ELSEVIER

Expert Systems with Applications 28 (2005) 79–91

Expert Systems with Applications

www.elsevier.com/locate/eswa

Web enabled expert systems using hyperlink-based inference

Wooju Kim[a,*], Yong U. Song[b,1], June S. Hong[c,2]

[a]Department of Computer and Industrial Systems Engineering, Yonsei University, 134 Shinchon, Seoul 120-749, South Korea
[b]Department of Management Information Systems, Yonsei University Wonju Campus, 234 Maji, Wonju, Gangwon 220-710, South Korea
[c]Division of Business Administration, Kyonggi University, San 94-6 Yiui-Dong, Paldal-Gu, Suwon, Kyonggi 442-760, South Korea

Abstract

With the proliferation of the WWW, providing more intelligent Websites has become a major concern in the e-business industry. Recently, this trend has been even more accelerated by the success of Customer Relationship Management (CRM) in terms of product recommendation, and self after service, etc. As a result, many e-companies are eager to embed Web-enabled, rule-based systems, i.e. that is, expert systems, into their Websites, and several well-known commercial tools to facilitate this are already available in the market. So far, most of those tools are based on CGI, but CGI-based systems inherently suffer from problems related to overburdening, when there are too many service demands at the same time. To overcome the limitations of the existing CGI-based expert systems, we propose a new form of Web-enabled expert system that uses a hyperlink-based inference mechanism. In terms of burden to the Web server, our approach has proven to outperform the CGI-based approach theoretically as well as empirically. For practical purposes, our approach is implemented in a software system, WeBIS, and uses a graphic rule editing methodology, Expert's Diagram, that is incorporated into the system to facilitate rule generation and maintenance. WeBIS is now successfully operating for financial consultation on the Website of a leading financial consulting company in Korea.

Keywords: Expert Systems; HTML; Inference; JavaScript; Rule; WWW

1. Introduction

With the advent of the customer-centric paradigm in business strategy, e-companies go beyond merely making the sale to acquire relative competitiveness in terms of customer satisfaction, thereby maximizing the lifetime value of a business relationship (McCarthy, 1999). Giving more intelligence to e-commerce sites is popularly recognized as one of the effective strategies that increase customer satisfaction because they react intelligently and can give a personalized response to each customer. Current Web-enabled, rule-based system tools or expert system tools such as Blaze Advisor (Fair Isaac Corporation) and ILOG JRules are playing a major role in more intelligent Websites. This Web-enabled, rule-based inference technology is applied to various areas of application such as product recommendations, distance learning and training, and help desks for technical support, etc.

Most of the Web-enabled, rule-based systems have been developed using CGI technology, but these CGI-based systems usually cause a relatively high burden to Web servers in terms of both required memory and response time. In the worst case, the system may crash causing a very critical problem to most commercial Websites. One major reason for such crash is that each individual request to a CGI-based system requires higher resources than requests for HTML documents. Therefore, if we can build a rule-based inference system based only on HTML documents, we can be sure that the same inference task can be performed much more efficiently than the CGI-based approach within a given computer resource.

If the HTML-based approach is better, the first problem that presents itself is the building of a system that can perform the same rule-based inference tasks using only one set of HTML documents. Song and Lee (2000) have already shown that a rule can be represented by a set of HTML documents hyperlinked to each other, and the rule can be inferred by following the hyperlinks. To solve this problem,

* Corresponding author. Tel.: +82 2 2123 5716; fax: +82 2 364 7807.
E-mail address: wkim@yonsei.ac.kr (W. Kim).
[1] Tel.: +82 33 760 2340; fax: +82 33 763 4324.
[2] Tel.: +82 31 249 9459; fax: +82 31 249 9459.

0957-4174/$ - see front matter © 2004 Published by Elsevier Ltd.
doi:10.1016/j.eswa.2004.08.011

we applied a generalized transformation algorithm deriving from a set of rules a set of hyperlinked HTML documents equivalent to the set of rules in terms of inference.

The second potential problem arising from adopting an HTML document-based rule inference is that the maintenance of rules will be somewhat different from the CGI-based approach. While the CGI-based approach can use a classical rule maintenance approach applied directly to rules, our approach might eventually need to maintain directly a set of HTML documents in order to properly manage the rule base. Since we provide an automatic transformation mechanism, the FES algorithm, deriving a set of HTML documents from the rules, the classical rule maintenance approach can still be applied to our HTML document-based rule inference without any additional efforts. To address this issue, we developed a framework that can systematically maintain a set of HTML documents. This framework adopts a graphical rule representation mechanism, Expert's Diagram (Lee, Lee, & Choi, 1990), by which a rule base manager, that is, a knowledge engineer, can view and edit a set of HTML documents through an easy-to-understand and graphical rule representation model.

Finally, we have designed and implemented our proposed rule transformation algorithm and HTML-based rule inference system maintenance framework as an automated system called WeBIS. For the empirical validation of our approach, we also present a performance evaluation of our approach through a comparative study with CGI-based approaches.

The remainder of the paper is organized as follows. Section 2 presents reviews of the current rule-based inference systems and addresses their practical limitations. In Section 3, we first present our basic idea of using the hyperlinked HTML document for rule inference and then we propose the application of a generalized rule base transformation algorithm to a set of hyperlinked HTML documents. Section 4 presents an HTML document-based rule inference system maintenance framework. System architecture and implementation issues of WeBIS are also presented. We present a performance evaluation of our proposed inference approach in comparison with the currently popular rule-inference approach, that is, the CGI-based approach, in Section 5. Finally, our conclusions are addressed in Section 6.

2. Related work

We can classify existing methodologies to build Web-enabled, rule-based systems into five categories in accordance with applied technologies. Table 1 shows those five categories and their summarized technical features as developed in Web-enabled, rule-based systems.

As shown in Table 1, we can first categorize existing methodologies into two broad categories, the server-side and the client-side depending on the location of the inference engine of a Web-enabled, rule-based system. The server-side category can be further divided into three more detailed categories, the CGI program, the server-side script, and the Web server embedded module depending on the types of inference engine implemented. On the other hand, the client-side category is further classified into two sub-categories, the external viewer and the Java applet. The technical features column of Table 1 describes the characteristic features of each category that differentiate them from each other.

Table 1
Categories of Web-enabled, rule-based systems

Location of inference engine	Type of inference engine	Technical features
Server-side	CGI program	Uses CGI (Common Gateway Interface) standard. Web server invokes a CGI program while passing required parameters according to the CGI standard.
	Server-side script	Inference engines are developed under environments such as JSP, ASP, and PHP. Transaction processing and multi-threading functions are provided by default.
	Web server embedded module	Inference engines are embedded into the Web server as a sub-module using API such as NSAPI.
Client-side	External viewer	Is developed as an independent program. Is invoked by the Web browser according to predefined MIME type.
	Java applet	Bytecodes for inference engine is located on the server-side but is transmitted to a Web browser. JVM on the client-side interprets the bytecodes and executes them.

