An Agent System that Assists User's Work Using Case-based Reasoning

Yoshiaki YASUMURA, Sachiko SUZUKI, Katsumi NITTA

Tokyo Institute of Technology, CISS
4259 Nagatsuta Midori, Yokohama 226-8502, Japan
Tel: +81-45-924-5215, Fax: +81-45-924-5215
E-mail: yasumura@ntt.dis.titech.ac.jp, sachi@ntt.dis.titech.ac.jp, nitta@ntt.dis.titech.ac.jp

Abstract

This paper introduces an agent system that assists in user's work on the Internet. First, the agent receives requests from the user or other agents. Since there are various kinds of requests, it is difficult to describe a complete set of request-handling rules in advance. Therefore, the agent makes a plan referring to old cases. The agent executes the plan, which is a sequence of basic operation. If the agent fails to execute basic operation or to create a plan, then it makes a new plan by interacting with the user or other agents. Finally, the agent stores this new case with user's evaluation score into the case base.

Keywords:

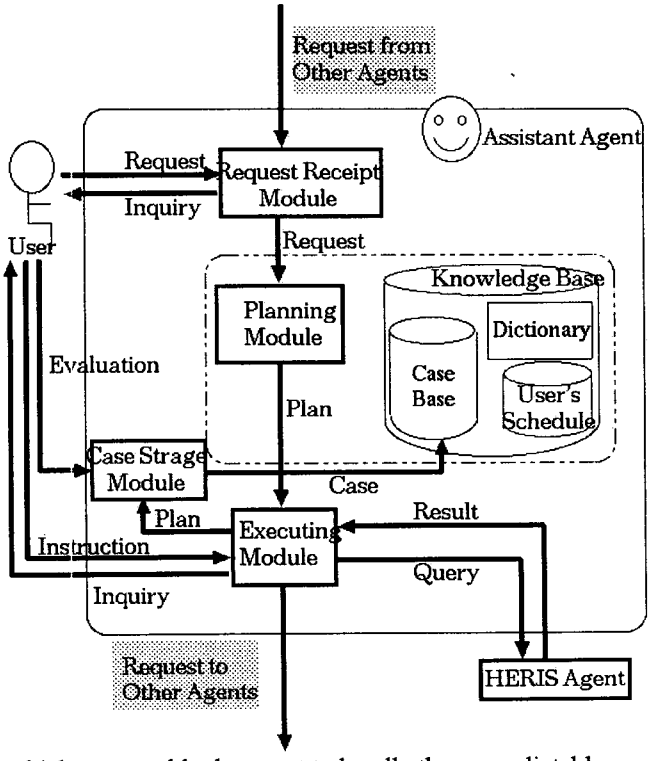
CBR; Secretary Agent; XML; Planning

Introduction

Recently, most of office workers use computer network for various kinds of dairy work such as contact with others and information retrieval. To assist the workers, the importance of assistant agents like a secretary seems to increase.
So far, assistant agent systems have been developed. Maes and Kozierok introduce interface agents for the electronic mail domain [1]. The agents learn users' action for the electronic mail by monitoring their action, and then offer to automate them. Kida et al. developed a meeting scheduling system with multi-agent [2]. In this system, the agent knows user's preferences, and negotiates with the other agent to coordinate group meeting.

These agents are designed for a specific domain such as the electronic mail domain and meeting scheduling. In order to assist a user in his/her network-based work, the agent needs the ability to handle the requests in the general domain. However, such requests might be unpredictable or inexecutable, because it's difficult to describe a complete set of request-handling rules in advance.

Therefore, we develop an agent system to assist in user's dairy work referring to old cases. The agent creates a plan to handle user's requests by Case Based Reasoning (CBR),



which may enable the agent to handle the unpredictable requests.

If the agent fails to execute the plan or to create plans, then it makes new plan by interacting with the user or the other agents. The new plan and its request are stored as a new case into case base.

Overview of Assistant Agent System

The agent system assists user's work such as information retrieval, agent-based contact with other users, and meeting scheduling. This system has following functions.

- Using case-based reasoning (CBR), the agent has flexibility to handle requests. CBR is an approach to problem solving referring the similar old cases. In

this paper, the assistant agent stores previous problems and the plans to solve it in the case base, and then it uses the stored case (problem) to solve a new problem.

- For information retrieval, the assistant agent uses a HERIS (HEterogeneous Resource Information Search) agent [3]. This agent obtains information by searching both the Web and human groups. If the user wants to contact with the person that HERIS agent found, the assistant agent try to connect between them.
- To manage a contact request from other users, the agent should take account of the state of the user. For example, if the user is busy, it seems that the agent should not connect the user with the person. Therefore, the agent considers the state of the user, the person's social position and the importance of the request for intelligent management for connecting.

Structure of Assistant Agent System

Figure 1 shows the overview of the assistant agent system. This system contains two kinds of agents: an assistant agent and a HERIS agent. These agents are owned by each user. The assistant agent consists of four modules: a request receipt module, a planning module, an executing module, and a case storage module. The following explains the HERIS agent and the modules of the assistant agent in detail.

