ET: an Enrolment Tool to Generate Expert Systems for University Courses

Neil Dunstan
University of New England
Australia

1. Introduction

Expert Systems are in use today in many fields where there exists a well-defined problem domain (Giarratano & Riley, 2005). In this chapter, XML is used to help define a knowledge domain for academic course rules and used as the starting point for web-base expert systems.
Requirements for the satisfactory completion of university and college courses can be quite complex. Courses such as undergraduate bachelor degrees and postgraduate masters degrees are typically composed of units (sometimes called subjects) that must be completed according to the course rules. Such rules may impose constraints on the units that may be taken from specific groups of units, as well as constraints like prerequisite units and corequisite units. Many universities designate a human expert – the Course Coordinator, to guide students through their enrolment process to ensure that students' programs conform to course rules. In addition, many universities provide web-based descriptions of courses and units. However, such web sites are usually purely descriptive and lack a level of interaction with students that would enable answers to complex enrolment questions. It is therefore tempting to consider the automation of the course coordinator's role and its delivery. This chapter will provide a detailed description of the generation of a variety of expert system products intended to provide online advice to students about university bachelor and masters level courses. These products include course rules, unit descriptions, enrolment advice and course planners. They are designed following knowledge acquisition from experienced academic course coordinators about typical student queries in relation to their enrolment choices.
An XML Document Type Definition (DTD) will be described for university and college courses. It will be compatible with the European Credit Transfer System (EU, 2004), thus allowing a course to be composed of units with set credit points, term or semester of offering, and other unit constraints. Course rules may be expressed in terms of credit point requirements from groups of units. The XML data definition is sufficient to express the typical course requirement rules of higher education institutions such as universities and colleges.
The automatic generation of course-specific expert system products is accomplished via ET, an XML parser that translates XML course documents into Prolog predicates. This is the knowledge base. A Prolog interpreter then acts as the inference engine to solve course enrolment queries based on the course knowledge base.

Source: Expert Systems, Book edited by: Petrică Vizureanu,
ISBN 978-953-307-032-2, pp. 238, January 2010, INTECH, Croatia, downloaded from SCIYO.COM

The expert system products are intended for use by the students. A user-friendly web-based interface is described. The XML parser can also generate course-specific HTML and CGI files that interact with the Prolog interpreter to report the results of enrolment queries through a web browser in a use-friendly fashion.
There is an interesting synergy of XML and Web technologies, following the adoption of XHTML (W3C, 2000) and growing browser support for XML (Ciancarnini et al., 1998; Ghislain et al., 2009). Consideration is given to the place of the course XML DTD in the higher education knowledge domain.

2. Discussion.

A prototype ET was used to describe the method in (Dunstan, 2008) for generating domain-specific web-based expert systems. That method used Open Source products Linux (Moody, 2001), Perl (Wall & Schwartz, 1992) and SWI-Prolog (Wielemaker, 2003) and a domain-specific XML parser to generate a web-based expert system. The prototype produced a single web page with limited functionality and used low-level parsing techniques.
A recommender system that matches student interests with elective units was proposed in (O'Mahony & Smyth, 2007) and decision support for enrolment at the institutional level is described in (Maltz et al., 2007). Current generation university and college web sites provide only minimal online support for the complex task of guiding student enrolment. Web services typically only provide tables of rules and unit descriptions. ET produces expert systems with a web-based user interface.
An overview of requirements for expert systems on the internet is described in (Grzenda & Noemczak, 2004) and (Caldwell et al., 2003) compares two possible architectures. There have been many approaches to building web-based expert systems (Li et al., 2002; Li, 2005; Huntington, 2000; Riva et al., 1998). They propose solutions in isolation. The approach of the ET project features an XML DTD and takes advantage of the growing synergy of XML and Web technologies. Many browsers include XML parsers and are able to display raw XML files in a tree fashion at least, or use style guides to present XML data. Web-based applications can request XML data files from remote servers for processing and presentation. The XML DTD provides a vocabulary, namespace and data file structure for Web and XML applications that use academic course rule data.

