A Hybrid Architecture for Web-based Expert Systems

Neil Dunstan *neil@cs.une.edu.au*
Faculty of Arts and Sciences,
School of Science and Technology,
Computer Science Division,
University of New England,
Armidale, 2351, Australia

Abstract

A recent technique is to represent the knowledge base of an expert system in XML format. XML parsers are then used to convert XML data into expert system language code. The code is executed or interpreted when providing responses to user queries. Web-based expert system (WBES) architectures may be characterized according to where the application knowledge base resides. Applications of both client and server-sided WBES architectures appear in the literature. A hybrid architecture is proposed where servers provide responses to complex queries using server-based processing of code, and clients handle simple queries using data from the XML knowledge base file. That is, both client and server have access to the knowledge base and share the processing of handling user queries. These concepts are illustrated by a comprehensive description of a small passenger information system, deployed in the hybrid web architecture. A server hosts an XML file describing the passenger network, services, stations and connections. This version of the knowledge base is imported by the client generic web page and used to provide a custom-built user interface consisting of entities derived straight from the knowledge base. At the server, the XML format is converted to Prolog code for handling of complex queries.

Keywords: Web-based Expert Systems, Prolog, XML, Client-Server Architecture.

1. INTRODUCTION

This Despite its longevity, Expert Systems continues to be a popular area of research and as a software engineering paradigm for building intelligent information systems (see [1] and [2] for comprehensive reviews). Some recent application areas include industrial processes [3], agriculture [4] and marine technology [5]. The World Wide Web has increased the reach and accessibility of information systems including those providing expert system services. In [6], [7] and [8] aspects of implementing web-based expert systems (WBES) are discussed. These include knowledge acquisition and representation, inference methods, and system architecture. Although Duan et al. [7] lists use of server and client side inferencing as a challenge, there has been little recent discussion in the literature of WBES of the issues surrounding the choice of client or server-sided approaches and a lack of identifying which approach is used by various applications and development technologies. The more general issue of client and server-side programming is discussed in [8]. The authors cite communications latency and load caused by multiple clients as problems for server-side programming and security and representation of structured information as problems for client-side programming.

In this paper, the client and server approaches to WBESs are explored and a sample of WBES applications are categorized according to their diverse client-server architectures and implementation technologies. A new hybrid architecture that shares query processing between client and server is described. This new approach relies on the knowledge based starting out as an XML file of facts and rules. It is argued that the access to the knowledge base by both client and server permits a balancing of query processing duties as well as a customized user interface.

The method is illustrated by describing a small application including the XML knowledge base, expert system (ES) language code, interaction between client and server and inference engine.

2. WEB-BASED EXPERT SYSTEM ARCHITECTURES

WBES architectures are illustrated in the following figures. Figure 1 shows a typical client-sided architecture where the web page comes with embedded expert system (ES) application code, that is, the knowledge base and any associated code for handling supported queries. This requires the client to have a suitable interpreter for the ES language used. In possession of both application code and interpreter, the client is able to handle user queries without further reference to the server.

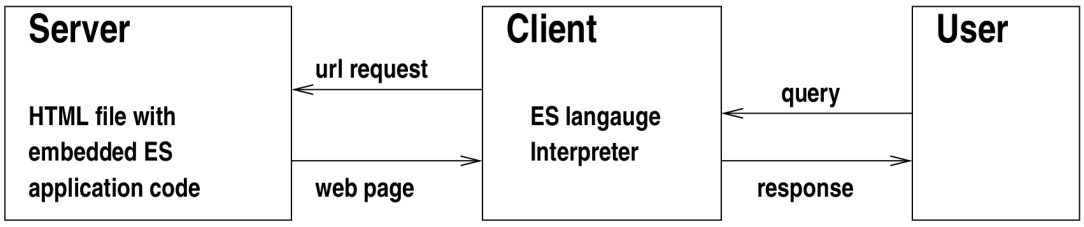

FIGURE 1: Client-sided WBES architecture.