Concerning practicality and efficiency in developing and operating a Web-enabled, rule-based system, let us examine the pros and cons of each category mentioned above. The most popular approach to the Web-enabled, rule-based system is the CGI program. Several well-known rule-based systems such as EXSYS, Blaze Advisor, and ILOG are based on the CGI program. One of the major reasons for CGI's popularity is its generally easy maintenance and its ability to extend rule-based systems compared with the Web-server embedded module approach or the client-side approaches. The CGI program, however, has revealed a serious weakness in its ability to respond to requests from clients because the number of clients is growing faster than most computing resources. Currently, server-side script approaches are recognized as an alternative that can partially complement this weakness of the CGI program since they allow multi-threading instead of forking and provide an automated transaction processing function to reduce the burdens on Web servers more efficiently than the less savy CGI program. Even though the server-side script approach is better than the simple CGI program approach,

the Web server burden incurred by the server-side script approach is still much bigger than the HTML document-based approach, and causes a problem of overburdening with limited computer resources.

The third method belonging to the server-side approach is the Web server embedded module approach. In this case, since the Web server contains a rule-based system as a sub-module, it is easily expected that this approach will cause more difficulties and cumbersome tasks in the maintenance and extension of the rule-based system in comparison with the CGI program or server-side script approaches. We speculate that this is a major cause of the low number of commercial Web-enabled, rule-based systems that can be developed by using the Web server embedded module approach.

As a sub-category of the client-side approach, the external viewer approach uses an independent rule-based inference system which is contained in the client computer. Therefore, the computational burden to Web server may be reduced on some limited level, but maintenance of the system is more difficult than the server-side approaches since each program must be installed separately from the Web server and this requires a reinstallation of the system every time it is updated. These problems prevent it from being applied to the development of a Web-enabled, rule-based system for commercial use. Another sub-category in the client-side approaches is Java applet, which solves the major problems of the external viewer approach, such as version management and consistent maintenance, with its platform independent framework. Although ILOG and e2gLight (eXpertise2Go) also provide this Java applet based approach because of these advantages, it is nevertheless true that Java applet is not widely accepted yet as a practical, commercial approach to develop the Web-enabled, rule-based system in a real world industry. The Java applet approach needs bytecodes, that is, executable programs, to be downloaded only once from the Web server to the client at the time of consultation to a rule-based system, and does not create any further burdens to the Web server. Instead, it creates additional data traffic such as rule-base or parts of databases (in some cases, it may be whole databases) between server and client. But the size of the rule base and required parts of databases for inference are usually much bigger than the size of inference engine in most commercial systems; this seems to be the main reason for the lack of popularity of the Java applet approach for commercial purposes.

Reducing latency time is most critical to the success of a Website (Zari, Saiedian, & Naeem, 2001), and is also significant to Web-enabled, rule-based systems. In this review of related approaches to Web-enabled, rule-based systems, we have identified reducing latency time, that is, reducing burdens caused by rule-based systems to the Web server, as a most important, but not fully resolved, issue. This means it is still worthwhile to keep making an effort to improve the efficiency of Web-enabled, rule-based systems in terms of latency time.

Based on the premise that we can make a rule-based system only with hyperlinked HTML documents, and no functional differences between hyperlink-based inference system and conventional Web-enabled, rule-based systems exist, the hyperlink-based inference system is definitely much more efficient than the conventional methodologies that use the CGI program or server-side scripts in terms of latency time. If, however, we want to make use of this advantage of the HTML-based system, we must first establish the premise of our proposal by showing how a rule-based system can be implemented with a set of hyperlinked HTML documents. In addition to this, we have to secure a general-purpose mechanism to transform a set of rules into a set of functionally equivalent hyperlinked HTML documents that work in real world, practical application.

If we can address both of the above prerequisites, we can take full advantage of the hyperlink-based inference systems in Web-enabled, rule-based inference tasks. Section 3 discusses our proposed approach to satisfy both of these prerequisites.

3. Hyperlink-based inference systems

3.1. Hyperlink-based Inference

If we want to use a hyperlink-based inference system instead of a conventional rule-based system, first we have to prove the hyperlink-based inference system is functionally equivalent to the conventional rule-based system with any given rule base. In this paper, the definition of the rule base that we will deal with is restricted only to the backward-chaining style rule. That is, this set of rules is devised to deduct one or a group of goals. Now, let us assume, we have the following two rules to prove proposition *c*:

Rule #1: **If** *a* **And** *b* **Then** *c*
Rule #2: **If** *d* **And** *e* **Then** *a*

Fig. 1 depicts these two rules by using a popular graphic rule representation scheme, AND/OR graph (Giarratano & Riley, 1994; Nilson, 1980; Rich, 1983).

If we apply these two rules to a typical rule-based system with the purpose of proving the proposition *c*, it will ask a user

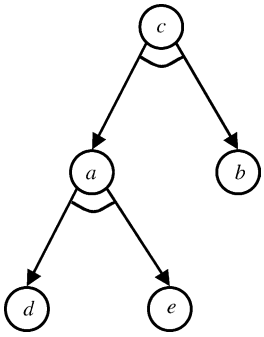

Fig. 1. An AND/OR graph.

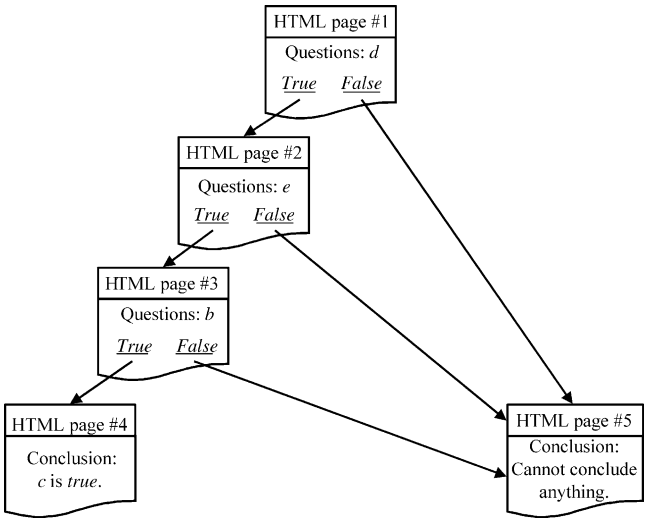

Fig. 2. An example of a functionally equivalent set of hyperlinked HTML documents.

about *d*, *e*, and *b* consecutively under the assumption that the applied backtracking method follows the left-to-right principle. If the results from these three questions are all *true*, then *c* is proved to be *true*. Otherwise, if any one of the answers is *false*, the rule-based system cannot accurately conclude anything. Related to the two above rules, we can posit the same series of questions and conclusions as typical rule-based systems do, by using a set of hyperlinked HTML documents. The set of hyperlinked HTML documents for our example is illustrated in Fig. 2, and the readers can easily identify that these hyperlinked HTML documents can do exactly the same inference tasks as the rule-based systems. Let us define such a functionally equivalent set of hyperlinked HTML documents for a given set of rules in terms of inference as Functionally Equivalent hyperlinked HTML document Set (FES) denoted by *FES*(*x*) for a given set of rules *x*.