HERIS Agent

This agent searches information in the Web as well as in the human groups. First, the user sends a query as a keyword list to the agent. Then the agent searches information by a search engine and knowledge profile for each member. As a result, the agent shows a hit list of URLs and a list of members who can answer the query.

Request Receipt Module

This module receives a request from the user or other agents. The user makes a request by filling out a template. Main items of the template are:

- Request Type
- Purpose
- Date
- Place
- Deadline
- Addressee

These items are tagged by the user, and the request is transformed into a XML form. For example, let a request be ``We will have a meeting next week. So I want to know laboratory member's schedules from Sep. 19 to Sep. 28. Its deadline is Sep. 14." This request is transformed into the following form by interact with the user.

```
<Request>
 <Gather_Information>
  <Person>
   <Student>
    <Member>Agent lab. </Member>
   </Student>
  </Person>
  <Object>
   <Schedule>
    <Start_Date>Sep. 19</Start_Date>
    <End_Date>Sep. 28</End_Date>
   </Schedule>
  </Object>
  <Purpose>Meeting</Purpose>
  <Reply_Deadline>
    Sep. 14
  </Reply_Deadline>
 </Gather_Information>
</Request>
```

This request can be described in another form [4]

```
Request=Gathering_Information(
    Person=Student(Member=Agent lab.),
    Object=Schedule(Start_Date=Sep. 19,
    End_Date=Sep. 28),
    Purpose=Meeting,
    Reply_Deadline=Sep. 14)
```

For simplification, we describe a request or a case in this form in some cases.

Planning Module

This module makes a plan to handle the received request referring to old cases. The plan is a sequence of basic operations, which are defined in advance.

Case Representation

Cases consist of a request, an operation sequence and a user's evaluation.

```
Case = (Request, Operation Sequence, Evaluation)
```

The operation sequence is the sequence of some basic operations and/or some requests. The request in the operation sequence is transformed into a operation sequence based on the cases repeatedly. Finally, we can get a sequence of basic operation.

In this system, cases are also written in XML. An instance of the case is presented below.

```
<Case>
 <Request>
```

```
  <Gathering Information>
   <Person>
    <Student>
     <Member CT>$M: Agent lab.</Member>
    </Student>
   </Person>
   <Object>
    <Schedule>
     <Start Date LT>$SD: May 5</Start Date>
     <End Date LT>$ED: May 20</End Date>
    </Schedule>
   </Object>
   <Purpose> CT>$Pur </Purpose>
   <Reply Deadline LT>
    $RD
   </Reply Deadline>
  </Gathering Information>
</Request>

<Operation Sequence>
 <List>
  <Input>
   <Person>
    <Name> $N </Name>
    <Member> $M </Member>
    <Address> $Add </Address>
   </Person>
  </Input>
  <Output>
   <Addressee>
    <Name> $N </Name>
    <Address> $Add </Address>
   </Addressee>
  </Output>
 </List>

 <Request>
  <Gather Data>
   <Object>
    <Schedule>
     <Start Date> $SD </Start Date>
     <End Date> $ED </End Date>
    </Schedule>
   </Object>
   <Reply Deadline>
    $RD
   </Reply Deadline>
   <Addressee>
    <Address> $Add </Address>
   </Addressee>
   <Purpose> $Pur </Purpose>
  </Gather Data>
 </Request>
</Operation Sequence>

 <Evaluation> 5 </Evaluation>
</Case>
```

In this form, $ means variable which can be substituted for. For example, we can substitute <Name> Tom </Name> for <Name> $N</Name>. And some tags have attributes (i.e., CT, LT) that are used for calculating the similarity between a request and a case. The detail of the similarity is described in the following section.

Generating Operation Sequence

An operation sequence is generated by the following step.

1. Compare the request with old cases.
2. Select the case that is the most similar to the request.
3. Generate an operation sequence by substituting the constant in the request for the variable in the case.
4. If the operation sequence consists of only basic operation then it is the plan to handle the request. Else replace the current request with the request in the case, and return step 1.

Here, we explain how the agent generates an operation sequence by using the previous example case. The example case has two operations (List, Gather Data). List is a basic operation, however Gather Data is a request. Then Gather Data is processed as a new request. If Gather Data matches a case, the agent can obtain a new operation sequence.

In this way, the agent generates a sequence of basic operations by matching the request with the case repeatedly. Finally, this module sends the sequence as the plan to the executing module.

Intelligent management for connecting

We describe how the agent manages the contact request from other users. The agent makes decision to connect with the user using the idle time of the user, user's schedule, importance of the contact person and the purpose of contact.

Idle time means the time that the user does not use PC. From idle time *IT*, we define the user's state as the following.

$$User's\_state = \begin{cases} presence & (IT < 3) \\ short\_absence & (3 \le IT < 60) \\ long\_absence & (60 \le IT) \end{cases}$$

The importance of the person is evaluated according to the five-grade system. The evaluation is given by the user based on his/her subjectivity in advance.

