Automatic Filling in a Form by an Agent for Web Applications

Takeshi Chusho, Katsuya Fujiwara, and Keiji Minamitani
Department of Computer Science,
School of Science and Technology, Meiji University
Kawasaki, 214-8571, Japan
chusho@cs.meiji.ac.jp

Abstract

The number of end-users using the Internet increases on the inside and outside of offices. Enduser-initiative development of applications has become important for automation of their own tasks. This paper describes a multi-agent framework for the MOON(multiagent-oriented office network) systems which are distributed systems including window work in electronic commerce. Especially, since the window work is considered as a metaphor of the interface between service providers and the service requesters for web services, this paper is focused on automatic filling in a form by a user agent in collaboration with a broker agent, which is one of key technologies for electronic forms defined as HTML documents. At the first step, a satisfactory success rate is achieved by using cognitive rules based on a form layout. That is, the case rule of {*IF #case THEN #action* } *is introduced. The #case is composed of the four attributes and their values based on cognitive information on the upper, left, right and lower sides of the input field. The ontology of concept names is introduced for different expressions of the same meaning. Furthermore, by the reasoning of similarity on incomplete matching of the case parts of rules, the success rate of automatic filling is greatly improved. At the second step, the success rate is improved further to 100% by using experiential rules of other users, which the broker agent gathers up.* [1]

Key words : software agent, multi-agent, web application, rule, reasoning

[1]K. Fujiwara and K. Minamitani are with Akita University and Hitachi, Ltd. at present respectively.

1. Introduction

The number of end-users using the Internet increases on the inside and outside of offices. Enduser-initiative development of applications has become important for automation of their own tasks. This trend promotes the following our philosophy: “All routine work both at office and at home should be carried out by computers.”

As the solution based on CBSE(Component-Based Software Engineering) [2], this paper describes a form-based approach for end-user computing under distributed systems. As a typical distributed information system, we direct our attention to an application system for windows or counters in banks, city offices, travel agents, mail-order companies, etc.

In the near future, the information society will require such new technologies that domain experts can automate their own work by themselves and that almost all clients can operate computers at home or at office without extra training or without the help of others.

Multi-agent systems must be the solution for these problems because end-users may teach their operations to agents without programming [1, 9, 10]. Actually, multi-agent systems are used for advanced applications based on distributed systems and the Internet such as electronic commerce support systems [11]. An agent communication language(ACL) is one of the key technologies for interactions among independently-developed applications with agents. Then the standardization is being tried by FIPA(Foundation for Intelligent Physical Agents) [7] and OMG(Object Management Group) Agent WG [12].

Especially, since the window work is considered as a

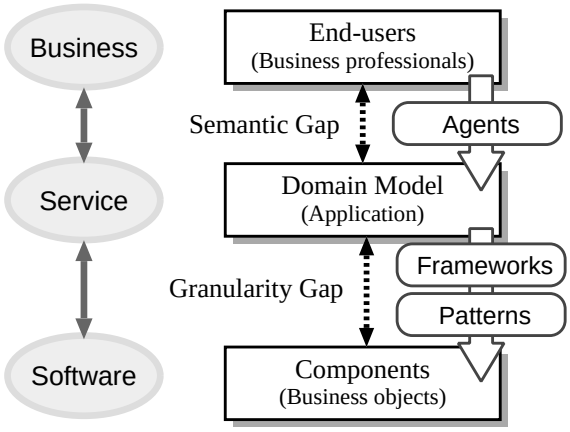

Figure 1. Technologies for bridging gaps between end-users and components.

metaphor of the interface between service providers and the service requesters for web services, this paper is focused on automatic filling in a form by a user agent in collaboration with a broker agent, which is one of key technologies for electronic forms defined as HTML documents. At the first step, a satisfactory success rate is achieved by using cognitive rules based on a form layout. At the second step, the success rate is improved by using experiential rules of other users.

This paper presents the multi-agent framework in Section 2, automatic filling in a form in Section 3, cognitive rules in Section 4 and experiential rules in Section 5.

2. Multi-Agent Framework

2.1. Basic concepts for web application

For web applications, the following two features are considered to be essential:

(1) Rapid development and continuous variation.