3. Rules, groups and units in XML

University and college courses such as Bachelors and Masters degrees are composed of units (or subjects taught over teaching periods called semesters or terms) that are selected from groups according to course rules. Grouping of units might be by year level, subject theme, or even just to distinguish between compulsory and non-compulsory. There may be further constraints on unit selection such as prerequisites requirements. Units may carry different credit point values towards a total required for course completion. A course document type definition is needed to standardize course information for applications, including data transfer across the internet. Here is the XML DTD for academic course rules.

```
<?xml version="1.0"?>
<!ELEMENT course (aname, acode, adescr, rules, group+, unit+, prog*)>
<!ELEMENT rules (maximum*, minimum*)>
```

```
<!ELEMENT group (gname, gunits)>
<!ELEMENT unit (uname, udescr, usem, uprer, ucore,urest, ucps)>
<!ELEMENT prog (pname, punits) >
<!ELEMENT maximum (#PCDATA)>
<!ELEMENT minimum (#PCDATA)>
<!ELEMENT aname  (#PCDATA)>
<!ELEMENT acode  (#PCDATA)>
<!ELEMENT adescr  (#PCDATA)>
<!ELEMENT gname  (#PCDATA)>
<!ELEMENT gunits  (#PCDATA)>
<!ELEMENT uname  (#PCDATA)>
<!ELEMENT udescr  (#PCDATA)>
<!ELEMENT usem  (#PCDATA)>
<!ELEMENT uprer  (#PCDATA)>
<!ELEMENT ucore  (#PCDATA)>
<!ELEMENT urest  (#PCDATA)>
<!ELEMENT ucps  (#PCDATA)>
<!ELEMENT pname  (#PCDATA)>
<!ELEMENT punits (#PCDATA)>
```

A *course* document is composed of these elements:

aname	: a name,
acode	: a code,
adescr	: a short description,
rules	: rules governing course requirements
group	: one or more groupings of units
unit	: one or more units
prog	: zero or more recommended programs

In the *rules* section, there can be any number of *maximum* or *minimum* elements. The text for each one consists of *group_name : credit_points*. For example:

```
<maximum>
firstYear : 36
</maximum>
```

meaning that at most 36 credit points can be counted for units from the group *FirstYear*. For *minimum*, the meaning is that at least that number of credit points must come from that group.
A *group* has a name and a list of unit names that belong to the group. For use in *maximum* and *minimum*, the group name *all* indicates a maximum or minimum number of credit points required from all groups. A unit has

uname	: a name or code,
udescr	: a short description that should include keywords for searching,
usem	: a list of semester names or numbers when the unit is offered,
uprer	: a list of unit names that are required before enrolment in this unit,
ucore	: a list of unit names that must accompany enrolment in this unit,
urest	: a list of unit names that may not be in a program with this unit,
ucps	: the number of credit points towards course completion.

For convenience when converting the XML to Prolog, the lists are in Prolog list form, that is:

[name1, name2, ... ,namen]

For example: here is the XML data for the unit *stat354*

```
<unit>
  <uname> stat354 </uname>
  <udescr> 'Distribution Theory and Inference' </udescr>
  <usem> [1] </usem>
  <uprer> [stat260, pmth212] </uprer>
  <ucore> [] </ucore>
  <urest> [] </urest>
  <ucps> 6 </ucps>
</unit>
```

A *prog* is a recommended program meant to represent a valid program for the course that focuses on a particular interest area or theme. It has a name and a list of unit names that belong to the program.

4. The parser

ET version 2 is an XML parser based on the Perl XML::Parser module. It processes the XML data for a course and:

- translates the XML data into Prolog rules and facts,
- generates web modules for use as enrolment guides,
- generates a SWI-Prolog script file to execute Prolog queries.

Fig 1. shows the operation of ET.