Figure 2 shows a typical server-sided architecture where both ES application code and interpreter reside at the server. The client must refer to the server in order to handle user queries.

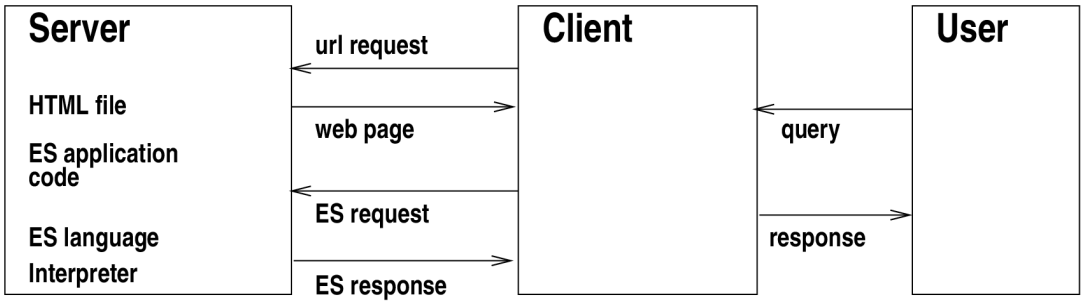

FIGURE 2: Server-sided WBES architecture.

The server-sided approach has the benefit of being able to guarantee service regardless of whether or not the client browser has the necessary software enabling it to process the ES application code. Moreover, the server may tailor the format of its response to queries in order to support service to low-level mobile devices that may have restricted display capabilities. However, the quality of service may be compromised by network load and latency problems. The client-sided processing approach is not effected by network issues (once the web page is loaded) but requires specialized software at the client, which may restrict the range of browsers and devices able to receive the service.

The most commonly used technique to develop WBESs is to use Java Expert System Shell (JESS) [10], which interprets a CLIPS-like language and utilizes the Rete inference algorithm. Applications are server-sided, requiring Java servlets and the Tomcat Servlet Engine. A more general server-sided technique is simply to use Common Gateway Interface (CGI) programs to deliver HTML form data to an expert system (with the ES application knowledge base and inference engine) residing at the server and send the response in HTML back to the client's browser. This more general depiction of the server-side approach is illustrated in Figure 3.

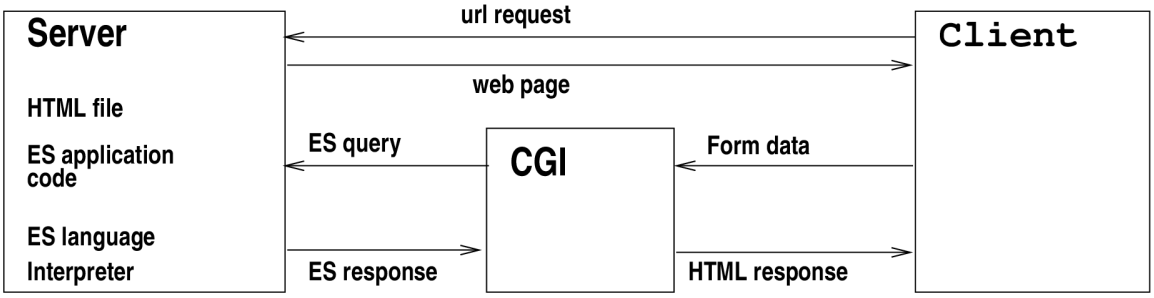

FIGURE 3: Server-sided WBES architecture with CGI.

The CGI program can massage the form data into an appropriate query to the ES as well as convert the response into a convenient form for viewing by the user in the client browser. In a JESS implementation, this is the job of the servlet.

3. WEB-BASED SYSTEMS APPLICATIONS

In this section a sample of WBES applications from the literature are described. They are chosen in order to illustrate diverse implementation technologies and web architectures. They are summarized in Table 1.

Application	Knowledge Base	Inference Engine	Web Architecture
LogicWeb	Prolog	SWI-Prolog	Client-sided
WITS	Javascript	Javascript	Client-sided
CSP	JESS	Rete	Server-sided
ET	XML/Prolog	SWI-Prolog	Server-sided

TABLE 1: WBES application summary.