(2) End-user initiative.

Business based on Internet technologies, is rapidly changed. For this reason, the application development period from defining a new business model through releasing new services, must be short. Furthermore, after the release, the application should be maintained continuously as the business world changes. Conventional ways for application development and maintenance by system engineers, are not suitable because of no timeliness. For “just-in-use,” enduser-initiative development is required.

Our approach to how to make web applications is shown in Figure 1 [5]. The three layers of the left zone and the central zone imply the abstract level and the concrete level respectively. The right zone implies technologies. The business model at the business level is proposed by end-users. Then, at the service level, the domain model is constructed and the required services are specified. That is, the requirement specifications of the application for the business model are defined. At the software level, the domain model is implemented by using components to be combined.

In this approach, there are two technological gaps, that is, the granularity gap between components and the domain model, and the semantic gap between the domain model and end-users. The granularity gap is bridged by business objects, patterns [8] and application frameworks [6] based on CBSE(Component-Based Software Engineering). On the other hand, the semantic gap is bridged by multi-agent systems.

2.2. A form as a metaphor for web services

As a typical distributed information system, we direct our attention to application systems for window work. Such window work is not limited to the actual window work in the real world. For example, in a supply chain management system (SCM), exchanges of data among related applications can be considered as the virtual window work.

Generally, window work or counter work is considered as service requests from clients to service providers. Forms to be filled in are considered as the interface between them. That is, the following concept is essential for our approach:

”One service = One form.”

2.3. Application architecture

Agent-based applications are constructed on a multiagent-oriented office network (MOON) for window work [3]. The MOON system is based on a client/server model and is partitioned into the following three parts as shown in Figure 2:

1. Client terminals with client agents for sending written applications to windows, such as personal computers

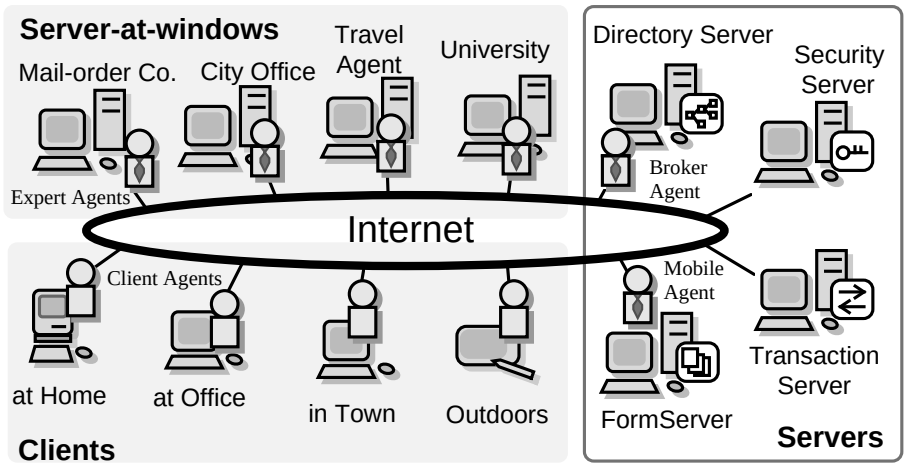

Figure 2. A MOON(multiagent-oriented office network) system.

and workstations both at home and at office, public telephones with terminals in town, portable computers for mobile computing outdoors, etc.

2. Server-at-windows with expert agents for receiving written applications, such as windows in mail-order companies, city offices, travel agents, universities, etc.

3. MOON servers for managing the network system.

The MOON servers imply the following four servers and some of them may be located physically in server-at-windows:

1. A directory server with a broker agent manages network addresses of server-at-windows as a service directory of windows.

2. A form server with a mobile agent manages various application forms for services at these windows, which forms are defined with help messages and selection menus by domain experts.

3. A transaction server stores written applications received by server-at-windows with the identification numbers, manages the states of the process and replies to inquiries about the states. It may be connected with a workflow system in the organization including the server-at-window.

4. A security server controls access rights to server-at-windows and the MOON servers, and manages authentication of clients.

