Agent-based Social Information Gathering on Internet

Eun-Seok LEE
Dept. Information Engineering
SKKU(Sung-Kyun-Kwan University)
Seoul, Korea

Roberto OKADA and **Norio SHIRATORI**
Graduate School of Information Sciences
Tohoku University
Sendai, Japan

Background

The popularization of computers and the Internet have produced an explosion in the amount of information, making difficult to find them. The problem with existing information-locating support systems is that although they allow user to search through a large number of information sources, they provide very limited capabilities for locating, combining and processing informations. The load of finding information is still on the user.
In order to support the user in finding information in such environments, we propose what we called a "completely agent-based framework for Information Gathering", named *Cooperative Agent Society for Information Gathering* or CAS_{IG} in short. This CAS_{IG} is based on the integration of collaborative interface agents, domain-specific mediators and cooperative information sources.

Architecture:

We divide the necessary functionalities between the user and the distributed information sources basically in three types of agents:
(1) User Agent (UA) or *user-specific agents*: one per user. It takes care of user preferences, manage personal information and acts as the user's electronic personal assistant. These agents are similar to the existing interface agents, which perform tedious, repetitive and time consuming tasks on behalf of the user or act as an abstracting interface between the user and the low-level details. In our framework, UA would dialogue with the user in order to acquire user requests, helping him formulating the proper query **(2)** Machine Agent (MA) or *information-specific agents*: attached to the information sources, controlling access to the information they provide. By doing so, negotiation capability, security and consistency of data can be ensured **(3)** Manager (Man) or *task-specific agents*: they stand in between the providers and consumers of information or services. In this framework, they have Domain-specific knowledge and plan how to satisfy user requirements in their domain of expertise. This type of agents have been researched at different levels of complexity, from very simple routers for supporting communication to complex mediators with task division, allocation and planning capabilities.

In this paper, we focus on two aspects: (1) the UA which (i) monitors user actions for finding information, (ii) learns user preference and keeps it as a personal data in user's personal profile, (iii) builds a trust relationship with other agents based on past experiences and (2) the social aspect i.e. the mechanism for interaction between UA owned by different users such that when one agent alone cannot satisfy user request it asks for advices or help to other UA it trusts.
We divided the implementation task in two phases. In the first phase we focus on *distributed intelligence* with the introduction of several agents and the negotiation aspects in such a way that this cooperative agents show an intelligent behaviour.
In the second phase, we focus on *autonomous intelligence*, i.e. making each of the agents more "intelligent", where we'll investigate more powerful real-time planning and learning algorithms.

The prototype

The multi-agent system under construction is implemented in C and using the libwww. It uses Netscape as a web browser and user interface, and the agents run as separate processes. This system incorporates one UA per user, several primitive Man (meta-index like) , and the existing web (including search engines e.g Lycos, InfoSeek and the document servers distributed all over the Internet). When the user invokes his personal UA, it opens a personal log file to keep meta-information about the documents which the user finds interesting. This UA also maintains a table of "trusted peers", their location and their trust relationship - which is updated based on user feedback after his evaluation of retrieved documents. When this UA receives the query as a set of keywords from the user, it first select the agents with higher trust values and send them the query. The selected UAs look at their personal log file for documents retrieved in the past related with the query in question, while the selected Mans choose some search engines to send this request. This engine selection is done based on a trust table with the list of known search engines, their location and a weight indicating how successful they were in the past answering about this query. Mans receive pointers to candidate solutions from the selected search engines, re-order them and return this ordered list of pointers to candidate solutions as an HTML document. This document is presented to the user, he selects one pointer causing the document to be retrieved. The user evaluates this document and provides a positive or negative feedback. This causes the update of the weights of the trust relationships of this UA to the agent which proposed this solution and the trust weights of the Man to the search engines as well as the user's personal profile.

Acknowledgement:
The authors would like to thank Hideto Kihara for his contribution in the prototype implementation.

