An Interface Agent for Wrapper-Based Information Extraction

Jaeyoung Yang[1], Tae-Hyung Kim[2], and Joongmin Choi[2]

[1] Openbase Inc., Seoul, Korea
jyyang@openbase.co.kr
[2] Department of Computer Science and Engineering,
Hanyang University,
Ansan, Kyunggi-Do 426-791, Korea
{tkim,jmchoi}@cse.hanyang.ac.kr

Abstract. This paper proposes a new method of building information extraction rules for Web documents by exploiting a user interface agent that combines the manual and automatic approaches of rule generation. We adopt the scheme of supervised learning in which the interface agent is designed to get information from the user regarding what to extract from a document and XML-based wrappers are generated according to these inputs. The interface agent is used not only to generate new extraction rules but also to modify and extend existing ones to enhance the precision and the recall measures of Web information extraction systems. We have done a series of experiments to test the system, and the results are very promising.

1 Introduction

Information extraction is the task of obtaining a particular fragment of a document that is relevant to the user interest. In general, information extraction is difficult mainly due to the heterogeneity inherent in different information sources. For example, Fig. 1 shows two different display styles of conference *CALL FOR PAPERS* pages, one with the list style and the other with the table style.

In order to cope with structural heterogeneity, the information-extraction systems usually rely on extraction rules tailored to a particular information source, often called the *wrappers*. In the previous methods of information extraction for semi-structured Web documents, wrappers are generated manually or automatically. In the manual wrapper generation method, the wrapper is written by a human expert who analyzes documents and builds rules. ARANEUS [1] and TSIMMIS [3] are the example systems that adopt the manual approach. The manual method reveals highly precise performance, but generally it is not scalable and effective.

On the other hand, the automatic method is known as the wrapper induction [4] in which the agent program analyzes a set of example documents and produces a wrapper through learning. Automatic wrapper induction can be based

M.W. Barley and N. Kasabov (Eds.): PRIMA 2004, LNAI 3371, pp. 291–302, 2005.

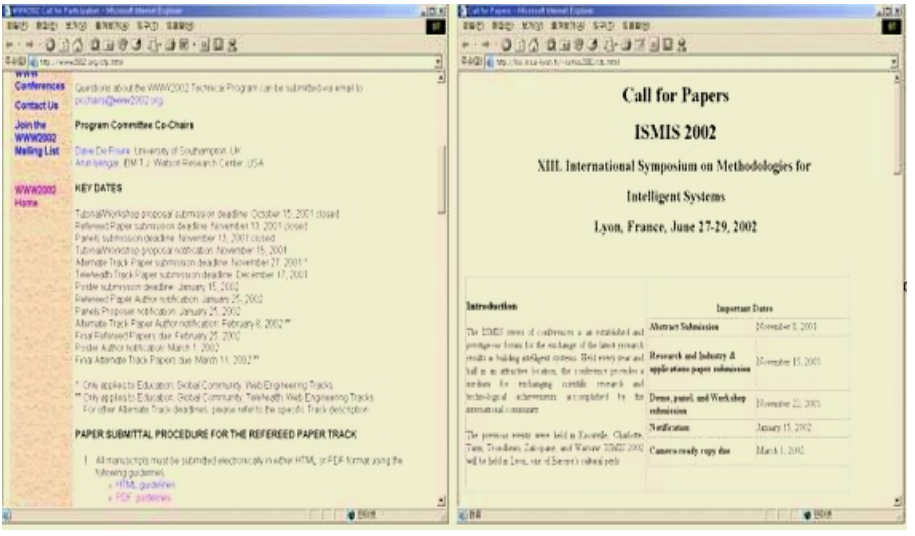

Fig. 1. Example of heterogeneous information sources.

on either heuristics or domain knowledge. Heuristic wrapper induction uses simple heuristics, e.g. *a dollar sign followed by a digit such as $250 denotes the price information*, to recognize the pattern of the target information. This approach has been adopted by most traditional wrapper-based information extraction systems such as SHOPBOT [2], STALKER [5], WHISK [6], and MORPHEUS [8]. Knowledge-based wrapper induction tries to solve the ineffectiveness of naive heuristics by defining and applying the domain knowledge during wrapper generation. Knowledge-based approach is expected to extract more features from the document than the heuristic approach. We have implemented knowledge-based information extraction systems named XTROS [9] and XTROS$^+$ [10]. Overall, the automatic wrapper-generation method is scalable and effective than the manual one, but it also has difficulty in generating correct rules due to the diversity and the heterogeneity of Web information structures. Also, the generated rules are sometimes unreliable.

