

The Bibliography Agent Community

Jane Yung-jen Hsu Tzong-han Tsai Keh-ming Luoh Shih-jui Lin
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

December 5, 1998

Abstract

There are more and more academic documents on the Internet. The bibliography agent community is designed to collect these online documents. This community is a multi-agent system. Users can query this system by a bibliography entry, which records some meta data about the target document. The agents in this community will try to find the online version of the document together with some relevant information on the Internet. Each agent has its own domain knowledge to assist it to gather information from different kinds of information resources. Agents will communicate with each other, and return the organized result to the user. Some experiments have been done, and their results will be presented in this paper.

1 Introduction

The World Wide Web is getting more and more popular and various kinds of resources are available on it nowadays. Besides entertainment and commercial information, the growing number of academic documents could be accessed on line. Downloading papers from the Internet becomes a new means to catch up with the rapid academic development.

The traditional library can give us authorized information and all of us are familiar with looking for materials there. Nevertheless, the local library usually does not have all the books, periodicals, or papers we want. Moreover, sometimes we only need a specific document instead of the whole volume; it is not convenient to get a pile of books only for several pieces of documents. Since the Internet contains richer and richer information, it can be taken as a large, worldwide library to complement the traditional one. Therefore, we can get more information more efficiently.

At present, lots of academic online documents can be retrieved from these information resources:

- homepages of publishers, journals, and conferences

- homepages of institutions, schools, and organizations
- homepages of the authors, editors
- homepages of other persons' collection

However, searching for documents on the Internet differs from looking for materials in the library. In the library, when we get result from the library querying system, we can make sure that the book includes the target paper is in this library. Nevertheless, searching for documents on the Internet will not be so easy. Since all kinds of information resources have different characteristics and they are organized in different ways, trying to find the target document among different information resources will be very complex. In our opinion, using multi-agent system can simplify this process, since we can divide all the work and conquer it. We can design some special purpose agents for each information resource. Each agent can have its own domain knowledge to make navigation in that information resource more efficient. Also, we can upgrade an agent without affecting the whole system architecture and other agents. In addition, when we want to explore a new kind of information resource, all we have to do is adding a new agent to handle it. Moreover, the agents can also share their functions and cooperate with others in this multi-agent framework.

2 The Bibliography Agent Community

2.1 Architecture

There is a bibliography agent community, including a broker, a UI (user interface) agent, a library agent, a spider agent, and several search agents (a direct-fetch agent, a publication search agent, a fixed-format agent, and a general search agent). This community is centralized, and the broker mediates all the communication between any two agents. The broker itself will not initiate any communication, but it can receive the message from the initiative agent and send it to the designated agent.

When the UI agent receives a query from the user, it can send the relevant information to the library agent, which will return some meta data about the bib (e.g. what library owns this book, or journal, what the code (//??) it is). The UI agent can also send the relevant information to search agents and ask them to look for the online document on the Internet. The search agents will process the relevant information with their domain-specific knowledge, then delegate the searching job to the spider agent with aided information. The spider agent is responsible for navigating on the Internet and fetching the target document. Afterwards, the spider agent returns its result to the search agent, which in turn, sends back this result to the UI agent.