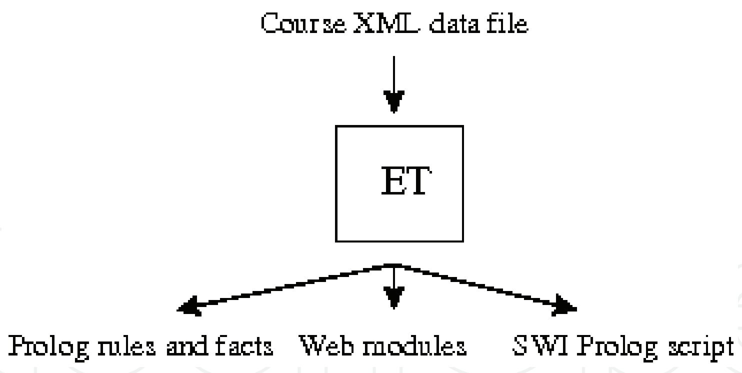


Fig. 1. ET operation.

The ET usage message is:

```
Usage: ET course.xlm -ieprl
        i rules and units
        e enrolment guide
        p course planner
        r recommended programs
        l output in indented xml layout
or Usage: ET course.xlm
        for the lot.
```

The options permit selective re-generation of web modules. The web modules consist of HTML and CGI files to generate user-interface pages and CGI programs to execute queries. Some HTML files are used to establish a home page for this course. These files are described in Table 1.

File name	Description
{coursecode}.html	Establish a frameset with title, navigation and display frames
links.html	Navigation panel with links to modules
title.html	Display the course title and description
blank.html	Initial blank page in display frame.

Table 1. Course HTML files

ET produces files for each web module required. These files are described in Table 2.

Web Module	File name	Description
Rules and units	*rulesunits.cgi*	General rule and unit information page
	allrules.cgi	Get course rules
	allunits.cgi	Get all unit information
	findkeyword.cgi	Find a keyword in a unit description
	unitinfo.cgi	Get unit information
	prerchain.cgi	Get the chain of prerequisite units
Enrolment guide	*enrolguide.cgi*	Enrolment guide page
	required.cgi	Get requirements to complete course
Planner	*planner.cgi*	Course planner page
	check_plan.cgi	Check a program against course rules
Recommended programs	*recomprog.cgi*	Recommended programs page
	findprog.cgi	Find a recommended program that includes chosen units
	showprog.cgi	Show a program

Table 2. Course Web modules.

Other files are listed in Table 3.

File name	Description
{acode}.pl	Prolog rules and facts for the course
auxiliary.pl	Auxiliary Prolog functions
{acode}_script	SWI-Prolog script file
{acode}.css	Cascading Style Sheet for the web site.

Table 3: Other Course files

The Cascading Style Sheet file describes the presentation of ET Web modules. This provides some web site customization.

5. The prolog

ET converts the XML course rules and units into Prolog predicates. A group of units from the XML data file such as:

```
<group>
  <gname> firstYear </gname>
  <gunits>
    [comp131, comp132, comp100, comp170,
     amth140, comp160, maths101, maths102]
  </gunits>
</group>
```

becomes:

```
get_group( M, firstYear ):-
  M = [comp131, comp132, comp100, comp170,
         amth140, comp160, maths101, maths102].
```

A rule such as:

```
<minimum> firstYear : 36 </minimum>
```

is included in a predicate to check a program against all the rules of the course, such as:

```
check_rules( N ):-
  get_group( G1, firstYear ),
  check_min( N, G1, firstYear, 36 ),
  ....
```

where *N* is a list of unit names and the intention is to check that the program represented by *N* contains units from group *firstYear* with a minimum of 36 credit points. If a rule is not satisfied the checking predicate outputs the reason. For example, the query

```
check_rules( [comp131, comp160] ).
```

has the response:

```
At least 24 more credit points required from Group firstYear:
comp131 comp132 comp100 comp170 amth140 comp160 math101 math102
```

along with the output from other rule predicates, such as:

```
At least 132 more credit points required altogether.
```

An XML unit representation such as that of *stat534*, shown in section 3, becomes in Prolog:

```
unit( stat354, 'Distribution Theory and Inference',
      [1], [stat260, pmth212], [], [], 6 ).
```