To take advantage of both the preciseness of manual extraction-rule construction and the scalability of wrapper induction, this paper proposes a new method of building information extraction rules for Web documents through the user interface agent. This method combines manual and automatic approaches of rule generation. We adopt the scheme of supervised learning in which a user interface agent is designed to get information from the user regarding what to extract from a document and XML-based wrappers are generated according to these inputs. The interface agent is used not only to generate new extraction rules but also to modify and extend existing ones to enhance the precision and the recall measures of Web information extraction systems.

This paper is organized as follows. Section 2 presents our approach of information extraction system based on the interface agent. Section 3 defines the

format of XML-based extraction rules. The algorithms of rule learner and rule interpreter are described in Section 4. Section 5 assesses the system with some experimental results. Finally, Section 6 concludes with the summary and future direction.

2 Interface Agent

The system architecture of the information extraction system we are proposing is shown in Fig. 2. In this system, the user inputs are obtained and converted into training examples through the interface agent. The rule learner, also called the wrapper generator, analyzes the training data and produces information extraction rules. These rules play the role of the wrapper and stored in the rule repository. The rule interpreter executes the learned rules for real Web documents to extract the target information.

The interface agent acts as a mediator between the user and the rule learner in the task of recognizing the category of each data fragment to be extracted. In order to do this, the interface agent provides an environment in which the user can easily assign a category to a user-selected data item. As a result of a series of interactions via the interface agent, training examples are produced and supplied to the rule learner. The interaction between the user and the interface agent is accomplished by the drag-and-click action of the mouse and the popup menu, which enables the user to select a data item to be extracted and assign a category to it.

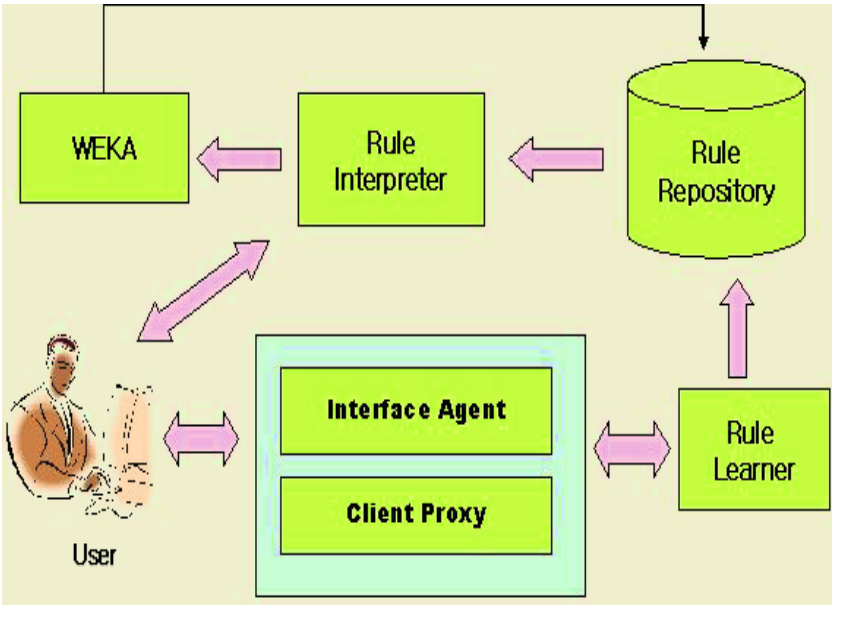

Fig. 2. The architecture of an information extraction system with an interface agent.