In our experiences of prototyping, the actual system configuration is the 4-tier architecture of browsers, web servers, application servers and DB servers. The front end of the system is supported by application frameworks and multi-agents. The back end is supported by domain modeling and business objects [4].

2.4. Features of agent-based applications

A customizable multi-agent system is developed as an application framework which implies a reusable semi-complete application that can be specialized to produce custom application. Then the customization of the hot spots in the application framework implies the agent development by end-users.

The first feature is electronic form processing which is navigated by agents both in client terminals and in server-at-windows. Clients can teach the fixed operations of filling in a form about such plain words as their names, addresses and phone numbers to their agents. Then their agents do so instead. Domain experts can teach their expertise to their agents. Then the agents guide clients in filling in the form and check the written form.

The second feature is standardization of ACL for communication between client agents and expert agents. De-

sign of ACL depends on features of multi-agent systems. This paper describes cooperative multi-agent systems.

In our previous work, intelligent navigation by expert agents was implemented in XML base [5], and a form-based agent communication language(FACL)[3] was developed.

In the remainder of this paper, technologies for automatic filling in a form by a client agent in collaboration with a broker agent, are described.

3. Automatic Filling in a Form

3.1. Conventional approaches

There are several conventional systems for automatic filling in a form. Some of them use predefined rules for the particular input fields of text [13]. This method has limitations of the number of predefined rules. The others use the context around the input field[14]. This method has limitations of the accuracy since the incorrect input is often entered.

A lot of people may be familiar with the auto-complete feature in the Internet Explorer. It helps them to fill in the blanks by showing the candidates which are recorded based on past experiences. This feature is sometimes useful but not always.

In many cases, the value of the name attribute in the input field of the HTML document, is checked. For example, the system selects the same text which was entered once in the input field with the same value of the name attribute. However this value is not always believable because the HTML document designer uses the arbitrary text as the value of the name attribute.

3.2. Basic problems

Basically, a domain expert builds expert agents by form definitions while teaching their expertise. Each expert agent is a mobile agent also since the form is sent from the form server to a client terminal and return to the transaction server. On the other hand, the directory server is a broker agent since a client agent asks about suitable expert agents.

The client agent is independent of the expert agent, the mobile agent and the broker agent. The client agent has facilities for automatically filling in the form. The knowledge is classified into two categories. One is knowledge on the owner itself such as a name, an address, a phone number and a birth day, which is independent of each form. The other is knowledge on each form such as a member number and a grade of membership, which is dependent on the server-at-window of the form.

As for automatically filling in the form of the first category, some intelligence is required for different expressions of the same meaning, such as ”Phone” and ”TEL.” This problem was solved by introducing such concept names as @name, @address, @phone and @birthday. The label name of an input field in a form is translated into the concept name. Then the input field is filled with the individual value corresponding to the concept name. For example, both ”Phone” and ”TEL” are translated into @phone. Then the input field is filled with the actual phone number of the owner.

As for the second category, some intelligence is required for the same expressions of the different meaning, such as a ”member number” of IEEE or a ”member number” of ACM. When there are two mapping rules for the concept name of @mno, the selection depends on the context in the form including the label name of ”member number.” For this reason, the mapping rules have constraints which are taught by the owner.

Actually, these two kinds of problems may happen in both categories although the solutions are same. For example, @address may have two values of a home address and an office address. A ”member No.” may be used in a form instead of a ”member number.”

3.3. Cultural problems

There are some characteristic problems of Japanese language. One of problems is that there are many different expressions of the same meaning, which are used as label names for input fields. The name is the typical example. As

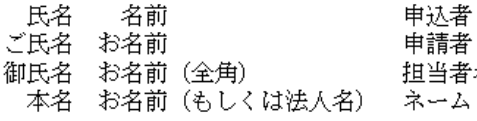

Figure 3. A part of different expressions for a name in Japanese (twelve examples).

a part of different expressions for the name, twelve examples are shown in Figure 3.