So far, we have only addressed the rules having propositional symbols in their rule sentences. This type of rule is usually called a 'Fact Type' rule. However, there is another type of rule which is frequently and generally used in rule-based systems that is different from the fact type rule in that it utilizes *variables* in representing sentences. If we introduce variables into rule representation, we have to consider a set of available values, that is, domains of the variables. The domain for the variable can usually be divided into two types, the symbolic type and the numeric type. For the cases of the symbolic type domain, the number of available values is usually finite, and so we are supposed to ask users to select one or several of the available values in the domain. Let us assume, we have the following three rules related to traffic control:

Rule #3: **If** *the color of the traffic signal* = green **Then** go
Rule #4: **If** *the color of the traffic signal* = yellow **Then** slow down and prepare to stop
Rule #5: **If** *the color of the traffic signal* = red **Then** stop

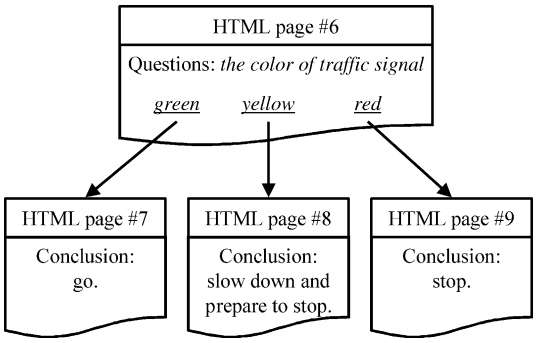

Fig. 3. An example FES for a traffic control rule set.

In this case, the variable is 'the color of the traffic signal' and its domain is {green, yellow, red}. We can also easily construct *FES*({Rule #3, Rule #4, Rule #5}) for these three rules as depicted in Fig. 3.

In the case of the variable that has a numeric type domain, however, the construction of a corresponding *FES*( ) is not so simple because we must first obtain the exact numerical value via hypertext from a user, and this is practically impossible to obtain. Of course, in the case of only one set of a finite number of ranges in value exclusive to each other, we may construct *FES*( ) in a way similar to the symbolic type domain by regarding each range as a separate hyperlink. But, in a general sense, since we may need to compute an expression based on the value of the variable and to compare it with a constant or other variables, we can say that the hyperlink on its own is not complex enough to represent the rules containing numeric type of variables.

One simple way to cope with this limitation of the hyperlink is to use a client-side script language such as JavaScript and VBScript to obtain the values of the variables and to compute and compare them if needed. Suppose the following two rules related to a tax consultation:

Rule #6: **If** *total income* $\geq 0.2 * threshold$ **Then** Have to pay tax.
Rule #7: **If** *total income* $< 0.2 * threshold$ **Then** Do not need to pay tax.

From these two rules, we can identify two numeric type variables of *total income* and *threshold*. We also find that numerical computations and comparisons must be performed for inference. Using JavaScript, we can construct *FES*({Rule #6, Rule #7}) with three hyperlinked HTML documents: one JavaScript-embedded document for the premise part and two simple HTML documents for the conclusions. Fig. 4 illustrates the required source of the first JavaScript embedded document. This source is then divided into two parts, the Form and the Script as shown. The Form obtains the values of the numerical variables in rules from users. The Script performs computations and comparisons of the obtained values

```
<HTML>
<HEAD>
<TITLE> Numeric Type </TITLE>
<SCRIPT LANGUAGE="JavaScript">
    <   !--
        function verifyValue(form)
        {
            if (form.total_income.value >= 0.2 * form.threshold.value)
                location="tax.html"
            else
                location="notax.html"
        }
    //-->
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="exprform">
Input the values of the following variables… <BR>
total income: <INPUT TYPE="text" NAME="total_income"> <BR>
threshold: <INPUT TYPE="text" NAME="threshold"> <BR>
<INPUT TYPE="button" VALUE="OK" onClick="verifyValue(this.form)">
</FORM>
</BODY>
</HTML>
```

Fig. 4. Source of a JavaScript embedded HTML page for numeric type variables.

required in the premise of the rule and then hyperlinks to the corresponding page for a conclusion depending on the results from the comparisons. The Script is written in boldface to distinguish it from the Form in Fig. 4. Fig. 5 illustrates example screen shots generated by *FES*({Rule #6, Rule #7}).

Of course, which of the two screens will be activated appears on the lower side of Fig. 5 and depends on the computation of the user's input performed in the upper part of the screen. So far, we have shown with several examples that typical but various types of rules can be transformed into functionally equivalent hyperlinked documents, that is, *FES*( )s. But, as of yet in spite of such examples, we are not sure if we can find *FES*( ) functionally equivalent to any arbitrary set of rules. To make sure that *FES*( ) exists for any arbitrary set of rules and can also be found, we propose an *FES* generation algorithm for an arbitrary set of rules. This algorithm is addressed in Section 3.2.

3.2. FES generation algorithm

Basically, our FES generation algorithm takes an arbitrary set of rules and transforms it into a functionally equivalent set of hyperlinked documents. To achieve this, the FES generation algorithm works in two major phases. In the first phase, we convert each rule of an arbitrary rule set into its corresponding canonical form. Based on

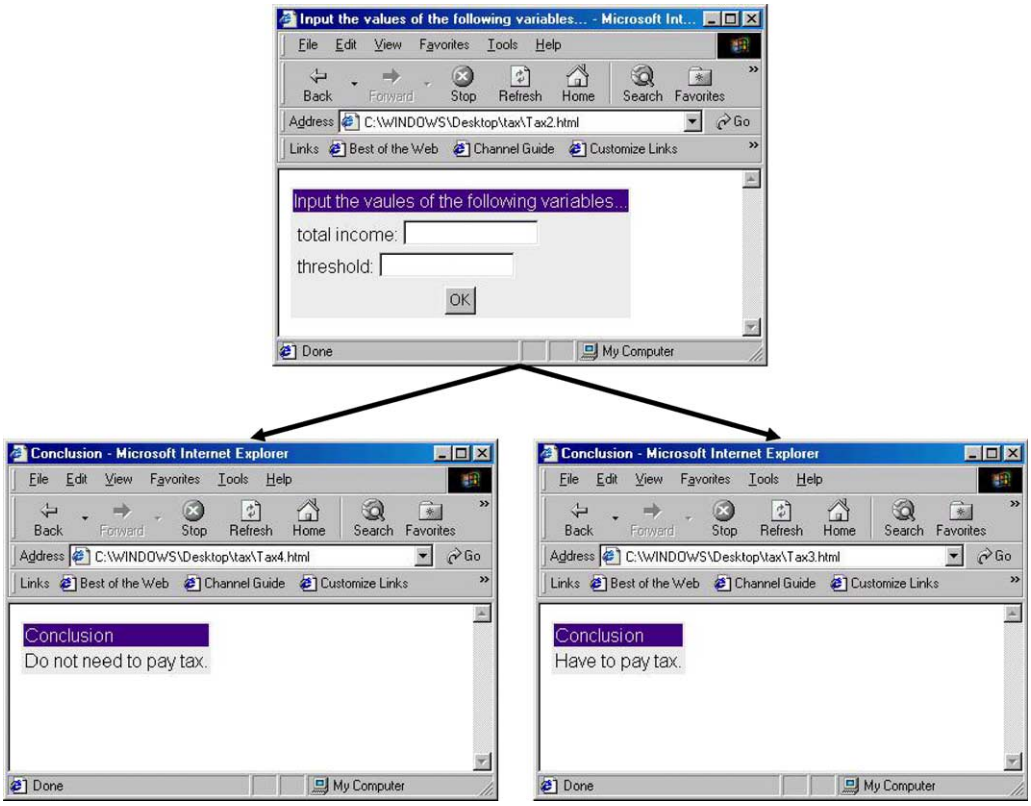

Fig. 5. Example screen shots generated from FES ({Rule #6, Rule #7}).