Auxiliary predicates include those that check prerequisites and other constraints, as well as providing answers to targeted queries. Here is an example:

```
prer_chain( [], A, A ).
prer_chain( [ H | T ], A, D ):-
  not( member( H, A ) ),
  unit( H, _, _, P, _, _, _ ),
  append( A, [H], B ),
  write( H ), write( ' :has prerequistes: ' ),
  show( P ),
  prer_chain( P, B, C ),
  prer_chain( T, C, D ).

prer_chain( [ H | T ], A, D ):-
  member( H, A ),
  unit( H, _, _, P, _, _, _ ),
  prer_chain( P, A, B ),
  prer_chain( T, B, D ).
```

This predicate will report on the prerequisites required by the units in the list of the first parameter, and the prerequisites required of those units. That is, it recursively finds all units required directly or indirectly by the units in the list.
Queries are put to the Prolog course file via an SWI-Prolog script:

```
#!/usr/bin/pl -q -t main -f
% bcompsci_script : for bcompsci enrolment guide
% Generated by ET version 2
% This script file uses auxiliary.pl
% This script file loads bcompsci into pl and executes
% a query given by the first command line arg
% Example: bcompsci_script 'check_rules( [ ] )'
main :-
      [award-aux],
      [bcompsci],
      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).
```

The script supresses all output except the application output from *write* statements in the predicates. An example of running a query using the script is:

```
$ award_script 'prer_chain( [stat354], [], [] )'
stat354 :has prerequisites:  stat260 pmth212
stat260 :has prerequisites:  math102
math101 :has prerequisites:
math102 :has prerequisites:  math101
pmth212 :has prerequisites:  math102
```

That is, *stat534* has prerequisites *stat260* and *pmth212*. In turn, *stat260* has prerequisite *math102*. And *pmth212* also had as prerequisite *math102,* which in turn has prerequisite *math101*, which has no prerequisites. Note that the output lists the prerequisites of each required unit only once. This invites the comparison between presenting a flat table of unit details and the capabilities of an expert system. The expert system is able to infer indirect requirements.

6. Web interface

The Web modules are a graphical user interface to the expert system. Instead of having to enter queries in Prolog syntax, users enter information into forms whose actions initiate appropriate queries and display results. That is, the web modules read form data, convert it into Prolog queries that are put to the expert system using the SWI-Prolog script. Responses are displayed to the user. This process is illustrated in Fig 2.

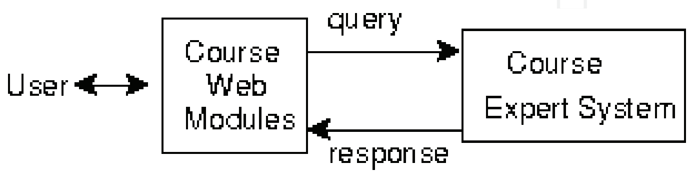


Fig. 2.Web interface

The Rules and Units module displays a web page like Fig 3.