3.1 Client-sided Applications

LogicWeb [9] was developed using SWI-Prolog [11] and the Mosaic web browser for logical programming on the web. A LogicWeb module consists of a graphical user interface (the browser) and a Prolog query processing engine. It is capable of interpreting web pages containing embedded Prolog code. The issue of the security of this client-sided approach is addressed in [12]. Another client-sided approach is provided by the application Web-based Intelligent Training and support System (WITS) [7]. This system uses Javascript to represent the knowledge base and to conduct inferencing. That is, the web page contains the knowledge base as embedded Javascript code, as well as the code to process it. This is essentially different from the architecture of LogicWeb because the client is only expected to be able to interpret Javascript rather than have some more specialized ES language interpreter.

3.2 Server-sided Applications

Class Schedule Planner [13] is a JESS based system for building class schedules appropriate to individual student's interests and course requirements. Also of interest is the use of XML to communicate user requests to the server-based controller. This is translated into JESS code for processing. CSP also allows real-time updating of course rules via an administration web page. A similar application is Enrolment Tool (ET) [14] which is based on the method described in Dunstan [15] for generating domain-specific WBESs. This method uses a domain-specific XML parser to generate a Prolog knowledge base, web page and associated CGI programs from XML data files describing the knowledge base of an application from the domain. In the case of ET, the domain is course rules, so a WBES to act as an enrolment guide for students can be generated automatically for each course. The CGI programs compose a Prolog query from form

data and execute it via an SWI-Prolog script, reading back the response and sending it as HTML to the client browser. This method corresponds to the architecture shown in Figure 3.

4. A HYBRID WEB-BASED EXPERT SYSTEM ARCHITECTURE

It has been noted that the facts and rules of a knowledge base can be stored in XML format that can be translated into ES language code, that in turn can be processed by an appropriate inference engine. XML can also be read by a web application and stored using the Javascript Document Object Model (DOM). This presents the opportunity to allow a client web page to conveniently access the knowledge base and solves the problem of representing structured information at the client. At the same time, the translated version in ES language code is available for processing by an inference engine at the server. That is, the XML data file representing the knowledge base is used both at the client and at the server. Its use by the client has two purposes:

- to customize the web page and user interface; and
- to permit handling of simple queries.

This new web architecture is illustrated in Figure 4.

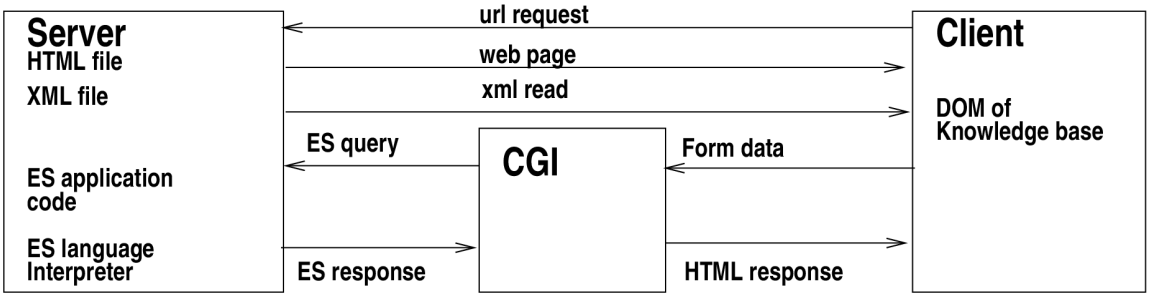

FIGURE 4: Hybrid WBES architecture.

An advantage of this web architecture is that the client is able to avoid referring simple queries to the server, while also avoiding the need for specialized software. Here, simple queries means those that can be handled by looking up DOM arrays of the knowledge base. Only a Javascript interpreter is required of the browser.