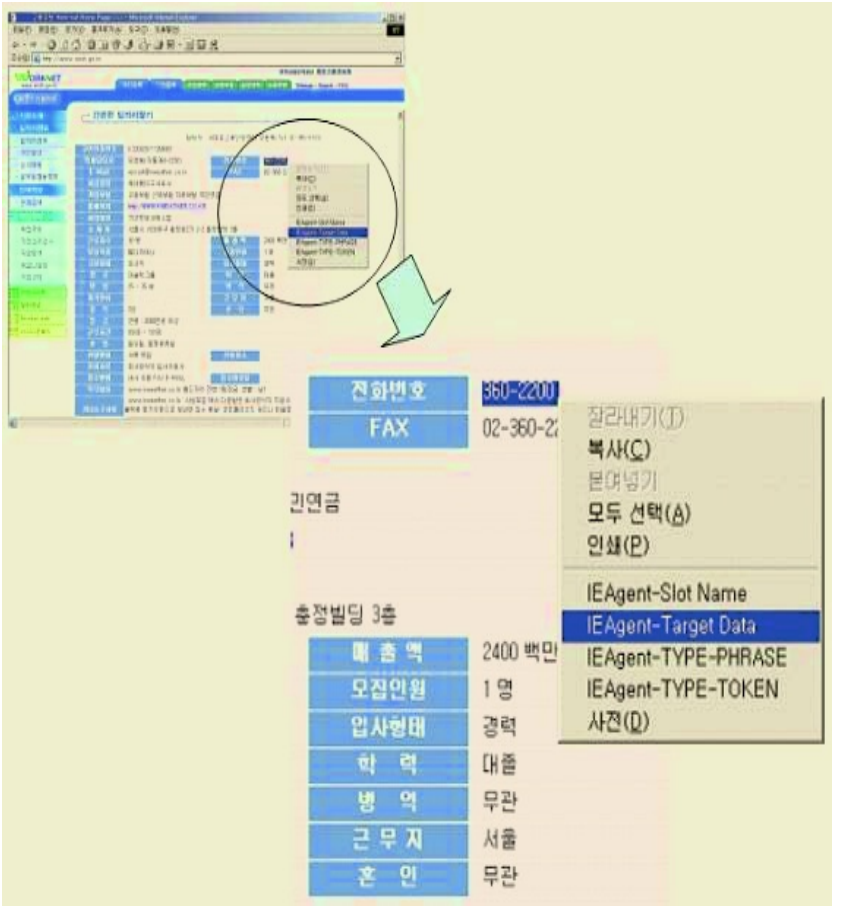

Fig. 3. Interface agents with popup menus.

A running example of the interface agent with the popup menu is shown in Fig. 3. The user first selects a data fragment from a Web document displayed in the interface by dragging it with the mouse. (In our example, a phone number is selected.) Then, the user clicks the right mouse button to bring up a special menu that enables the user to assign a category to the data item by selecting the *Slot-Name* menu and typing in a proper slot name in a dialog box. (In our example, *phone number* would be an appropriate slot name.) Finally, when the user selects the *Target-Data* menu, the data item is automatically associated with the slot name and stored as a training example. The interface agent also delivers the trigger information to the rule learner. A trigger is the symbol or word that helps to recognize the role of the target data item. In this example, the trigger for the *phone number* slot might be *TEL*, *Tel*, or *Telephone*. Triggers and slot names are determined according to specific application domains.

In short, the interface agent is required to have the following capabilities: recognition of mouse events initiated by the user, convenient display of Web

```
<Domain_name>
  <Slot_name>
    <pattern>
      <id> String </id>
      <type> TOKEN | PHRASE </type>
      <trigger> String </trigger>
      <format> [TRIG] String* TARG {String}* String* </format>
    </pattern>
        .. other patterns
  </Slot_name>
  .. other slots
</Domain_name>
```

Fig. 4. Format of information extraction rules.

documents, and acceptance of user feedback. To facilitate these functions, the interface agent is implemented by extending popup menus of a Web browser(in our paper, the Microsoft Internet Explorer) and by employing a proxy server. Extending popup menus to the browser is realized by adding new events to the context menu of the Internet Explorer in the MS Windows registry. Once extending the popup menus, the user input committed to the interface agent is delivered to the proxy server that relays information to the rule learner. Likewise, all the menu handling events are processed in the proxy server.

