhance human-computer interface without affecting other parts of the system.

We take the following hypothetical visit to illustrate specifically how VISITOR-HOSTER works. Suppose Marvin Minsky wants to visit CMU CS department. And suppose Minsky wants to meet with some faculty members at CMU who are interested in machine learning. The entrance point to VISITOR-HOSTER would be that the secretary or the host of Minsky's visit inputs the relevant information about Minsky himself (affiliated organization, e-mail address, etc), the date of the visit, and his preference about meeting attendees, i.e. machine learning researchers. Then the `personnel finder` agent gets invoked and first accesses the `Faculty Interests` information specific agent to get a list of potential meeting candidates whose research interests match Minsky's preference. Then the `personal finder` agent spawns multiple queries trying to collect personnel information from various information specific agents, such as the `Finger` agent, the Who's-Who agent, etc., simultaneously. The resulting information is merged, conflicts are resolved and VISITOR-HOSTER selects the e-mail addresses of the senior faculty and automatically sends them e-mail asking if they would like to meet with Minsky on the date of his visit. Upon receipt of answers as to which faculty is interested in meeting with the visitor, VISITOR-HOSTER starts its scheduling agent and works out a feasible schedule with the help from involved meeting candidates' software agents, e.g., through exchanging appointment agenda and personal calendar information.

There are some interesting features in our implementation that deserve being mentioned here: (1) Information specific agents have a term-translator module associated with them. This translator module processes the retrieved data as to disambiguate the information as well as standardize and transform the keys used later to compare information from different sources.

[2] One of the heuristic rules we are using is that if the information returned by `Finger` is different from what `Computer-Science-Directory` found, we assume that the information based on `Finger` is more relevant and up-to-date.

Take an example, the phone number prefix for CMU 412-268 is automatically added to CMU extensions. Another example is that the field name *campus* used by library `Who's-Who` database is transformed to *office* — the term used in other databases. (2) In our implementation of information specific agents every separate process is handled with a timeout cap, which guarantees that a hung-up database will not hang-up the whole system. A default action gets called if the timeout is exceeded. Exceptional handling mechanism could implement actions such as retry access or give up and report failure.

4 Research Issues and Future Work

There are lots of interesting research issues in a distributed intelligent software agent architecture. The following list contains some of them that we consider are of importance and practical significance:

- What kind of communication protocol would be expressive and efficient enough for multiple agents to exchange information, request some particular action, etc.? Will KQML be powerful enough?
- Since inconsistencies and conflicts are almost ubiquitous, how to resolve them in a multi-agent environment is a serious problem. What kind of bargaining/negotiation protocol will suffice? Will it enforce truth-telling? Are independent *mediators* necessary? Do we need *brokers* to make the multi-agent cooperation more effective?
- To bring down information retrieval costs, costly data base searches should be minimized. One way of doing this is to have agents *learn* data base regularities either through directly analyzing the database or relying on previous retrieval experience to provide quick (maybe less accurate) answers to queries.

We are currently exploring these issues and working on incorporating our VISITOR-HOSTER system with other software agents available at CMU, particularly, a calendar management agent CAP [DBM+92] and an E-mail task apprentice agent to tackle the broaden problem of software agent architecture. We are also planning to apply our system into real-world everyday use.

5 Acknowledgments

This research has been sponsored in part by ARPA Grant F33615-93-1-1330. We want to thank Tom Mitchell, Dana Freitag, Sean Slittery, David Zabowski and other members of the PLEIADES project for interesting discussions. We also want to thank Gilad Amiri for doing much of the implementation.

References

[DBM+92] Lisa Dent, Jesus Boticario, John McDermott, Tom Mitchell, and David Zabowski. A personal learning apprentice. In *Proceedings of the Tenth National Conference on Artificial Intelligence*. AAAI, 1992.

[KSC94] Henry A. Kautz, Bart Selman, and Michael Coen. Bottom-up design of software agents. *Communications of the ACM*, 37(7), July 1994.