A Multiagent Meeting Organizer that satisfies Soft Constraints

Joo-Hwee Lim†, *Jiankang Wu*† & *Siet-Leng Lai*‡

†RWCP Novel ISS Laboratory
Institute of Systems Science
Heng Mui Keng Terrace, Singapore 119597
Email: {joohwee, jiankang}@iss.nus.sg

‡Center for Computer Studies
Ngee Ann Polytechnic
535, Clementi Road, Singapore 599489
Email: lsl@np.ac.sg

As a distributed resource allocation problem, meeting scheduling is a tedious and time-consuming process. This paper proposes a multiAgent MEETing organiZER (AMEETZER) that respresents and reasons with soft constraints related to the meeting attendees and resources. An AMEETZER accepts call-for-meeting request from its user (host) and communicates with other AMEETZERs of the proposed attendees and with the agent managing the meeting rooms to arrive at some commonly free time slot, taking into account the pre-specified and dynamically created hard and soft constraints of all attendees and meeting resources. When necessary, the AMEETZER could perform negotiation of meeting time or relaxation of constraints with or without the intervention of its user. Last but not least, the AMEETZER reminds its user of forthcoming meetings at appropriate times.

With distributed AMEETZERs, meeting requests that involve disjoint sets of attendees and resources can be scheduled concurrently, as contrast to a centralized approach which faces communication and processing bottlenecks as well as fault tolerance and complex schedule maintenance issues. To further increase the concurrency of the system, each AMEETZER is decomposed into four subagents, *Receptionist* (interacts with user to manage his calendar and preferences and carries out meeting proposal and negotiation etc), *Scheduler* (infers the optimal time (and room) for a meeting under hard and soft constraints etc), *Messenger* (communicates with other AMEETZERs), and *Learner* (learns scheduling preferences). Together the calendars and preferences are used to generate instances of soft constraint set that specify the preference values ($\in [0,1]$) on the meeting times. These preference values are communicated to and inferenced by the host AMEETZER to generate utility-optimal time for the meeting.

Another challenging aspect in real life scheduling problem is the representation and reasoning of hard and soft constraints. In real life, *goals* can also be conveniently expressed as preferrences (soft constraints) since satisfaction-seeking is more realistic than optimality-seeking. Indeed, in fuzzy decision making, the decision set is the intersection (denoted as $\wedge$) (or more general, *confluence*) of the goals and the constraints,

$$\mu_D(X^*) = \max_X\{\mu_G(X) \wedge \mu_C(X)\} \tag{1}$$

where D, G, C are decision set, goal fuzzy set, constraint fuzzy set respectively and μ_D, μ_G, μ_C are their membership functions respectively, X^* is the optimal solution in the domain over which X ranges.

In AMEETZER, conventional crisp constraints (n-ary relations) are extended to fuzzy constraints founded upon fuzzy relations. Constraints imposed on meeting time by the attendees or meeting rooms can be expressed as fuzzy relations defined on the Cartesian product space of *Day* (in a week) $\times$ *Time* (in a day), denoted as T. In a simple case, a time preference is represented as

$$c_i = \{\mu(t) | t \in T\} \tag{2}$$

where $\mu(t) \in [0,1]$ represents the preference for a meeting to be scheduled at time t as seen by the system, an attendee, or a meeting room. While $\mu(t) = 0$ denotes an impossible time for meeting possibly due to other commitment, $\mu(t) = 1$ indicates maximal preference. Priorities among attendees (or meeting resources) and meetings can easily be incorporated by introducing weighting coefficients for c_i or sieving c_i through modifiers (or linguistic hedges).

Using the framework of (1), AMEETZER infers the most preferred time t^* by all N attendees as

$$\mu(t^*) = \max_t\{\min_{i \le N} c_i(t)\} \tag{3}$$

which provides a well-defined utility measure (optimal degree of joint constraint satisfaction).

Ameetzer is implemented in Java to take advantage of its object-orientation and platform independence.