3 IE Rule Format

Information extraction rules are written in XML for readability and maintainability. The format of rules used in this paper is shown in Fig. 4.

An extraction rule corresponds to a single domain. A rule consists of a set of slots with different slot names, and each slot consists of a set of information patterns. An information pattern, denoted by the `<pattern>` element, is the smallest unit of a rule, and is obtained from the user input. Patterns are incrementally added to a slot as more user inputs come, and the learning is performed based on this pattern information.

The `<pattern>` element consists of several subelements including `<id>`, `<type>`, `<trigger>`, and `<format>`. Among these, the value for `<trigger>` is obtained from the user input, and other values are determined automatically by analyzing the data. The `<id>` element denotes the identifier of a pattern. The `<type>` element indicates the structural type of the target data, and is defined as either `TOKEN` or `PHRASE`. Here, `TOKEN` means the target data must be separable as a single token in the document sentence, and `PHRASE` means the target consists of multiple words and the correct data can be extracted by partial matching. The `<trigger>` element plays the role of delimiter to determine whether the current rule should be applied to the target sentences. Only one trigger is assigned for

```
<Job_offer>
  <Company>
    <pattern>
      <id> companyName01 </name>
      <type> TOKEN </type>
      <trigger> (주) </trigger>
      <format> [TRIG] TARG </format>
      <format> TARG [TRIG] </format>
    </pattern>
    ... other patterns
  </Company>
  <From_to>
    <pattern>
      <id> from_to01 </id>
      <type> PHRASE </type>
      <trigger> 접 수기간 </trigger>
      <format> [TRIG] : TARG {DATE} </format>
    </pattern>
    ... other patterns
  </From_to>
  ... other slots
</Job_offer>
```

Fig. 5. An example rule for job advertisement.

each pattern, but each slot may be associated with several triggers. This phenomenon handles the situation where particular information can be expressed differently with various terms. The `<format>` element describes the position of the target data and its structural characteristics. Here, `[TRIG]` is the reference to a `<trigger>` element in a pattern, `TARG` is the target information to be extracted, and {`String`} is a string that should be included in the target data or in the sentence. `<format>` is used in analyzing strings in the rule interpreter, and the system is able to extract the right information from a wide range of documents by defining multiple `<format>` elements.

Figure 5 is an example of information extraction rules generated from a real Web site for job advertisement. In this rule, the first pattern is related to the *company name*, and the second pattern is for the *application submission period*. As you can see in this rule description, the `<trigger>` element acts not only as the initiator of sentence analysis, but also as a delimiter that indicates the existence of a particular slot data by redescribing it in the `<format>` element.

4 IE Rule Learner and Rule Interpreter

The rule learner produces patterns of information extraction rules from the user inputs obtained through the interface agent. The information provided by the interface agent includes slot names, target data, and trigger information. By

```
procedure Learner(examples)
  let rule_set = { };
   for each slot:
     for each example in examples:
        patterns = pattern_learner(examples);
        add pattern to rule_set;
  return rule_set;


procedure pattern_learner(examples)
   let patterns = { };
    repeat for each example in examples
       receive input_data from user;
       if input_data is not null
       then find covers
            build pattern
            add pattern to patterns
       remove those examples in examples covered by pattern
     until examples is null
   return patterns
```

Fig. 6. Pattern learning algorithm by using the covering algorithm.

analyzing these data, the rule learner first determines whether the type of target data is `TOKEN` or `PHRASE`, and sets the value of the `<type>` element accordingly. Then, the rule learner finds the trigger and input data, identifies the relationship between the two, and sets the value of the `<format>` element. For example, in a document sentence *TEL: 031-400-5666*, if the target data is *031-400-5666* and the trigger is *TEL*, the value of `<format>` should be `[TRIG]: TARG`.

Basically, the rule learner uses a covering algorithm. In this algorithm, a set of training documents are collected, and every time a new pattern is produced, the learner calculates the covering value of the pattern. Here, the covering value of a pattern is the number of training documents from which the system can extract the target information successfully by using the pattern. According to this covering value, the learner decides whether the new pattern should be accepted or rejected. In our system, the threshold of covering value for accepting a new pattern is set to 2. Figure 6 describes the pseudo-code of pattern learning by using the covering algorithm.