The other problem is that there are many types of input data, that is, Chinese characters, the Japanese cursive syllabary(hiragana), the square form of hiragana(katakana) and English letters. Furthermore, there are two kinds of character codes for katakana and English letters. That is, the other code for katakana is used for dealing it as same as a English letter of 8 bits and the other code for a English letter is used for dealing it as same as a Chinese character of 16 bits. There are these two kinds of character codes for Arabic numerals also.

In many forms, the type and the character code are specified. As for the name and the address, in addition to input data in Chinese characters, input data in either katakana or hiragana are often requested because both katakana and hiragana indicate the pronunciation of the name in Chinese characters. When these specifications are expressed in a form, there are a lot of variations and furthermore the note for a target input field may be described in the upper, left, right or lower side of the target field.

These factors limit the success rate of automatic filling for forms in Japanese.

3.4. Kinds of rules

At present, almost all electronic forms are defined as HTML documents which have a critical defect of lack of semantic information. Therefore, as representation of information exchanged between distributed applications, XML-base documents have begun to attract notice recently. This is because it is possible for applications to process information easily by using predefined metadata on semantics. In B-to-B, some XML-base documents have already been put into practical use[15, 16]. In the future, XML-base documents may replace a lot of HTML documents.

For the present, however, electronic forms in HTML will be used mainly. Then this paper describes how to fill automatically in a form in HTML. There are two categories of rules for automatic filling as follows:

- Cognitive rules
- Experiential rules

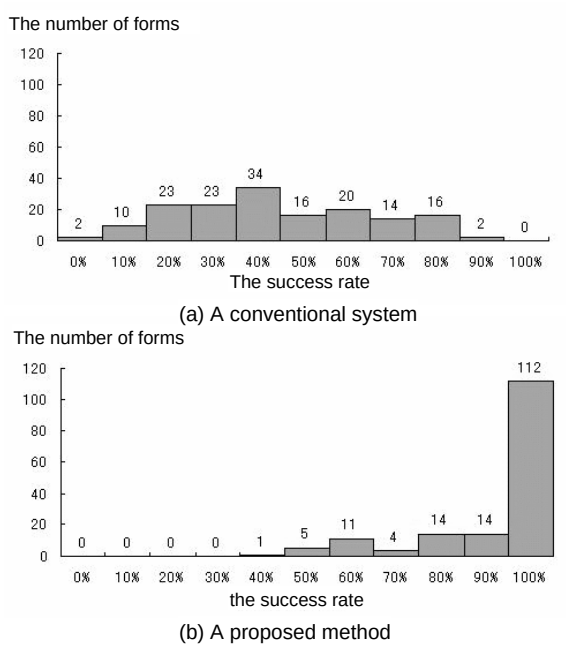

Figure 4. The relation between the success rate and the number of forms.

The cognitive rules are defined based on cognitive information of displayed forms. The experiential rules are defined based on experiences of other users' past behavior.

4. Cognitive rules

4.1. Cognitive information

When users fill in a new form which they never once did so, they must understand what should be described in each input field. For this purpose, users get such cognitive information from the context of the input field as the label of "address" on the left side of the field, the label of "name" and another input field on the upper side, and the label of "phone" and the other three input fields on the lower side.

Therefore this paper assumes that it must be effective to use information about the four sides of the target input field for cognitive rules. An experiment was performed for confirming this assumption. That is, the number of input fields in which exact data can be filled by using the above cognitive information, was counted.

The 160 sample forms to be tested, were gathered from

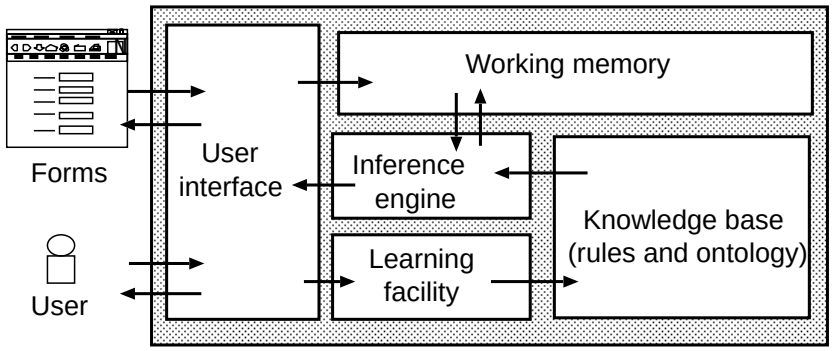

Figure 5. The system architecture of the user agent for automatic filling.

the Internet. Among them, there were 1,914 input fields for personal information.