the converted rules in the canonical form, the second phase generates, in terms of inference, a functionally equivalent set of hyperlinked documents (*FES*).

In the first phase, we define the required canonical form of rules in our context and name this required canonical form the simplified normal form (SNF). If a rule is in SNF, it must satisfy three conditions. First, the rule should not have any chaining relationship with any other rules in the rule set. In this case, the existence of a chaining relationship of a rule to other rules implies the rule has at least one element of the premise part which also appears in the consequent part of the other rules and vice versa. The second condition for SNF is that a rule can only have conjunctions in the premise part. In the third condition, each rule can have only one consequent proposition. To convert arbitrary rules into SNF, we provide a simple three step converting procedure in the figure below with some examples of each step.

3.2.1. Conversion procedure into SNF

Step (1) Eliminate all chaining variables by merging related rules.
Example: $\{Q \vee R \Rightarrow C, P \wedge C \Rightarrow S\} \rightarrow \{P \wedge (Q \vee R) \Rightarrow S\}$

Step (2) Convert the premise part of each rule into a disjunctive normal form by the associative law.
Example: $\{P \wedge (Q \vee R) \Rightarrow S\} \rightarrow \{(P \wedge Q) \vee (P \wedge R) \Rightarrow S\}$

Step (3) Eliminate disjunctions of the premise part of each rule by splitting it into multiple rules with same consequent.
Example: $\{(P \wedge Q) \vee (P \wedge R) \Rightarrow S\} \rightarrow \{P \wedge Q \Rightarrow S, P \wedge R \Rightarrow S\}$

After converting the rules into SNF, we can proceed to the second phase where *FES*( ) will be generated based on the rules in the SNF. For this second phase, we developed an algorithm which transforms a set of SNF rules into *FES*( ) and we call it FES generation algorithm. Before going into the details of FES generation algorithm, let us look into the required definitions and notations first. Let R be a set of rules and r be a rule, which belongs to R. And each r has a premise part, *pr* and a consequent part, *cs*. *pr* consists of a set of atomic propositions while *cs* has a single consequent. p and c denote atomic propositions contained in *pr* and *cs*, respectively.

Related to describing hyperlinked documents formally, let us adopt conventional notations from graph theory. T means a tree of hyperlinked documents and consists of two sets, V and E. V is a finite set of vertices, that is, HTML documents in our context, and E is a set of edges between vertices, that is, hyperlinks in our context. v and e denote, respectively, a vertex in V and an edge in E. Especially in our case, v corresponds to an HTML document and we can further represent the corresponding HTML document content of v_k with $content(v_k)$. An edge e can be naturally represented as a pair of vertices, (v_k, v_l) where v_k and v_l are kth and lth arbitrary vertices in V, and this means that the edge connects the two vertices. Furthermore, each edge has a direction and the left-hand vertex in the pair is a parent vertex, while the right-hand vertex is a child vertex because the generated graph is a kind of tree. Therefore, in the above case, v_k is a parent and v_l is a child. In terms of the edge e, Let us denote the parent and child vertices by $pv(e)$ and $cv(e)$, respectively. In addition to this, since each vertex will eventually be matched with an atomic proposition in a rule, each edge also represents one of the Boolean constants, a *true* or *false* value of the corresponding atomic proposition. The represented Boolean value of an edge e is denoted by $value(e)$.

Up to this point, we have addressed the notations about rules and hyperlinked documents themselves, but more definition is needed in order to successfully transform rules into *FES*( ). We formally define a term as *fact* when an atomic proposition has an arbitrary Boolean value and it is denoted by f. That is, f is represented by $(p, value(p))$, which means the atomic proposition p has $value(p)$ regardless of whether its value is *true* or *false*. To access p and $value(p)$ in f, we define two functions, $prop(f)$ and $propval(f)$ where they designate p and $value(p)$, respectively.

Let us make one more definition, preceding fact set (PFS). PFS is defined from the point of view of the vertex, and it designates facts accumulated along the path from the root to a specific vertex, v_k. As is commonly understood, a path consists of a set of edges; and each edge implies whether the related parent vertex, that is the proposition of that vertex, is *true* or *false*. Therefore, for a given vertex, we can have a series of related *fact*s along the path to that vertex. The set of those facts to a given vertex v_k is denoted by $PFS(v_k)$. To define *PFS* in a more formal manner, let $path(v_k)$ be a set of vertices laid along the path from the root to the vertex v_k, $\{v_1, v_{i1}, v_{i2}, \ldots, v_{im}, v_k\}$. Then,

$$PFS(v_k) = \{\text{a set of } f = (content(v_l),\ value(e)) \text{ s.t. } v_l \in path(v_k),\ pv(e) = v_l,\ cv(e) \in path(v_k),\ \text{and } l \neq k\}.$$

Based on the concepts and definitions addressed so far, we will now develop the second phase algorithm mentioned earlier in this subsection. The algorithm 'FES generation algorithm' consists of four routines (functions) including the main routine. Fig. 6 shows the main routine, ConvertRuleBaseIntoFES and the remaining three subroutines, ConvertRuleIntoTree, Match, and Exist, which are described in Figs. 7–9, respectively. It is easy to understand this FES generation algorithm with an example which shows how the algorithm proceeds. Let the following two rules in SNF be a rule set needed to be transformed into FES.

```
function ConvertRuleBaseIntoFES(R) returns FES
  inputs: R, a set of rules
  variables: T = (V, E), a hyperlinked pages and their connection information
             IncompleteVertices, a set of nodes which do not have any content
             e, an edge
  V ← ∅, E ← ∅, IncompleteVertices ← ∅
  for each r in R do
    IncompleteVertices ← ConvertRuleIntoTree(IncompleteVertices, r, T)
  end for
  for each v in IncompleteVertices do
    content(v) ← "No conclusion"
  end for
  return T
```

Fig. 6. ConvertRuleBaseIntoFES algorithm.

$$R_1 : A \wedge B \wedge C \Rightarrow X$$

$$R_2 : B \wedge D \Rightarrow Y$$

Fig. 10 illustrates how FES generation proceeds to generate hyperlinked HTML documents for the example rules at a step by step level. The following seven steps describe each step shown in Fig. 10.