The rule interpreter executes the rules obtained by the rule learner to extract the target information from real documents. The rule interpreter works as follows. First, the test documents are reconstructed as a set of sentences. Next, each sentence is analyzed to determine whether the trigger words occur in the sentence. Only the sentences that contain the triggers are selected, and by applying the value of the `<format>` element, it is determined if the structural

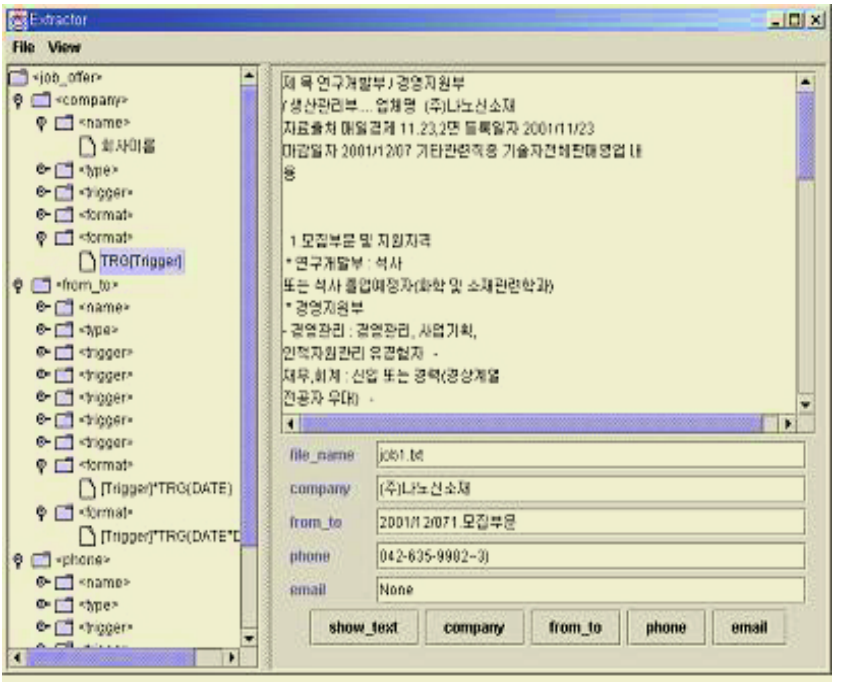

Fig. 7. A running screen of the rule interpreter.

characteristics of the target data is matched with the rule. If the matching is successful, `TARG` part which indicates the correct target data is extracted as a result.

The rule interpreter is implemented by using the XML DOM parser and the Java language. Figure 7 shows a running screen of the rule interpreter. In this figure, the left pane of the window shows the hierarchical structure of the generated XML rule, and the right pane shows the preprocessed document at the top and the result of information extraction at the bottom.

Rules are verified and refined by using the WEKA module [7] that uses the decision tree method. The rules are reconstructed by using the decision tree, and by this way, new rules can be generated. Adding new rules generated from the decision trees to the rule repository enhances the precision and recall measures. Also, the original rules can be verified by re-generating them by decision trees. Furthermore, learned rules can be modified by WEKA if they are not matched with the newly generated rules.

Figure 8 shows the input data including rules and sample data used by the WEKA module to produce decision trees. The attributes are those used in extracting slot values from learned patterns, and the data is produced by the rule interpreter.

```
@relation job.phone

@attribute type {TOKEN, PHRASE}
@attribute trigger {Tel, 문의, 전화}
@attribute format {[TRIG] TARG{PN}, [TRIG]*TARG{PN},
                   [TRIG]*:TARG{PN}}
@attribute rightAccess {yes, no}

@data

phrase,전화,[TRIG]*:TARG{PN},no
token,전화,[TRIG]*TARG{PN},yes
phrase,전화,[TRIG]TARG{PN},yes
.....
```

Fig. 8. Rules and data used by the WEKA module.