From the information described above, the agent selects a next action out of 'Connect', 'Inform busyness' and 'Inform absence'.

Example of basic operation

We present some examples of basic operations below.

List

- Input = Person (Name=$N, Member=$M, Address=$Add)
- Output = Addressee (Name=$N, Address=$Add)
- Function: The agent makes the name list of members in $M. The name list consists of the name and his/her address.

Make Massage

- Function: From input information, the agent generates a message and its ID.

Send Message

- Input = (Message= $Mes, ID=$MID, Addressee (address=$Add))
- Function: The agent sends the message $Mes(ID=$MID) to the address $Add.

Receive Reply

- Input = Reply (ID=$MID)
- Output = Reply (Message=$Mes)
- Function: The agent receives a reply (ID=$MID) and shows user it.

Call search Agent

- Input = (Keywords = $Key)
- Function: The assistant agent sends keywords $Key to a search agent, and then receives a result of search.

Executing Module

This module executes the operation sequence generated by the planning module. However, the agent needs interaction with the user in the following cases.

- The basic operation requires user's decision making.
- The planning module cannot find a similar case.
- The executing module fails to execute the basic operation.

In the case 2 and case 3, the agent makes a new plan by interacting with the user.

Case Storage Module

This module stores the new case into case base. To form the new case, the user adds the evaluation score to the request and the operation sequence.

Table 1-Matching Condition

Tag Type	Case	Request (b)	
NA	Constant	a, $X:a	If a=b, matching succeed
	Variable	$X	Matching succeed.
CT	Constant	a, $X:a	If $S_w(a,b)=1$, matching succeed.
	Variable	$X	Matching succeed.
LT	Constant	a, $X:a	Matching succeed.
	Variable	$X	Matching succeed.

Searching for Similar Cases Based on XML

In this section, we discuss similarity between a request and a case. Since they are written in XML in this system, the agent can easily match them.

Tag Attribute

In order to match a request with a case written in XML, the agent calculates the similarity considering tags and tagged words. For this calculation, we should take into account the importance of the tags. For example, the tag that includes a variable for operation is important to make a concrete plan. Therefore, in this system, some tags for a case have attributes, which mean the importance of the tags. There are two types of tag attributes:

- **CT (Critical Type)** – This type means that the tag is critical for matching. If a request does not have the tag that is critical type in the case, then the request cannot match with the case.
- **LT (Loose Type)** – This type means that the tag is loose for matching. Even if a request does not have the tag that is loose type in the case, the request can match with the case.

Definition of Similarity

Here, we define the similarity between a request and a case. First, we discuss the condition to match a request with a case.

- **NA (No Attribute)** – Matching succeed, iff the tagged words in the request and case are same.
- **CT (Critical Type)** – Matching succeed, iff the tagged words in the request and case are similar.
- **LT (Loose Type)** – Matching succeed in any case.

This is summarized in Table 1.

We define the similarity between words using thesaurus. The thesaurus used in this system is the concept classification dictionary in EDR [5]. This dictionary includes the concept classification information, which describes super-sub relation between concepts. The distance between concepts is defined as the distance of the path

between them.

Let the distance in thesaurus between concepts, w_1 and w_2 be D. We define $S_w(w_1, w_2)$, the similarity between w_1 and w_2 as:

$$S_w(w_1, w_2) = \begin{cases} 1 & (0 \le D < 2) \\ 0.5 & (2 \le D < 4) \\ 0 & (4 \le D) \end{cases}.$$

Now we define *Sim*, the similarity between a request and a case. Let the number of NA, CT and LT tags be N_{NA}, N_{CT} and N_{LT}, respectively.

$$Sim = \frac{N_{NA} + N_{CT} + \sum S_w}{N_{NA} + N_{CT} + N_{LT}}$$

Using this similarity, the agent can find the most similar case. If the similarity of this case is less than threshold, the agent fails to find a similar case.

Conclusion

We introduced an agent system to assist user's network-based work. This system handle user's request referring old cases written in XML. XML enables the agent to match a request with a case easily. The agent can handle more various kinds of user's requests with the increase of the number of cases.

References

[1] P. Maes and R. Kozierok, 1993. Learning Interface Agents. In Proceedings of the Eleventh National Conference on Artificial Intelligence, 459-465.

[2] K. Kida, K. Yoshifu, T. Asakura and T. Miyashita, 1999. Goodinator: Quick and Prudent Meeting Coordination with a Multi-Agent System. In Proceedings of the first Asia-Pacific Conference on IAT 364-373.

[3] S. Yamada and M. Mase, 2000. Integrated information search in the WWW and a human group. In Proceedings of INFORMS / KORMS 2000.

[4] A. Hassan and N. Roger, 1986. LOGIN: A Logic Programming Language with Built-In Inheritance. The Journal of Logic Programming, 185-215.

[5] Japan Electronic Dictionary Research Institute, Ltd. EDR Electronic Dictionary. http://www.iijnet.or.jp/edr/.