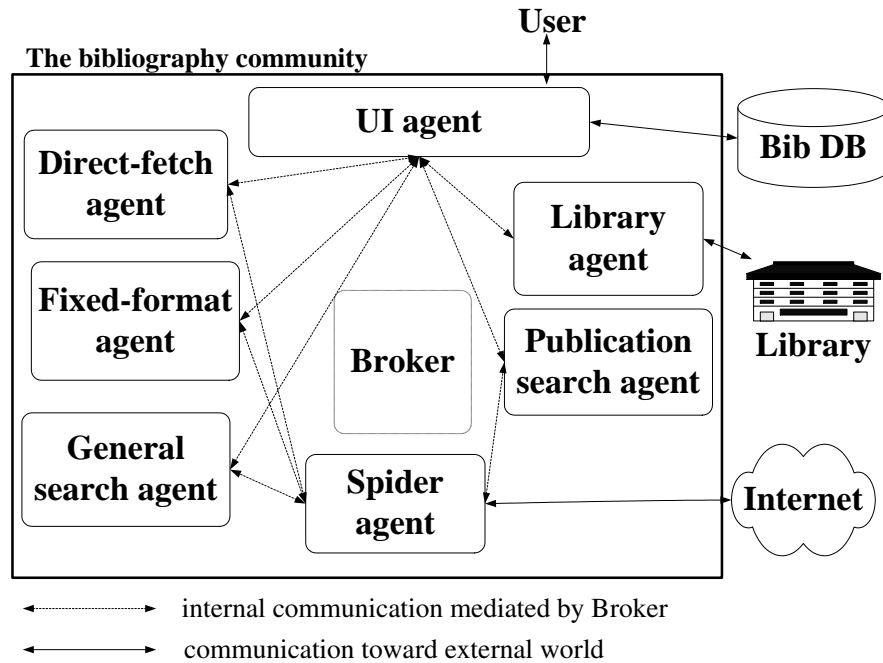


Figure 1: Architecture

2.2 Communication Protocol

The communication protocol we designed is like KQML (Knowledge Query Manipulation Language). A message communicated between two agents includes: (1) sender (2) receiver (3) performative (4) content. All the messages are sent by means of Internet socket .

The performative indicates which kind of task the sender asks the receiver to do and what kind of content this message contains. We have implemented these performatives:

- Query by bib: The UI agent asks the library agent and search agents to search for information related to the given bibliography entry.
- Found by bib: the library agent and search agents return the search result to the UI agent.
- Find by bib: The search agents ask the spider agent to search for the target document from a given webpage.
- Found links: The spider agent returns its search result to the search agent.
- Tell new heuristic: Search agents tell each other the new heuristic learned from recent experience.

The content contains many fields. Each field has a field name and a stream of value. The value can be a character string, a structured record, or even a file. After getting the message, the receiver specifies the field name and gets the corresponding value. When there are more than one value in the field, the receiver can use the primitives "get first" and "get next" to get all of those values one by one.

3 Implementation

3.1 Broker

The broker is similar to the facilitator in KQML. However, our broker is a simplified one. It has two main functions: (1) registration, and (2) mediation.

Every agent should register at the broker when entering this agent community at the first place. Therefore, the broker knows every agent's name and location (the machine IP and the communication port). When any of the agents in this community wants to send a message to other agent, it only needs to tell the broker the name of the designated agent along with that message. Since the broker knows where these two agents are, it can mediate the messages between the agents successfully. Moreover, when one agent requests to communicate with another one, who has been locked by other agents, the broker will turn down the request and tell the initiative agent to handle this exception.

3.2 UI agent

The UI agent has a CGI front-end. It can receive two kinds of queries: (1) uploading a bibliography entry from a local file; (2) filling the form on the webpage. Their queries need to be parsed into standard bib fields, especially the former one. (~~//erase this sentence? especially..~~)

Both kinds of queries need to be refined if the information in the bibliography entry is too little to uniquely identify the document. For example, if the bibliography entry only contains "author = Maes, Pattie", or "journal = AI Magazine", it is impossible to recognize which document the user really needs. Because there are hundreds of documents satisfying these loose constraints at the same time. Since we have problems to define what we need precisely, how to get it on the Internet would become an unreasonable task. To solve this problem, the UI agent will query a bib collection site (~~//ref~~), and feedback all the corresponding bib entries to the user. Given this further information, the user could revise his query to specify which document he really wants to get.

As long as the UI agent gets enough information, it would dispatch this query to search agent. There are two modes which can be specified by the user: (1) to delegate to only one specific search agent, (2) to ask for all the search agents together with the library agent. When the latter mode is chosen, the UI agent will call the search agents one by one. In order to return the result to the user as soon as possible, the UI agent calls the simplest, fastest, yet low-recalled agent

first. Then call the more powerful, high-recalled agent, which would spend more time to navigate on the Internet before getting the target document. (//need to mention the order?)

After gathering the results from those search agents, which could be a bunch of hyper links or some relevant meta data corresponding to that required document, the UI agent presents the organized results to the user in the form of a webpage. (//sentence too long?)

In sum, the UI agent is not only an interface between the user and search agents, but also an agent getting back the missing fields when the bibliography entry is incomplete. It can provide the user with some useful meta data about the target document by itself.

3.3 Library agent

The library agent receives meta data of a paper from the UI agent, sends query to the CCU Integration Library Query System (//ref) and returns querying results.

Since the library querying systems maintain complete meta data of its collections manually, by which we can correct the possible errors or deviations of the original bibliography data from users. In case the given document doesn't have an online version, the library agent can also guide users to find this paper in libraries.