5 Experimental Results

Performance of our system is evaluated for a collection of documents in three domains: online bookstores(`Book`), job advertisements(`Job`), and conference call for papers(`CFP`). For the `Book` domain, we choose 32 bookstores in the Internet, and collect five search-result pages from each store. Collected documents are divided evenly into training and test sets. For this domain, we extract the author, the publisher, and the price information. A similar document-collecting process is done for the `Job` domain, and we extract four kinds of target information including the company's name, phone number, and email address, and the job application deadline. For the `CFP` domain, we select the first 100 documents from the search results of Google for the conference call for papers, and divide them into 50 training documents and 50 test documents. The target information in this domain includes the submission deadline, the date for acceptance notification, and the date for camera-ready version submission.

Figure 9 shows the precision and recall measures for the three domains. For the `Job` domain described in Fig. 9(a), the precision and recall values are both over 80 percent in average. In particular, the measures are over 90 percent for the email address attribute, since there are only a few number of trigger words and also the format of the target data is limited, which makes the recognition easier. Since a single trigger exists for each pattern, the precision and recall measures get higher as there are more patterns. In the case of the phone number which is mostly represented in digits, the chances are that it can be mistakenly recognized as the submission deadline. In this case, the triggers play crucial role in distinguishing the two kinds of data.

Fig. 9(b) depicts the result for the `CFP` domain. Here, the measures for the paper submission deadline are good, so are the date for acceptance notification.