First, these samples are used for the conventional system of InternetExplorer 5 for Mac which supports automatic filling facilities. The result is shown in Figure 4(a). The success rate is calculated for each form, that is, the percentage of the number of fields with success in automatic filling among all fields in each form. In the figure, the horizontal axis implies the success rate and the vertical axis implies the number of forms corresponding to the rate.

Next, the same samples are used for the proposed method. The result is shown in Figure 4(b). The average success rate is 87% since the number of input fields with success is 1,657. On the other hand, the average success rate for the conventional system is 31%. Consequently, it is confirmed that it is effective to use information about the four sides of the target input field for cognitive rules.

4.2. System architecture

The system architecture of the user agent for automatic filling is shown in Figure 5. The user agent implies a client agent in Figure 2,

Basically this is a rule-base system. First, the related information about a target form is described into the working memory. The inference engine fills in a form automatically by using the knowledge base and the working memory. Sometimes the learning facility enhances the knowledge base. The details will be described later.

4.3. Knowledge representation

The ontology of concept names is introduced for different expressions of the same meaning as mentioned before. For example, both label names of ”Phone” and ”TEL” are translated into the same concept name of @phone. The user profile based on these concept names is defined in advance by the user and is stored with the ontology into the knowledge base. That is, the value corresponding to each concept name is registered via the learning facility in Figure 5.

The primitive case rule using cognitive information is considered as follows:

IF #case THEN #action

The #case is composed of the four attributes and their values. That is, the UPPER, LEFT, RIGHT and LOWER imply information on the upper, left, right and lower sides of the input field respectively. The #action specifies the concept name. The value corresponding to this concept name is extracted from the user profile, and is described into the input field. An example of a primitive rule is shown as follows:

IF (UPPER: (“Address” TEXTFIELD), LEFT: ”Phone”, RIGHT: NONE, LOWER: (“Email” TEXTFIELD)) THEN @phone

Attributes	Values
UPPER	“Address”, TEXTFIELD
LEFT	”Phone”
RIGHT	
LOWER	“Email”, TEXTFIELD

Figure 6. An example of the working memory.

This rule is executed when the contents of the working memory as shown in Figure 6 matches with the condition part of the rule.

However this primitive rule is not flexible because the target input field does not match with this rule if the label on the left side of the field is "TEL." Therefore the abstract case rule is used instead. That is, the actual label name is replaced with the corresponding concept name as follows:

IF (UPPER: (@address TEXTFIELD), LEFT: @phone, RIGHT: NONE, LOWER: (@email TEXTFIELD)) THEN @phone

In the same way, the content of the working memory is replaced with the concept names. For example, different expressions shown in Figure 3, should be replaced with the concept name of @name. Consequently, the actual possibility depends on the ontology.

4.4. Feasibility studies

From 293 input fields of the name in the above-mentioned 160 forms, 240 abstract case rules with the action of the @name were extracted. Then these abstract case rules were applied for the 239 input fields of the name in 139 forms other than the above-mentioned 160 forms. The result is, however, not satisfied because only 63 fields, that is, 26%, were successful in automatic filling. The other 176 fields were not filled in with anything.

4.5. Extension of reasoning

In the feasibility study, the automatic filling is performed only if the cognitive information of four attributes of the target field matches completely with the case part of some rule.

In actual cases, however, there are a lot of variations of a case part which implies the same action. Therefore the reasoning of similarity on incomplete matching of the case part of rules, is introduced. The following procedure was developed:

1. The matching is performed for each one of the four attributes of the case part, that is, four times for each rule.
2. The number of matching with the same action is counted for each attribute and for each action.
3. The relative frequency of each action for each attribute is calculated.
4. The average of the relative frequencies for the four attributes is calculated as a certainty factor for the action.
5. The action with the maximum certainty factor is selected.