3.4 Direct-fetch Agent

The direct-fetch agent is the most simple search agent. First, this agent examines whether the bibliography entry contains a URL. If it does not, the direct-fetch agent won't do anything but report its failure to the UI agent. If the bibliography entry does contain a URL, the direct-fetch agent further checks up what kind of URL it is. If it is a site or a homepage, the target document could be just around the corner. Therefore, the direct-fetch agent asks the spider agent to search for the target document from this URL. Furthermore, if the URL is a text file (e.g. *.doc, *.txt), or a postscript file (*.ps), or any kind of compressed file (e.g. *.gz, *.tar), the URL itself may be the target document since these kinds of files have no hyper links in it.

Therefore, the direct-fetch agent will return this URL to the UI agent as the result. Because the task of the direct-fetch agent is simple, it won't take too much time to return the result on the average. However, its recall would be low because it refuses all the bib entries without URLs.

3.5 Fixed-format agent

The fixed-format agent has a knowledge base, which describes how to get documents from some well-organized web sites. Each entry in the knowledge base contains two fields:

- Match pattern: it's a list of (field-name, field-value) pair describing the criteria to perform this fixed-format search
- Solution:
 - A URL: recording where the journal's or the author's homepage is
 - A list of matching fields: recording what fields in the bibliography entry to match when browsing the pages in the site

If a bibliography entry doesn't match any pattern in the knowledge base, the fixed-format agent refuses this job and returns to the UI agent. If a bibliography entry matches a specific pattern in the knowledge base, the fixed-format agent sends the URL to the spider agent together with the list of matching fields. This information is used to indicate where the spider agent should start its navigation and how to traverse in that site. Therefore, it can guide the spider agent to fetch the target document through the shortest path.

The knowledge base could be built in by human manually, or accumulated by the general search agent when it performs a new task successfully.

In sum, the fixed-format agent cuts down the branching factor of hyper links, and makes the spider agent head for the target document efficiently. Because the spider agent expands only one link each level, the searching time won't be too long. However, as the direct-fetch agent, the recall of the fixed-format agent may not be so high, either. Even if it can learn from experience and accumulate its knowledge, there are still lots of documents, yet not well organized, scattered on the Internet, and couldn't be retrieved in this way.

3.6 Publication search agent

The publication search agent is an intelligent agent, which can receive data of a bibliography entry from the UI agent, and output the URL of this bibliography entry if online document exists. Or it will tell the UI agent that no result has been found.

The function of the publication agent can be divided into two parts. First, it examines the value of publisher field of the given bibliography entry. If the name isn't in the knowledge base, it will tell the UI agent that the publisher of the bibliography entry is not supported by it. Otherwise, it produces the query of the bibliography entry, sends the query to the appropriate publisher's query interface in WWW, and gets the results. Second, it will analyze the results with two strategies. If the results contain exactly one URL of the online document, it will tell the UI agent what the URL is, including some descriptions. In case the results consist of two or more URLs of online documents, it will ask the spider agent to determine which link is the target document.

3.7 General search agent

The general search agent also has a knowledge base, which describes how to formulate the query string for each document type. For example, if a bibliography

entry refers to an "article", we could query the search engine by "journal" first, while an "tech-report" could be found by querying the "institution" first.

When the general search agent gets a bibliography entry from the UI agent, it picks up the values of the fields on the list, then generates the query string according to the syntax of a specific search engine (e.g. MetaCrawler). After getting back the result page, it will eliminate irrelevant information on the page (e.g. logos and advertisement links) and send the simplified page to the spider agent.

If there is some useful result returned by the spider agent, it terminates this job and returns to the UI agent. However, if no promising result is found by the spider agent, it checks up its knowledge base to formulate another query string and does all the steps above until some results are found or no more heuristic in the knowledge base.

When the result is excellent (//note 1), the general search agent will record every step of this navigation, and tell the fixed-format agent as a new heuristic. Then, when a similar bibliography entry is encountered next time, the fixed-format agent can provide the spider agent a more efficient navigation path to get the target document.

3.8 Spider agent

The spider agent takes a URL, as input and this URL should be HTML file. Then the spider agent fetches the webpage back and starts to navigate the surrounding Internet resources from this page. The main workflow of the navigation includes:

- to evaluate the whole page, determine the relevance score between this page and the target document (or it is the target document)
- to extract each hyper link on this page by a SGML parser(//ref)
- to evaluate each hyper link according to the title of the link and the text around it
- to weight the score of the whole page, and the link, (//and the score of its parent page), keep the links and their scores in a fringe
- to pick up the link with the highest score in the fringe and fetch the URL back (it is just like the best-first search)

Repeat this process until the target document is found or all the links in the fringe are below the pre-defined threshold (that means, all the links are possibly irrelevant).