(1) In the first step, the main routine ConvertRuleBaseIntoFES is invoked with a parameter $R = \{R_1, R_2\}$. Then the main routine invokes the subroutine ConvertRuleIntoTree for each rule in R ($= R_1$ and R_2) while passing the *IncompleteVertice*, a rule r, and a tree T.

```
function ConvertRuleIntoTree(IncompleteVertices, r, T) returns IncompleteVertices
inputs: IncompleteVertices, a set of incomplete vertices
        r = (pr, c), a rule
        T = (V, E), a binary tree to represent decision tree
variables: tmpVertices, a set of newly created incomplete vertices
           CurrentVertex, a vertex
  tmpVertices ← ∅, CurrentVertex ← NULL
  if IncompleteVertices = ∅ then
    CurrentVertex ← NULL
    for each p in pr do
      if CurrentVertex = NULL then
        make a vertex v s.t. content(v) = p
        V ← V ∪ {v} and CurrentVertex ← v
      else
        content(CurrentVertex) ← p
      end if
      make a vertex, falseVertex s.t. content(falseVertex) = NULL
      make a vertex, trueVertex s.t. content(trueVertex) = NULL
      make an edge, falseEdge = (CurrentVertex, falseVertex) s.t. value(falseEdge) = false
      make an edge, trueEdge = (CurrentVertex, trueVertex) s.t. value(trueEdge) = true
      V ← V ∪ {falseVertex, trueVertex} and E ← E ∪ {falseEdge, trueEdge}
      tmpVertices ← tmpVertices ∪ {falseVertex}
      CurrentVertex ← trueVertex
    end for
    content(CurrentVertex) ← c
  else
    for each v in IncompleteVertices do
      if Match(pr, v) = false then
        tmpVertices ← tmpVertices ∪ {v}
      else
        CurrentVertex ← v
        for each p in pr do
          if Exist(p, CurrentVertex) = false then
            content(CurrentVertex) ← p
            make a vertex, falseVertex s.t. content(falseVertex) = NULL
            make a vertex, trueVertex s.t. content(trueVertex) = NULL
            make an edge, falseEdge = (CurrentVertex, falseVertex) s.t. value(falseEdge) = false
            make an edge, trueEdge = (CurrentVertex, trueVertex) s.t. value(trueEdge) = true
            V ← V ∪ {falseVertex, trueVertex} and E ← E ∪ {falseEdge, trueEdge}
            tmpVertices ← tmpVertices ∪ {falseVertex}
            CurrentVertex ← trueVertex
          end if
        end for
        content(CurrentVertex) ← c
      end if
    end for
  end if
  return tmpVertices
```

Fig. 7. ConvertRuleIntoTree algorithm.

```
function MATCH(pr, v) returns Boolean
inputs: pr, a set of propositions
        v, a vertex
  for each p in pr do
    for each f in PFS(v) do
      if p = prop(f) and propval(f) = false then
        return false
      end if
    end for
  end for
  return true
```

Fig. 8. *MATCH* algorithm.

At this moment, *IncompleteVertice* and T are empty and r is R_1 in our example. Then the subroutine CONVERTRULEINTOTREE transforms a given rule into a tree of HTML documents by creating and appending vertices and edges. This subroutine can be divided into two parts, the initialization part and the tree growing part. In the case of R_1, it performs the initialization part which will process every atomic proposition in the premise part, and then it finishes by updating the content of the leaf node with a consequent proposition. During processing each proposition of the premise part, the tree, T will grow. As a result of the first iteration of this processing, Step 1 in Fig. 10 shows the generated tree based on the first proposition, A of R_1 in our example.

(2) By processing the second proposition, B of R_1, we have the tree as depicted in Step 2 in Fig. 10. The white box node means it is expected to become a member of the IncompleteVertice in the next stage in order to process the remaining rules.

(3) In the third step of Fig. 10, the algorithm processes the final proposition, C of R_1 and updates the *true* side node with the consequent of the rule, X. The conclusion nodes are denoted by circle nodes here. t and f attached on the arcs means that the proposition of the parent node of that arc is either *true* or *false*. In this step, the transformation of rule R_1 into the tree is finished and so, the execution of the subroutine Convertruleintotree for the rule R_1 is also ended by returning to the main routine.

(4) In the fourth step, the algorithm invokes the subroutine CONVERTRULEINTOTREE again for the rule, R_2 together with the IncompleteVertice, which consists of the vertices marked with a white box and the generated tree, T that was the result in (3). In this case, CONVERTRULEINTOTREE applies the tree growing part procedure which is located at the outermost *else* part to the rule R_2. The first vertex of the IncompleteVertice is the *false* side child node of A and the algorithm first applies MATCH with $\{B, D\}$ as pr to this vertex. Since there is no evidence that B or D is *false* along the path, MATCH is evaluated as *true* and this leads to the *else* part where the tree is growing while considering the existence of propositions in the path. In our example, B is first considered and is appended to the tree and its result is shown in Step 4 of Fig. 10.

```
function EXIST(p, v) returns Boolean
inputs: p, a proposition
        v, a vertex
  for each f in PFS(v) do
    if p = prop(f) then
      return true
    end if
  end for
  return false
```

Fig. 9. *EXIST* algorithm.

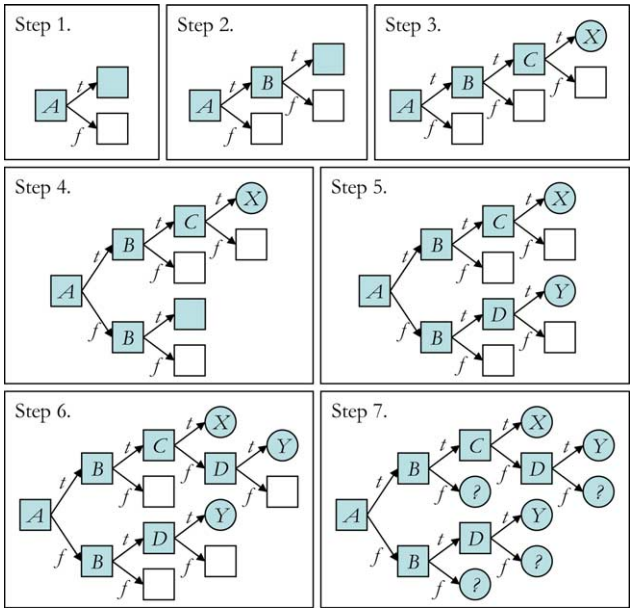

Fig. 10. Progress of FES generation algorithm with the example rules.

(5) Proposition D of rule R_2 is checked against MATCH and EXIST and then is added into the tree as shown in Step 5 of Fig. 10. At the same time, since all propositions in the premise part of R_2 are processed, the consequent Y is updated on the *true* side leaf node.