A small passenger information ES deployed in the hybrid architecture is now described. The knowledge base consists of a description of the passenger network in XML and stored in the file passenger.xml:

```
<?xml version=”1.0”>
<!DOCTYPE passenger_network SYSTEM “passenger_network.dtd”>
<passenger_network>
 <service>
  <name> coastal </name>
  <number> 312 </number>
  <connection>
   <source> walcha </source>
   <days> [monday, wednesday, friday] </days>
   <depart> 9:00 </depart>
   <destination> uralla </destination>
   <arrive> 10:15 </arrive>
  </connection>
  <connection>
```

```
      <source> uralla </source>
      <days> [monday, wednesday, friday] </days>
      <depart> 10:30 </depart>
      <destination> hillgrove </destination>
      <arrive> 11:05 </arrive>
    </connection>
  </service>
  <service>
    <name> inland </name>
    <number> 320 </number>
    <connection>
      <source> hillgrove </source>
      <days> [monday, saturday] </days>
      <depart> 19:10 </depart>
      <destination> guyra </destination>
      <arrive> 20:05 </arrive>
    </connection>
    <connection>
      <source> guyra </source>
      <days> [monday, saturday] </days>
      <depart> 20:20 </depart>
      <destination> walcha </destination>
      <arrive> 21:25 </arrive>
    </connection>
  </service>
</passenger_network>
```

A generic (to this domain) web page uses Javascript to read this data file from the server:

```
    if (window.XMLHttpRequest) {         // read the data file
      xhttp=new window.XMLHttpRequest()
    } else {
      xhttp=new ActiveXObject("Microsoft.XMLHTTP")
    }
    xhttp.open("GET","passenger.xml",false);
    xhttp.send("");
    xmlDoc = xhttp.responseXML;
```

The resulting xmlDoc can be used to access the elements of the knowledge base by straightforward access to data arrays. For example:

```
    stations = xmlDoc.getElementsByTagName( "source" );
```

The DOM can be used to provide a user interface customized for this knowledge base as shown in Figure 5 where information required to build html buttons and forms are extracted directly from DOM arrays. The DOM is also used to handle basic queries, for example displaying passenger service information about services from individual stations. This is also illustrated in Figure 5, where the services from the selected station are currently being displayed. That is, using Javascript, the client has the capacity to handle basic queries of the knowledge base without the need to refer to the server.

At the server, the XML knowledge base is converted to Prolog by an XML parser and stored in the file passenger.pl:

```
service( coastal, 312, [monday, wednesday, friday],
```

```
    connection( walcha, 9:00, uralla, 10:15 ),
    connection( uralla, 10:30, hillgrove, 11:05 ) ).
service( inland, 320, [monday, saturday],
    connection( hillgrove, 19:10, guyra, 20:05 ) ).
    connection( guyra, 20:20, walcha, 21:25 ) ).
```

More complex queries requiring the use of the inference engine are referred to the server via a CGI program. For example, should there be no direct connection between two locations it may be possible to find a route based on more than one connection and possibly from more than one service. The Prolog rule is:

```
route( Src, Src, [], _ ).
route( Src, Dest,
      [ (Src, Dtime, Dest1, Atime, Serv, Numb) | RestRoute ], Max ) :-
  Max > 0,
  Max1 is Max -1,
  route( Dest1, Dest, RestRoute, Max1 ),
  service( Serv, Numb, _, CL ),
  member( connection( Src, Dtime, Dest1, Atime ), CL ),
  not( member( ( _, _, Src, _, _, _ ), RestRoute ) ),
  write( Src ), write( ' to ' ), write(Dest1), nl.
```