For the attribute i $\{i = 1, 2, 3, 4\}$, the relative frequency of the action j $\{j = 1, 2, \cdots, M\}$ is defined as follows:

$$O_{ij} = \frac{n_{ij}}{\sum_{k=1}^{M} n_{ik}}$$

, where n_{ij} implies the number of rules with the action j which matches with the attribute i. The certainty factor for the action j is defined as follows:

$$CF_j = \frac{\sum_{i=1}^{4} O_{ij}}{4}$$

The action with the maximum certainty factor is selected.

By applying this reasoning to the above-mentioned 239 input fields, the other 113 fields became successful in automatic filling in addition to the previous 63 fields. Consequently, the success rate of automatic filling is greatly improved from 26% to 74%. The problem to be solved, however, remains that 62 fields of 63 unsuccessful fields were filled in with wrong values.

4.6. Analysis of unsuccessful fields

The failure factors on the above-mentioned unsuccessful fields are considered as follows:

- lack of rules
- lack of keywords registered with the ontology

In the first experiment by using complete matching of abstract rules, there are 176 unsuccessful fields which were not filled in with anything. In detail, 168 unsuccessful fields are caused by the lack of rules and 8 unsuccessful fields are caused by lack of keywords to be translated into the concept names.

On the other hand, in the second experiment by using the certainty factor, there are 63 unsuccessful fields. In detail,

57 unsuccessful fields are caused by the lack of rules and 6 unsuccessful fields are caused by lack of keywords to be translated into the concept names.

These problems are solved by learning of the agent through the learning facility. For improving lack of rules, the agent can acquire new rules by monitoring what the user fills in the target field with, in which the agent could not fill with the correct value.

For improving lack of keywords, the agent can acquire new keywords as well as new rules. In comparison with the rule acquisition, however, it is difficult to decide the correspondence of the new keyword to the concept name. In most cases, the keyword on the left side of the target field corresponds to the concept name of the actual value inputted. Sometimes, however, it does not so, and one of the keywords on other sides may so. Therefore the agent inquires of the user whether the keyword on the left side of the target field corresponds to the concept name of the actual value inputted. It seems that it is easy for the user to ask this question.

5. Experiential rules

5.1. Automatic rule generation

At the second step, a broker agent is introduced and gathers experiential rules of other users up for improvement of the success rate. In Figure 2, there is the broker agent on the directory server. The first task is the directory service on forms. The broker agent answers where the form is when a user inquires about a necessary form for some service request.

The second task of the broker agent is the management of experiential rules. The experiential rules are gathered in collaboration with user agents as follows:

1. A user agent inquires of the broker agent about a necessary form and gets it from some URL in response to the user request.