(6) In the sixth step, the *false* side vertex of B in the upper side of the tree is examined, but it does not pass the MATCH test, so the vertex still remains as the *IncompleteVertice*. Now, the *false* side vertex of C in Step 3 is examined and the rule satisfies the MATCH condition. Therefore, B and D are examined against EXIST and only D satisfies the *false* condition. As a result, D is added to the tree and finally, its consequent part Y is also updated on the *true* side of D.

(7) In the final step, the algorithm updates all remaining vertices in the IncompleteVertice by 'No Conclusion' which is denoted by a question mark, '?', because all the rules were processed in R. The completed result of the hyperlinked HTML documents, that is, the *FES* correspondence to the example rule set is depicted in Step 7 of Fig. 10.

So far, we have discussed our proposed FES generation algorithm. In Section 3.3, we will address the minor extension of the algorithm and show a brief real world example based on a bank loan approval.

3.3. Algorithm extension and its real word example

The algorithm proposed in Section 3.2 can only cover the so called 'Fact Type' rule mentioned earlier. We already explained that there are two additional popular rule types, rules with symbolic variables and numerical variables. But with minor revision on the FES generation algorithm, we can easily take into account those types of rules. We are not going into the details of revision here since they are so trivial. Fig. 11 shows a part of the real world example rules for a bank loan approval process. These example rules include all three types of rules, so this real world example will show how the example rules are transformed into a set of hyperlinked HTML documents by using our extended algorithm.

We applied the extended FES generation algorithm to the example rules in Fig. 11 that creates a set of hyperlinked HTML documents in Fig. 12. As you can see in this figure, the condition which has a symbolic variable is reflected in the root HTML document and the condition having a numerical variable is reflected in the HTML document titled, "Current Amount of Monthly Debt". Especially in the latter document, users are required to enter the exact value of their current amount of monthly debt and the corresponding JavaScript code will decide which condition is satisfied by the user's input and then lead to the corresponding HTML document.

4. WeBIS approach

4.1. Architecture of WeBIS

So far, we have proposed the idea that the hyperlinked HTML document can perform inference tasks, and we have shown how a rule base can be automatically converted into a FES. To facilitate our approach, we provide a system architecture, Web Based Inference System (WeBIS), which can support users to develop a set of hyperlinked HTML documents to perform an inference task systematically. Fig. 13 shows the overall architecture of WeBIS and its three major components. Major information flows are also depicted. As shown in Fig. 13, the knowledge engineer can input rules to the system via both the general text editor and the graphical rule editor provided by WeBIS. Then, WeBIS transforms rules into functionally equivalent set of hyperlinked

IF Please describe which best suits your needs = Purchase a Home
AND Are you a first Home Buyer? = Yes
THEN Accept

IF Please describe which best suits your needs = Purchase a Home
AND Are you a first Home Buyer? = No
THEN Reject

IF Please describe which best suits your needs = Refinance your Home
AND Does your current monthly mortgage payment exceed your monthly income? = Yes
THEN Reject

IF Please describe which best suits your needs = Refinance your Home
AND Does your current monthly mortgage payment exceed your monthly income? = No
AND Current amount of Monthly Debt <= 100
THEN Accept

IF Please describe which best suits your needs = Refinance your Home
AND Does your current monthly mortgage payment exceed your monthly income? = No
AND Current amount of Monthly Debt > 100
THEN Reject

Fig. 11. Example rules for a bank loan approval.

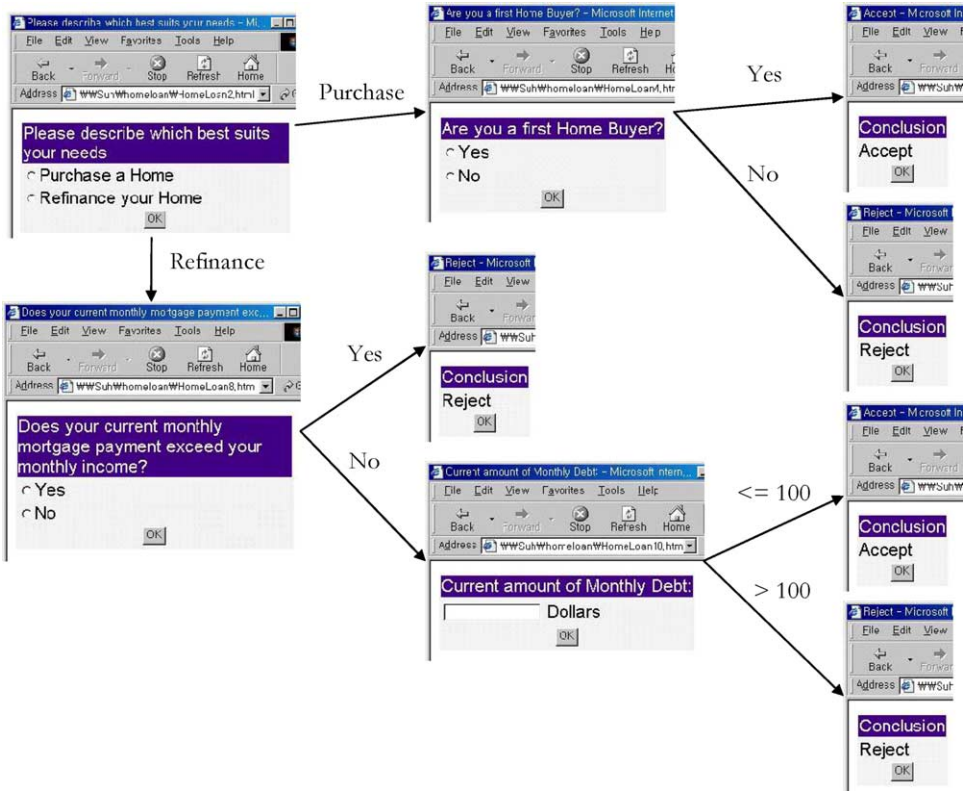

Fig. 12. Generated hyperlinked HTML documents for bank loan example rules.

HTML documents. Those documents are registered to Web servers and finally users can access them and get inference services via the Internet.

Now Let us briefly introduce each of the components, their roles, and related architectural issues.