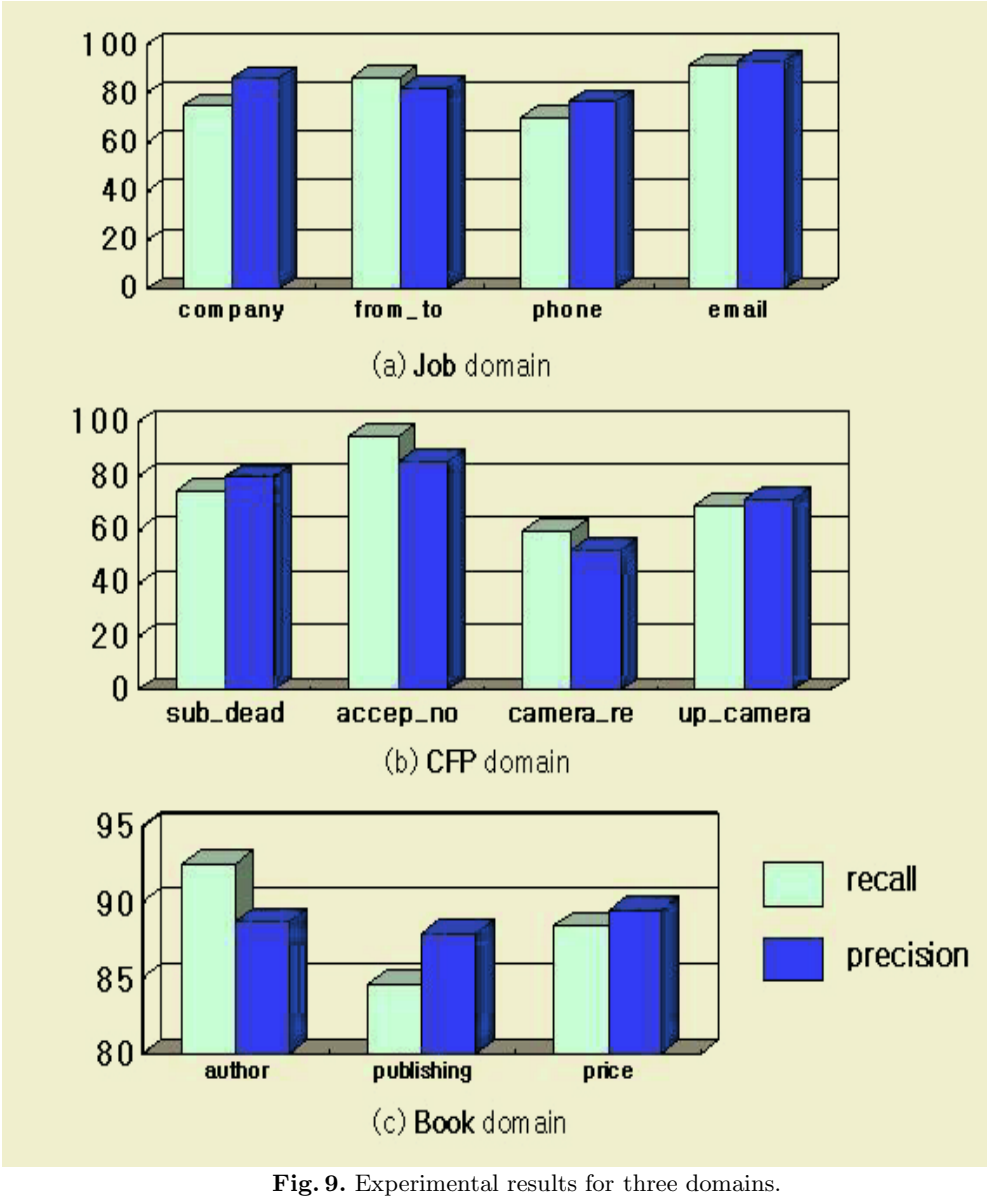

Fig. 9. Experimental results for three domains.

But, the recognition of the date for camera-ready submission is not done well, since the word *final version* which rarely occurred in the training documents appeared frequently in the test documents. This is the result of overfitting of learning algorithm to the training data. To remedy this, we have used the WEKA module that performs the learning processing again to enhance the extraction performance of the system. In this case, by adding the word *final* as a new trigger in WEKA, the measures for the camera-ready attribute get better while the