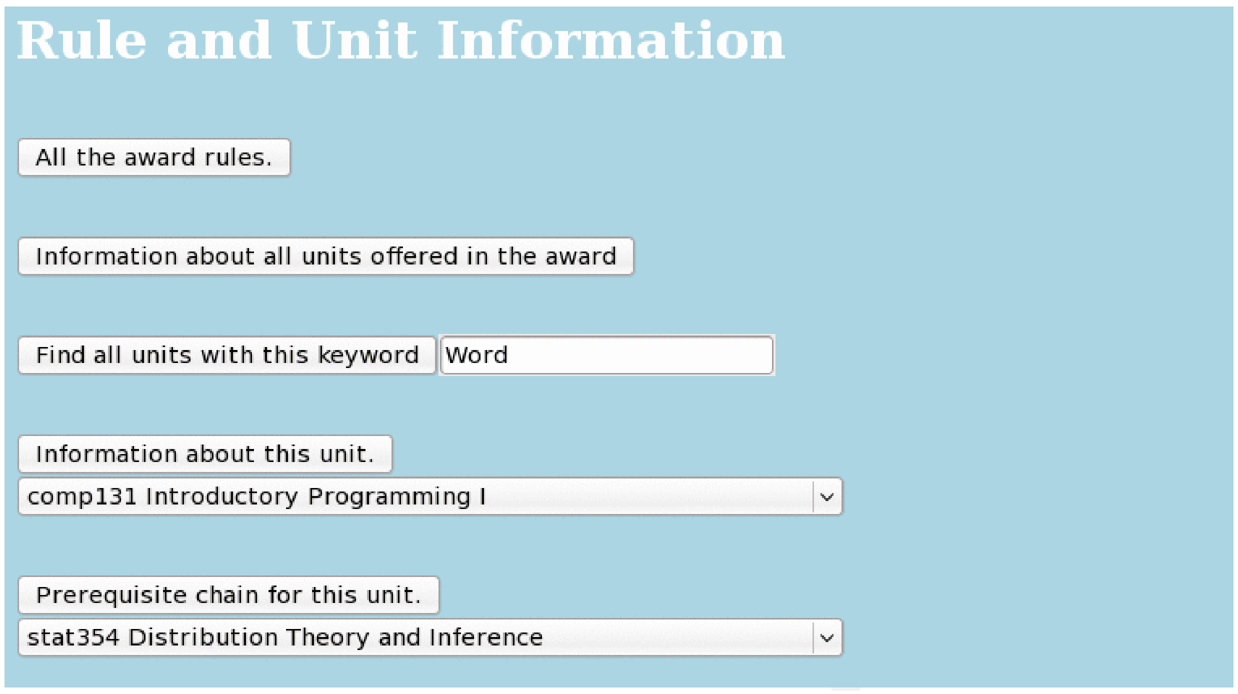