(1) *Graphical Rule Editor (Expert's Diagram Approach).* The graphical rule editor supports complete rule editing process by specifying Expert's Diagram in a GUI environment. Lee et al. (1990) claimed that expressing rules in a graphical manner is more efficient in most cases

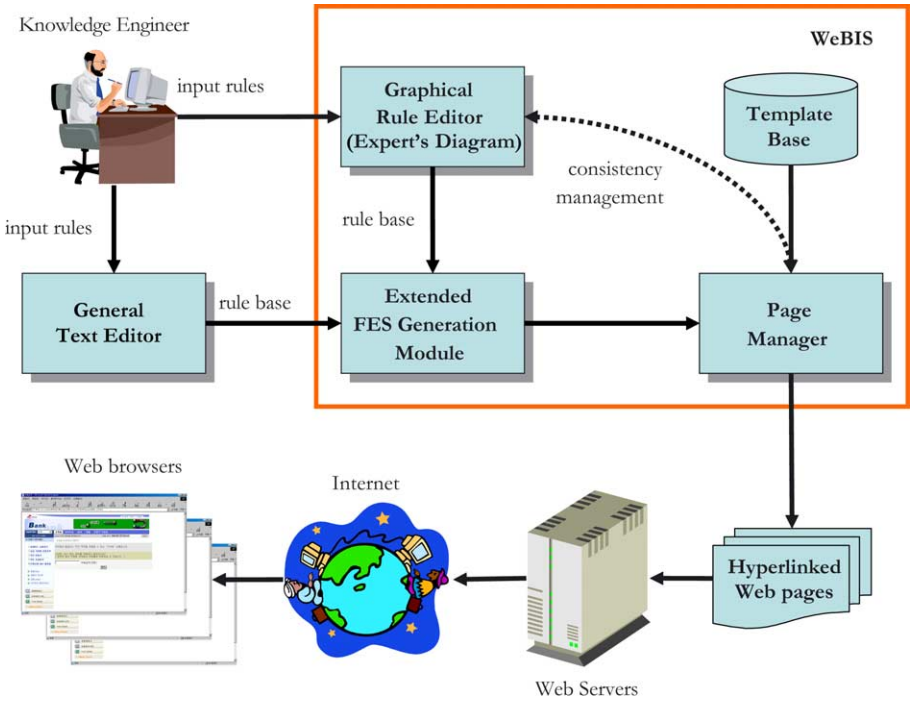

Fig. 13. Architecture of WeBIS.

especially when a knowledge engineer is not skilled in knowledge engineering, and they proposed a graphical rule representation scheme, Expert's Diagram. We adopt this rule representation scheme as default in our system and develop a rule editor which supports user to express rules in Expert's Diagram format. But skilled knowledge engineer can still input rules directly by using any general text editors. Fig. 13 also depicts such direct rule input process. Whether the rules are edited by a graphical rule editor or general text editor, they are fed into an extended FES generation module.

(2) *Extended FES Generation Module*. The extended FES generation module implements the algorithm addressed in Section 3. It is responsible for transforming rules into a functionally equivalent set of hyperlinked HTML documents. After finishing the transformation, it invokes the page manager by passing the generated HTML document set.

(3) *Page Manager*. The page manager has two major roles: (1) to finalize completion of each generated HTML document by applying a corresponding template to it, and (2) to manage consistency between Expert's Diagram generated by a graphical rule editor and a generated set of hyperlinked HTML documents. The additional benefit of using Expert's Diagram is that the user can easily manage consistency by cooperation between the page manager and the graphical rule editor when there are minor changes in rule content.

4.2. Implementation

We have incorporated the framework of our Web-based inference approach into a working prototype written in Visual C++ on the Windows environment. Fig. 14 shows an illustrative screen where the user creates a rule base and hyperlinked HTML documents, transforms the rule base into equivalent hyperlinked HTML documents, FES back and forth, edits FES using a graphical interface, and generates styled HTML documents using templates. As shown in Fig. 14, the user can choose a rule base file on the left frame and then convert it into FES as displayed graphically on the right upper frame. The user can also create directly hyperlinked HTML documents based on Expert's Diagram technology. Finally the user can create the templates to be applied to the HTML documents, then the system automatically generates and displays completed HTML documents after applying the templates. The right lower frame in Fig. 14 shows a template-applied HTML document for the selected node (marked with a red border) on the right upper frame. At the time of finishing all these jobs, you can publish the HTML documents to the Web server as a hyperlink-based inference service.

5. Performance evaluation

To prove the efficiency of our approach, we compare the throughput of a hyperlink-based inference system with that of inference systems which were developed by using conventional commercial rule-based systems. To evaluate the throughput of each system, we count the number of HTTP (Fielding et al., 1999) responses per 3 min from the Web server when 10 client computers send HTTP requests simultaneously via the Web. The hardware specification of the server computer is Pentium 4 CPU of 1.5 GHz with 256 MB of main memory. The OS of the server is the Microsoft Windows 2000 Server.

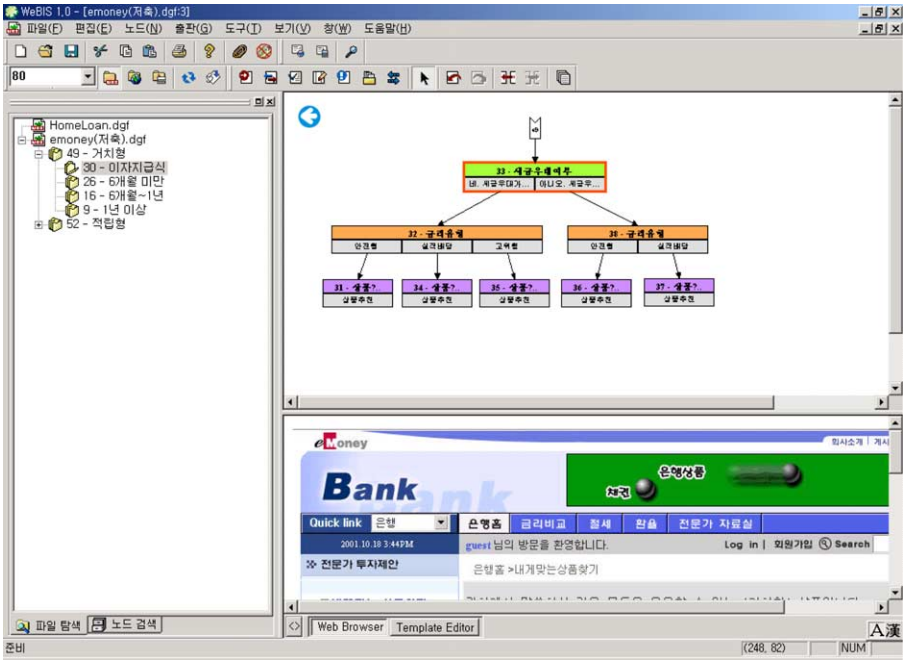

Fig. 14. An illustrative screen shot of WeBIS.

Table 2
Performance evaluation results (one client)

Client	Hyper-link	A[a]	Ratio[b]	B[a]	Ratio[c]
PC1	30,411	307	99.06	5832	5.21
Sum	30,411	307	99.06	5832	5.21
Average	30,411	307		5832	

Unit: #responses per 3 min.
[a] Conventional rule-based systems A and B.
[b] Ratio between Hyperlink and A.
[c] Ratio between Hyperlink and B.

The hardware specification of all of the client computers is Celeron CPU of 633 MHz with 128 MB of main memory. The OS of the client is Microsoft Windows 98. Because the commercial rule-based systems deploy their application systems using Java Server Pages (JSP), we use Tomcat as the Web server. The results appear in Tables 2–4, and Fig. 15.