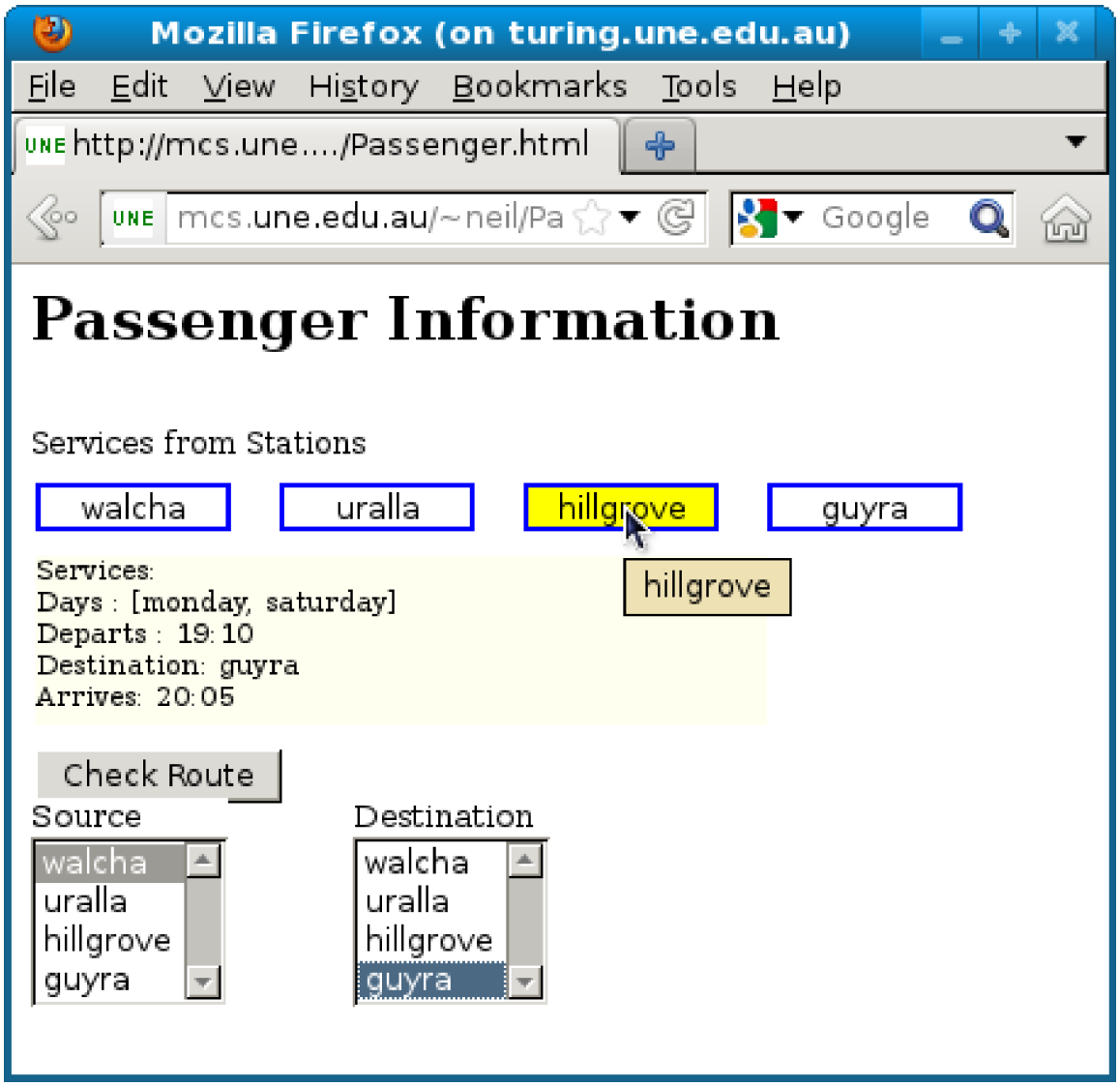

FIGURE 5: Web interface customized from the XML knowledge base.