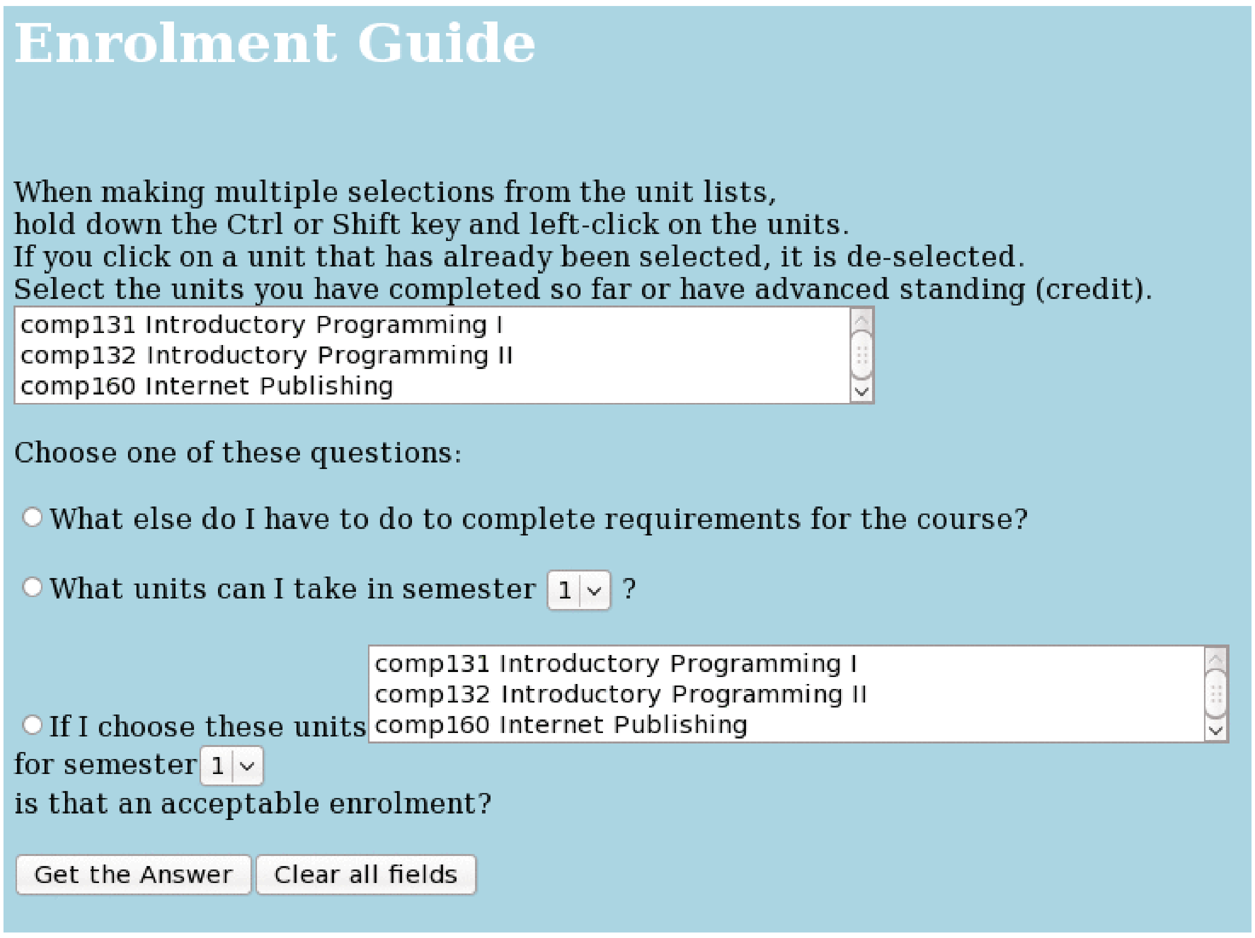