Table 2 shows the results of performance evaluation when a single PC is used as the client. The second column denoted as Hyperlink shows the throughput of our hyperlink-based inference approach. The third column denoted as A shows the throughput of a commercial rule-based system, and the fourth column shows the ratio between the throughput of our approach (Hyperlink) and the throughput of the rule-based system A. The fifth column denoted as B shows the throughput of another commercial rule-based system, and the sixth column shows the ratio between the throughput of Hyperlink and the throughput of the rule-based system B. When the number of clients is one, the total numbers of HTTP responses per 3 min for Hyperlink, A, and B are 30,411, 307, and 5832, respectively. The ratio between Hyperlink and A is 99.06 and the ratio between Hyperlink and B is 5.21. The result means that even in the case of serving a single client our approach is much faster than the two competitive commercial tools.

Table 3 shows the results produced when five PCs are used as the clients. In this case, the total numbers of HTTP responses are 74,785, 352, and 9583, respectively, and the average numbers of HTTP responses are 14,957, 70, and 1917, respectively. The average performance ratios of Hyperlink to A and B are 212.46 and 7.80, respectively. In terms of ratio, the readers can easily see that the out-performing level is higher than in the case of serving a single client only.

Table 3
Performance evaluation results (five clients)

Client	Hyper-link	A[a]	Ratio[b]	B[a]	Ratio[c]
PC1	14,829	69	214.91	1997	7.43
PC2	15,092	70	215.59	1995	7.56
PC3	14,886	70	212.65	1845	8.07
PC4	14,950	72	207.64	1897	7.88
PC5	15,030	71	211.68	1849	8.13
Sum	74,785	352	212.46	9583	7.80
Average	14,957	70		1917	

Unit: #responses per 3 min.
[a] Conventional rule-based systems A and B.
[b] Ratio between Hyperlink and A.
[c] Ratio between Hyperlink and B.

Table 4
Performance evaluation results (10 clients)

Client	Hyper-link	A[a]	Ratio[b]	B[a]	Ratio[c]
PC1	7471	34	219.74	932	8.02
PC2	7568	34	222.59	931	8.13
PC3	7496	33	227.15	900	8.33
PC4	7563	33	229.17	921	8.21
PC5	7534	32	235.44	895	8.42
PC6	7435	35	212.43	893	8.33
PC7	7486	34	220.16	913	8.20
PC8	7189	33	217.83	930	7.73
PC9	7681	34	225.90	947	8.11
PC10	7755	33	235.00	919	8.44
Sum	75,176	335	224.41	9181	8.19
Average	7518	34		918	

Unit: #responses per 3 min.
[a] Conventional rule-based systems A and B.
[b] Ratio between Hyperlink and A.
[c] Ratio between Hyperlink and B.

The results produced when there are ten PC clients are shown in Table 4. In this case, the total numbers of HTTP responses are 75,176, 335, and 9181, respectively, and the average numbers of HTTP responses are 7,518, 34, and 918, respectively. The average ratios are 224.41 and 8.19, respectively, and the performance gaps between our approach and commercial systems keep growing.

Fig. 15 summarizes the transition of the total numbers of HTTP responses per 3 min for 1-client, 5-client, and 10-client cases. We can expect here that the total number of HTTP responses will be stabilized as the number of clients is increased, and at the same time, the high performance of our approach over two competitive commercial rule-based systems will persist as well. In summary, we can conclude that there are some fixed

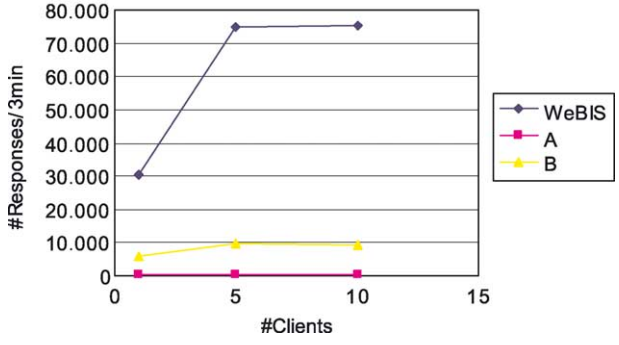

Fig. 15. Performance evaluation results.

and significant gaps between the performance of Hyperlink and those of A and B, and the results of our performance evaluation predicts that the ratio will be approximately 224.41 and 8.19.

6. Conclusions

We have proposed a hyperlink-based inference approach to alleviate the over-burdening problem that troubles most of the conventional Web-enabled, rule-based systems including commercial tools. Since this problem is nearly caused by the CGI-based approach, we have found a way to perform equivalent inference tasks based on hyperlinks between HTML documents as an alternative to the CGI-based approach.

To demonstrate our idea, we first show that a hyperlink-based inference can perform an equivalent inference task to conventional Web-enabled, rule-based systems. Secondly, to facilitate our approach, we proposed an FES generation algorithm which automatically transforms the rule base in the conventional text format into the hyperlinked HTML documents, FES, and proved its completeness. Furthermore, to support the user to create directly hyperlinked HTML documents for inference, we also develop an ease-to-use graphical knowledge acquisition function to build hyperlinked HTML documents systematically based on the Expert's Diagram. Finally, we design and implement a WeBIS which incorporates all of the functions and mechanisms mentioned above.

Using this WeBIS, we have performed an empirical experiment and performance comparison between our approach and two major commercial Web-enabled, rule-based systems. Through this experiment, our approach has proven to outperform both of the commercial rule-based systems in terms of throughput. In addition to this, WeBIS was successfully applied to a real world example, the financial consultation done via the Website of a leading financial consulting company in Korea and it is still running at a very satisfactory level. We expect the best performance of our approach when the number of service requests for rule inference is very large.

References

eXpertise2Go.com, [Online]. Available: http://www.expertise2go.com/

EXSYS Inc., [Online]. Available: http://www.exsys.com/

Fair Isaac Corporation, [Online]. Available: http://www.blazesoft.com/

Fielding, R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T. Hypertext Transfer Protocol—HTTP/1.1, RFC2616, IETF, 1999. [Online]. Available: ftp://ftp.isi.edu/in-notes/rfc2616.txt.

Giarratano, J., & Riley, G. (1994). *Expert systems: principles and programming* (2nd ed.). PWS Publishing Company.

ILOG Inc., [Online]. Available: http://www.ilog.com/

Lee, J. K., Lee, I. K., & Choi, H. R. (1990). Automatic rule generation by the transformation of Expert's Diagram: LIFT. *International Journal of Man–Machine Studies*, *32*, 275–292.

McCarthy, J. C. (1999). The social impact of electronic commerce. *IEEE Communications Magazine*, *37*(9), 53–57.

Nilson, N. J. (1980). *Principles of artificial intelligence*. Berlin: Springer.

Rich, E. (1983). *Artificial intelligence*. New York: McGraw-Hill.

Song, Y. U., & Lee, J. K. (2000). Automatic generation of web-based expert systems. *Journal of Intelligent Information Systems*, *6*(1), 1–16 [in Korean].

Zari, M., Saiedian, H., & Naeem, M. (2001). Understanding and reducing web delay. *Computer*, *34*(12), 30–37.