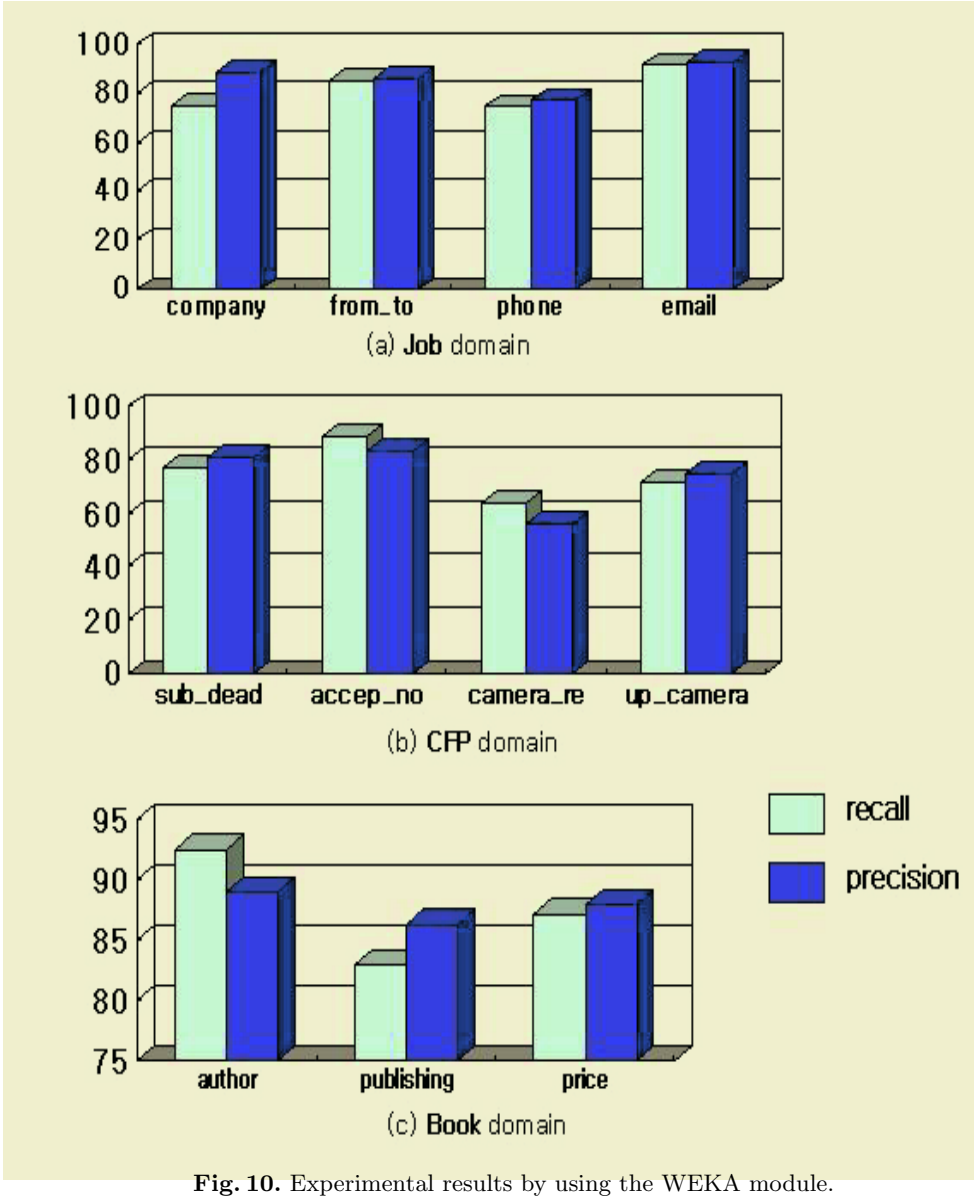

Fig. 10. Experimental results by using the WEKA module.

results for other attributes remain unchanged. This is reflected in the `up-camera` part of the figure, whereas `camera-re` indicates the original result.

To evaluate the effect of rule refinement by using the WEKA module more specifically, we test the system by adding new patterns obtained from the decision tree structure of WEKA to the original learned rules. The result is shown in Fig. 10. We found that in average only one or two new patterns are added to the original rule, which are effective enough to contribute to the improvement of 6% in recall and 1% in precision.

6 Conclusion

We have presented a new method of building information extraction rules for Web documents through the user-interface agent. This method combines the manual and automatic approaches of rule generation, and adopts the scheme of supervised learning in which a user interface agent is designed to get information from the user regarding what to extract from a document, and XML-based wrappers are generated according to these inputs.

In a prototype system that we have implemented, the user inputs are obtained and converted into training examples through the interface agent. The rule learner analyzes the training data and produces information extraction rules, and the rule interpreter executes the learned rules for real Web documents in order to extract the target information. Our prototype system shows good performance for a collection of documents in three different domains.

One of the limitations of our approach is that the triggers have to be rigourously handled, which means that the word comprising the trigger must be exactly matched to activate it. To resolve this, we are working on managing a semantic ontology for each domain to facilitate the semantic matching of trigger words. Also, we plan to employ a learning method that uses both positive examples and negative examples to avoid overfitting.

References

1. P. Atzeni, G. Mecca, P. Merialdo: Semi-structured and structured data in the Web: Going back and forth. Proc. ACM SIGMOD Workshop on Management of Semistructured Data (1997) 1–9
2. R. Doorenbos, O. Etzioni, D. Weld: A scalable comparison-shopping agent for the world wide web. Proc. Int. Conf. on Autonomous Agents (1997) 39–48
3. J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, V. Vassalos: Template-based wrappers in the TSIMMIS system. Proc. ACM SIGMOD Int. Conf. on Management of Data (1997) 532–535
4. N. Kushmerick: Wrapper induction: Efficiency and expressiveness. Artif. Intell. **118** (2000) 15–68
5. I. Muslea, S. Minton, C. Knoblock: A hierarchical approach to wrapper induction. Proc. Int. Conf. on Autonomous Agents (1999) 190–197
6. S. Soderland: Learning information extraction rules for semi-structured and free text. Machine Learning, **34** (1999) 233–272
7. I. Witten, E. Frank: *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann (1999)
8. J. Yang, E. Lee, J. Choi: A shopping agent that automatically constructs wrappers for semi-structured online vendors. Lecture Notes in Computer Science. **1983** (2000) 368–373
9. J. Yang, J. Choi: Knowledge-based wrapper induction for intelligent Web information extraction. In: N. Zhong, J. Liu, Y. Yao (eds.), *Web Intelligence*, Springer (2003) 153–172
10. J. Yang, J. Choi: Agents for intelligent information extraction by using domain knowledge and token-based morphological patterns. Lecture Notes in Artif. Intell. **2891** (2003) 74–85