Rule and Unit Information
All the award rules.
Information about all units offered in the award
Find all units with this keyword
Word
Information about this unit.
comp131 Introductory Programming I
Prerequisite chain for this unit.
stat354 Distribution Theory and Inference

Fig. 3. Rule and Unit Information Web interface

The user selects or enters required information and clicks on the buttons. Should the user select unit *stat354* and click *Prerequisite chain for this unit*, the form action executes *prerchain.cgi*, which executes the query:

prer_chain([stat354], [], []).

via the script, and displays the application output response on screen.
The Enrolment guide web interface is shown in Fig 4. It enables three questions that are based on the list of units already completed and the course rules. The Course Planner


Enrolment Guide

When making multiple selections from the unit lists,
hold down the Ctrl or Shift key and left-click on the units.
If you click on a unit that has already been selected, it is de-selected.
Select the units you have completed so far or have advanced standing (credit).

comp131 Introductory Programming I
comp132 Introductory Programming II
comp160 Internet Publishing

Choose one of these questions:

○ What else do I have to do to complete requirements for the course?

○ What units can I take in semester 1 ?

○ If I choose these units comp131 Introductory Programming I / comp132 Introductory Programming II / comp160 Internet Publishing
for semester 1
is that an acceptable enrolment?

Fig. 4. Enrolment guide module web interface

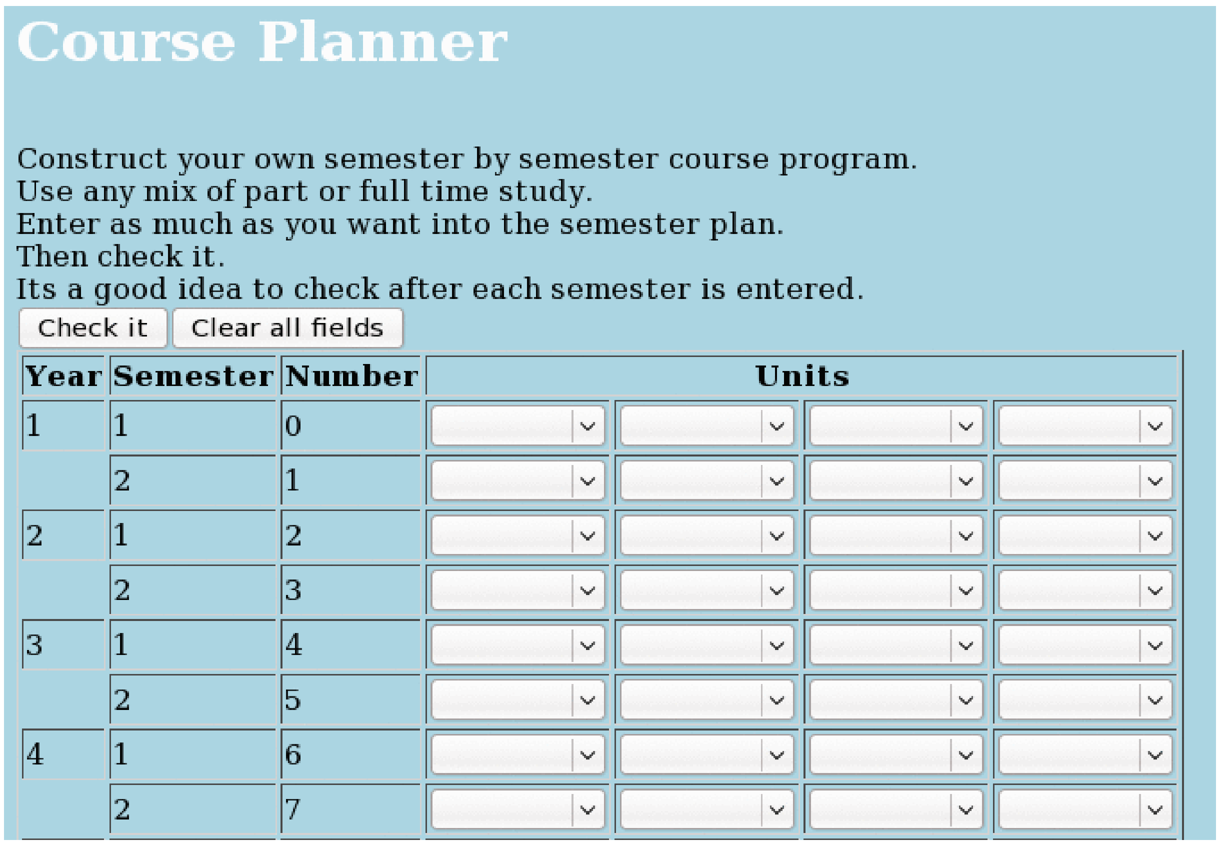
module web interface is shown in Fig 5. It permits free-hand construction of a valid program of study scheduled over a period of years. The Recommended Programs web interface is shown in Fig 6. It permits the searching of valid recommended programs to find those with units of interest to students.
The web sites generated by ET have few HTML elements and conform to the XHTML 1.0 transitional DTD. In order to provide a customizable and consistent presentation a site style sheet is provided:

```
/* style sheet for ET html documents */
body         {background-color: lightblue; color: black}
input        {background-color: white}
input:focus  {background-color: yellow}
label        {color: red}
h1           {color: white}
```

It is also desirable to provide a style sheet for the raw XML data files.

7. Conclusion and future work

The potential of web-based expert systems as knowledge servers was recognised in (Erikson, 1996). An XML DTD was developed to represent academic course rules and formalize this knowledge domain. The DTD can be used to validate course rule documents