2. The broker agent sends the experiential rules on the form to the user agent.

3. The user agent fills in the form automatically by using these rules.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf=
"http://www.w3.org/1999/02/22-rdf-syntax-ns\#"
 xmlns:o="http://wwhww.org/schemas/userprofile/1.0/"
 xmlns:="http://wwhww.org/1.0/">
<FormItem rdf:about="http://www.se.cs.meiji.ac.jp/
library/entry/\#form[entry].item[name]">
  <history>
    <Profile amount=10>
      <value><o:User.Name.First /></value>
      <separator> </separator>
      <value><o:User.Name.Last /></value>
    </Profile>
  </history>
</FormItem>
</RDF>
```

Figure 7. An example of an experiential rule.

4. The user fills in the blank fields without automatic filling. For the fields with automatic filling, the user check and modified them if necessary.

5. The user agent sends the form to the window. Furthermore the user agent sends the broker agent the information about fields in the form, that is, which ones of the fields with automatic filling are modified or not, what values are inputted into the fields modified, and what values are inputted into the blank fields.

Actually, communication between the user agents and the broker agent are performed by using the concept names instead of actual values because an experiential rule implies that some field corresponds to a concept name. The user agent replaces the actual value with the concept name by using a user profile.

The form-based agent communication language, FACL [3], designed for the MOON system shown in Figure 2, was implemented in XML. A description language of experiential rules is designed in XML also. An example of an experiential rule is shown in Figure 7. This example implies that both the first name and the last name have been inputted ten times into the name field of the form at http://www.se.cs.meiji.ac.jp/library/entry/.

5.2. Feasibility study

For an experiment, the 50 forms on the Internet were selected. They includes 738 fields. Among them, the items

to be tested are 497 fields related to the personal information on a name, an address, a telephone, a fax, a birthday and an email address.

The first testee filled in the all fields manually in the initial state that the broker agent had no experiential rules. As a result, the 531 experiential rules were extracted.

When the second testee begins to fill in, the 531 fields have already filled in automatically. Among them, the 497 fields to be tested were correct. The other 34 fields, however, were incorrect. This is because the 34 wrong rules had been extracted from the first testee's behavior. That is, the causal coincidences happened that the values of the corresponding concept names in the user profile were the same as the value inputted into the different fields. For example, the month of the birthday was the same as the month of the graduation date. The second testee modified these incorrect value and filled in the other blank fields.

Before the third testee begins to fill in, the threshold value was set on the broker agent's behavior that an experiential rule was not adopted if the number of rules recommending the same concept name is not more than 50 % among the total number of rules recommending any concept name with respect to some field. As a result, while the 497 fields to be tested were correct, the number of fields with incorrect automatic filling reduced to 4 fields.

Consequently, in this experiment, the success rate was improved to 100% for the 497 items to be filled in automatically although there were the four incorrect fields not to be filled in automatically.

6. Conclusions

For enduser-initiative application development of web applications, the multi-agent framework was proposed. In particular, as the front end system of a web application, the user agents and the broker agent for automatic filling in a form were developed. Some feasibility studies confirmed the effectiveness of both the abstract case rules of cognitive rules and experiential rules of other users.

References

[1] Bradshaw, J. M., "An Introduction to Software Agent," Software Agent, MIT Press, pp.3-46, 1997.

[2] Brown(Ed.), A. W., "Component-based software engineering," IEEE CS Press, 1996.

[3] Chusho,T. and Fujiwara, K., "FACL : A Form-based Agent Communication Language for Enduser-Initiative Agent-Based Application Development," COMPSAC2000, IEEE Computer Society, pp.139-148, Oct. 2000.

[4] Chusho, T., Ishigure, H., Konda, N. and Iwata, T., "Component-Based Application Development on Architecture of a Model, UI and Components," APSEC2000, IEEE Computer Society, pp.349-353, Dec. 2000.

[5] Chusho, T., Fujiwara, K., Ishigure, H. and Shimada, K., "A Form-based Approach for Web Services by Enduser-Initiative Application Development," SAINT2002 Workshop (Web Service Engineering), IEEE Computer Society, pp.196-203, Feb. 2002.

[6] Fayad, M. and Schmidt, D. C. (Ed.), "Object-Oriented Application Frameworks," Commun. ACM, Vol. 39, No. 10, pp. 32-87, Oct. 1997.

[7] FIPA, "Agent Communication Language," FIPA Spec 2-1999, Draft ver.0.1, Apr. 1999.

[8] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., Design Patterns, Addison-Wesley, 1995.

[9] Griss, M. L., and Pour, G., "Accelerating Development with Agent Components," IEEE Computer, Vol. 34, No. 5, pp.37-43, May 2001.

[10] Jennings, N. R., "An Agent-Based Approach for Building Complex Software Systems," Commun. ACM, Vol. 44, No. 4, pp.35-41, Apr. 2001.

[11] Maes, P., Guttman, R. H. and Moukas, A. G., "Agents That Buy and Sell," Commun. ACM, vol.42, no.3, pp.81-91, Mar. 1999.

[12] OMG Agent Working Group, "Agent Technology, Green Paper," OMG Document no. ec/99-12-02, Dec. 1999.

[13] The Gator Co., `http://www.gator.com/`

[14] Microsoft Co., Microsoft InternetExplorer 5 Macintosh Edition, `http://www.microsoft.com/mac/products/ie/`

[15] RosettaNet, `http://www.rosettanet.org/`

[16] W3C Note, "Simple Object Access Protocol (SOAP) 1.1 ," May 2000, `http://www.w3.org/TR/SOAP/`