A query to find a route from locations walcha to guyra limited to 4 stops has this response:

```
?- route( walcha, guyra, R, 4 ).
R = [ (walcha, 9:00, uralla, 10:15, coastal, 312),
      (uralla, 10:30, hillgrove, 11:05, coastal, 312),
      (hillgrove, 19:10, guyra, 20:05, inland, 320)] .
```

The CGI program constructs a Prolog query from the form data and interacts with the SWI-Prolog interpreter via a script:

```
#!/usr/bin/swipl -q -t main -f
% This script file loads passenger.pl and passnet.pl
% It executes a query given by the first command line arg
% Example: passenger_script 'route( walcha, guyra, R, 4 ).'
main :-
       ['passenger.pl'],
```

```
        ['passnet.pl'],
        current_prolog_flag(argv, Argv),
        append(_, [--|Args], Argv),
        concat_atom(Args, ' ', SingleArg),
        term_to_atom(Term, SingleArg),
        config_term_to_object(_,Term,Object),
        Object,
        halt.
main :-
        halt(1).
```

where passnet.pl contains the rule augmented with write statements to output the intermediate destinations. The flags of the script suppress all SWI-Prolog output except those from write statements. The CGI program then reads the SWI-Prolog output and constructs a response to the client in HTML. The response is shown in Figure 6. Of course the response could be more detailed and the passenger network more complex.

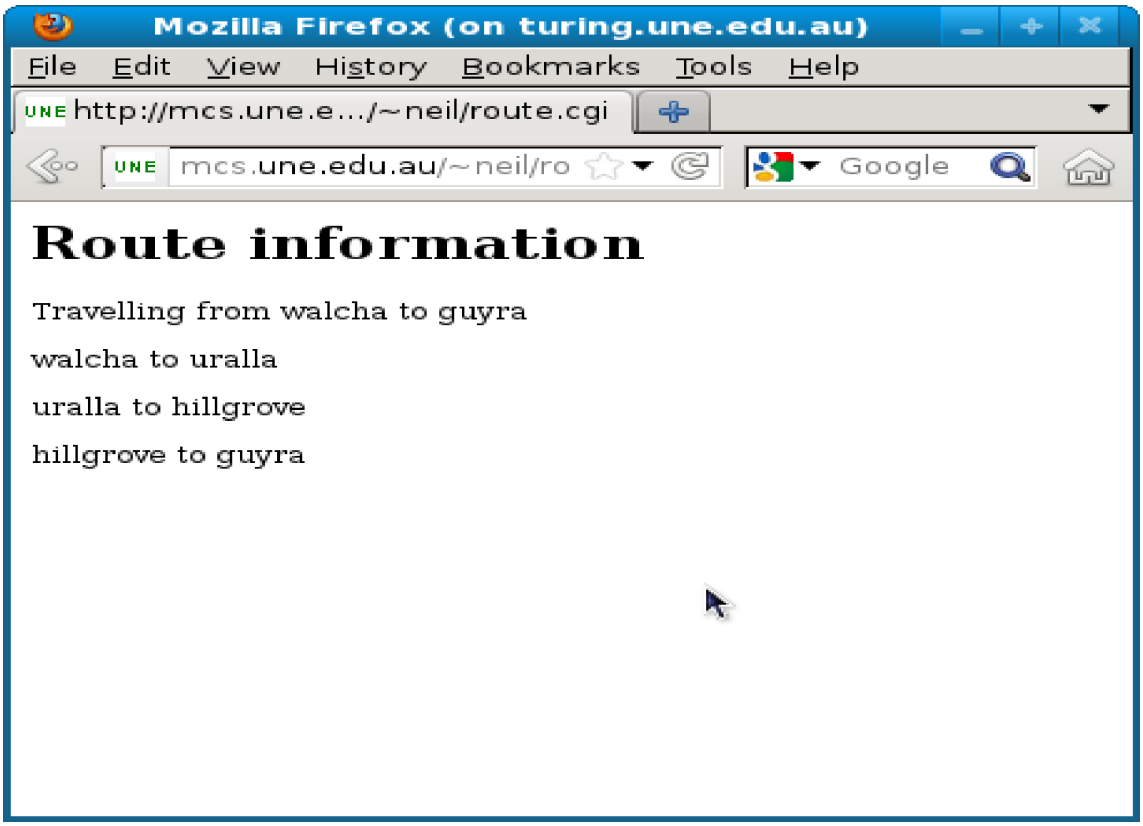

FIGURE 6: HTML response from the CGI program.