The whole page evaluation and the link evaluation determine the score according to how well the page (or link) matches the fields of the bibliography entry (that is: author, title, key, journal, booktitle, editor, publisher, year, volume, number, organization, note, school, institution).

- The heuristic of the page evaluation and link evaluation is similar:
- The more fields are matched, the higher the score will be.
- The more important fields are matched, the higher the score will be. In our present implementation, the order of importance is title > author > key = journal = booktitle > editor = publisher > year = volume = number = organization = note = school = institution
- The score of complete match is higher than that of partial match. In some fields, such as author, editor, publisher, would have more than one value. The more complete the page (or link) matches the field, the higher score it can get.
- The more important the match location is, the higher its score will be. For example, in the page evaluation, match at the top gets more score than at the bottom. Similarly, in the link evaluation, match at the topic of the hyper link (//topic??) gets more score than at the surrounding text of that link.

In our implementation, the spider agent can also receive a "processed" HTML file or get a "matching list". The word "processed" means this HTML file has been examined by other agents, and some irrelevant information has been eliminated. The "matching list" indicates what fields to match on the webpage. Both of these special inputs can help the spider agent perform more efficiently on evaluation and navigation. The cooperation between the generic spider agent and other domain-specific agents will make the navigation more efficient.

4 Experiment Result

5 Discussion

During the process of designing these agents and the agent framework, we observe that:

- The overhead of communication among agents cannot be ignored. Since all messages must be transmitted, the broker agent possibly becomes the bottleneck of the agent system. Besides, the broker has to repeat messages, do string processing, and handle requests from other agents. In my opinion, there really exists some overhead in agent communication. However, the architecture of multi-agent system improves the flexibility and autonomy of this system.
- Not all papers we want are on the Internet. This is a serious problem for our work. As a matter of fact, the amounts of papers that have been published on the Internet are few. However, we believe that more and

more papers will be published on the Internet. If our spider agent can recognize the link of target document more clearly, we'll improve our hit rate better and better.

- Some proceedings of conferences and articles of journals are available in the abstract form. Some publishers provide abstracts of papers on the Internet. There are two possible conditions. The first is that the paper has an online version, but the online version is not in the conference or journal web sites. Maybe the online document is collected in the author's homepage. In this condition, the abstract from the publisher will enhance the meta data. Therefore the search agents will formulate better query and recognize the target document more precisely. The second is that the online document of the paper really doesn't exist. In this condition, the abstract of the paper can be returned to the UI agent, and our users can use it to improve their understanding of this paper.
- The Internet is dynamic. This is also a serious problem for our search agents, especially the direct-fetch agent and the fixed-format agent. If the target file is removed, the direct-fetch agent will fail to find the target document. In case the organization of the publisher web site is changed, the fixed-format agent will mismatch the new navigation path. There are two remedies for this problem. The first is finding the root page of the original web site heuristically. Then, the spider agent will discover the new navigation path and organization rules of this web site. The second is using the general-search agent to find this paper again. If the paper still exists on other mirror site of original site or just change its URL, the strategy will be successful.

6 Advanced Features

In order to make this bibliography agent community more robust and more efficient, some enhancement could be done in the future:

1. To enhance the function of the broker:
 - generate multiple copies of agents dynamically when needed
 - classify agents with respect to their functions and services
 - specify the agents category instead of agent's name (when calling service)
2. To enhance the function of the general search agent:
 - try more search engines to increase the recall
 - tune the heuristics in knowledge base to improve the performance
3. To aim at personal homepage:

- use Ahoy! (`//ref`) to find the homepage of the authors and editors
- design suitable heuristics when navigating the personal homepage

7 Conclusion

We have designed a bibliography agent community. Given a bibliography entry, agents in this community will try to find the corresponding online documents throughout the Internet. We have implemented:

1. a UI agent — to interact with the user and to make the bibliography entry complete
2. a direct-fetch agent — to get the target document from the URL appeared in the bibliography entry
3. a library agent — to get some relevant meta data of the document from libraries
4. a fixed-format agent — to get documents from well-organized sites
5. a publication search agent — to search for documents by querying the database of publisher sites
6. a general search agent — to search for documents by using the result returned by MetaCrawler

These agents do work, and some experiment results have been presented in this paper.

8 Acknowledgement

We gratefully acknowledge the contributions of Bo-chieh Yang and Liang-yun Wang for their work on the framework of this bibliography agent community. We also thank Wen-jyh Chen, Yu-chong Li, and Bo-heng Lin for the implementation of agents.