5. CONCLUSIONS

Although web-based expert systems are common and survey papers found in the literature provide comprehensive analysis of their respective features and methods of implementation, there has been a lack of discussion regarding their architecture in regard to how query handling is distributed between web client and server. Most applications deploy the expert system knowledge base at the server, although examples were found that are client-based. Both techniques have advantages and disadvantages although the server-side approach has the benefit of being able to guarantee expert system services over the web even to low capability

browsers and devices by tailoring its output accordingly. This paper has provided descriptions of a sample of applications with diverse architectures and implementation technologies.

A hybrid architecture is proposed that deploys the knowledge base at both client and server, allowing them to share the processing of user queries. A way to implement this technique is to use XML as the original form of representing the application's knowledge base. The XML file is read by the client web page Javascript code into a Document Object Model variable so that the structured data of the knowledge base is conveniently available. This permits both a customized web page to be constructed for the application as well as simple queries to be handled locally. This approach therefore only requires that the client browser has a Javascript interpreter rather than any more specialized software. The XML file is also translated into expert system language code at the server so that more complex queries can be handled there using specialized expert system language interpreters and auxiliary code. A small application was described to illustrate the method, using SWI-Prolog as the inference engine.

6. REFERENCES

[1] S. Loao, “Expert system methodologies and applications – a decade review from 1995 to 2004”, Expert Systems with Applications, 28(1), pp. 93-103, 2005.

[2] S. Kumar and B. Mishra. “Web-based expert systems and services”, The Knowledge Engineering Review, 25(2), pp. 167-198, 2010.

[3] K. V. Babu, R. G. Narayanan, and G. S. Kumar. “An expert system for predicting the behavior of tailored welded blanks”, Expert Systems with Applications, 37(12), pp. 7802-7812, 2010.

[4] F. Witlox. “Expert systems in land-use planning: an overview”, Expert Systems with Applications, 29(2), pp. 437-445, 2005.

[5] S. Helvacioglu and M. Insel. “Expert system applications in marine technologies”, Ocean Engineering, 35(11-12), pp. 1067-1074, 2008.

I. M. Dokas. “Developing web sites for web based expert systems: A web engineering approach”, in Proceedings of the Information Technologies in Environmental Engineering, September, Germany, 2005, pp. 202-217.

[6] Y. Duan, J. S. Edwards, and M. X. Xu. “Web-based expert systems: benefits and challenges”, Information & Management, 42(6), pp. 799-811, 2005.

[7] M. Grzenda and M. Niemczak. “Requirements and solutions for web-based expert systems” in, Artificial Intelligence and Soft Computing – ICAISC 2004, Lecture Notes in Computer Science, Volume 3070/2004, 2004, pp. 886-871.

Davison and S. W. Loke, S. W. “LogicWeb: Enhancing the web with logic programming”,
[8] The Journal of Logic Programming, 36(3), pp. 195–240, 1998.

[9] E. J. Friedman-Hill. “JESS, Java Expert System Shell”, Sandia Nation Laboratories, Livermore, CA, http://herzberg.ca.sandia.gov/jess, 1997, [May,2012].

[10] J. Wielemaker. “An Overview of the SWI-Prolog Programming Environment”, in Proceedings of the 13th International Workshop on Logic Programming Environments, 2003, pp. 1-16.

[11] S. W. Loke and A. Davison. “Secure Prolog Based Mobile Code”, Theory and Practice of Logic Programming,1(3), pp. 321-357, 2001.

[12] K. K. L. Ho and M. Lu, M. “Web-based expert system for class schedule planning using JESS”, in Information Re-use and Integration, Conf, IRI-2005 IEEE International Conference, 2005, 166-171.

[13] N. Dunstan. “ET: An Enrolment Tool for generating expert systems for university courses”, in Expert Systems, P. Vizureanu, Ed. Croatia: Intech, 2010, pp. 35-46.

[14] N. Dunstan. “Generating domain-specific web-based expert systems”, Expert Systems with Applications, 35(3), pp. 686-690, 2008.