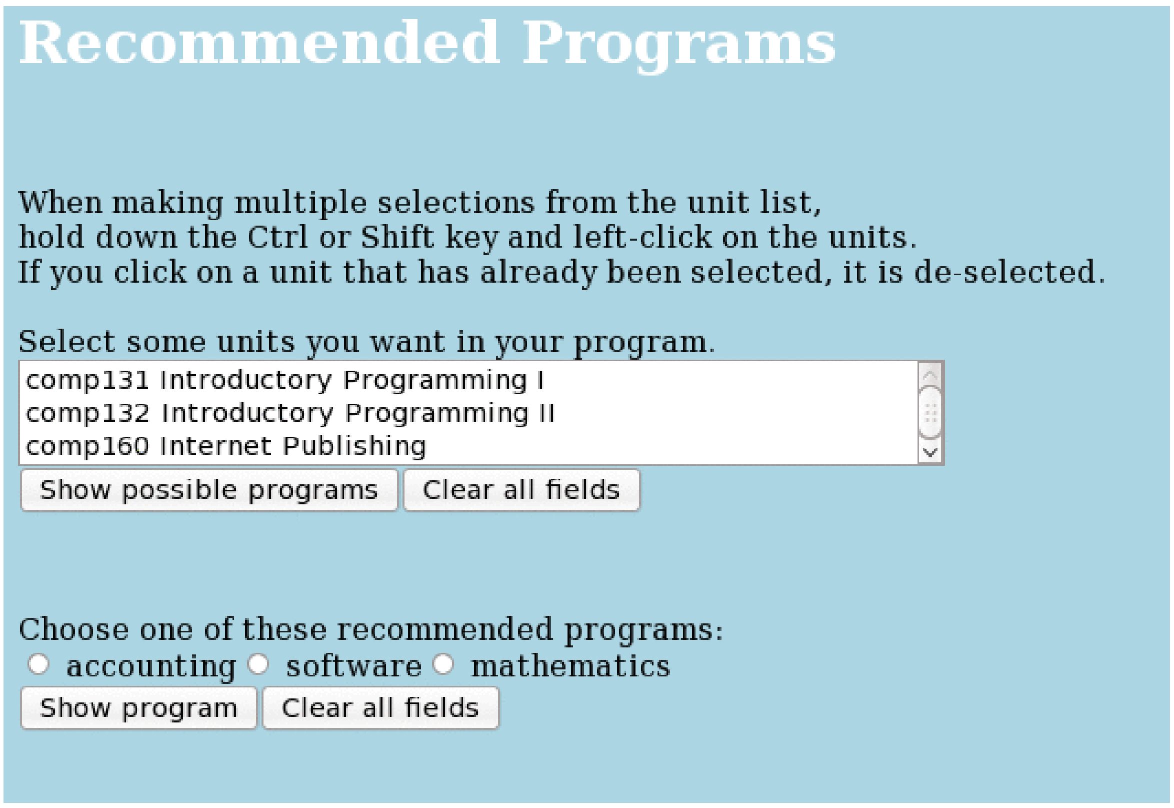


Course Planner

Construct your own semester by semester course program.
Use any mix of part or full time study.
Enter as much as you want into the semester plan.
Then check it.
Its a good idea to check after each semester is entered.

Year	Semester	Number	Units			
1	1	0				
	2	1				
2	1	2				
	2	3				
3	1	4				
	2	5				
4	1	6				
	2	7				

Fig. 5. Course planner web interface



Recommended Programs

When making multiple selections from the unit list,
hold down the Ctrl or Shift key and left-click on the units.
If you click on a unit that has already been selected, it is de-selected.

Select some units you want in your program.

comp131 Introductory Programming I
comp132 Introductory Programming II
comp160 Internet Publishing

Choose one of these recommended programs:

Fig. 6. Recommended programs web interface

and provide a standard format for representing course rules. The DTD thus supports application programs that process academic course rules. Applications may be web-based since there is script language support for fetching XML data files from remote servers, such as the *JavaScript XMLHttpRequest*. The DTD provides a *namespace* and vocabulary for such web-based applications. The adoption of the DTD by a network of university and college servers hosting XML course data files would enable comparative studies of courses and programs. Simply publishing XML course data files on the Web with a suitable style sheet to assist browser presentation would be an effective way to distribute course information.
An XML parser was described that converts a course rule data file to a Prolog knowledge base. It also generates web modules that serve as a graphical user interface to expert system. The web modules permit users to pose complex queries about their proposed enrolment choices. The range of types of queries is limited but future research will investigate:

- the addition of new types of enrolment queries
- new web-based interface methods for posing queries.

The open source products used to implement ET proved to be convenient and powerful. Perl continues to be an effective language for writing CGI scripts and SWI-Prolog is a mature implementation of the popular Prolog language for building expert systems.
Web-based applications of course data files are now planned with a view to supporting comparative studies of courses both within universities and across the higher education sector.

8. References

Caldwell, N. H. M., Breton, B. C. and Holburn, D. M. (2003). Information clients versus knowledge servers, In: *Studies in Fuzziness and Soft Computing: Intelligent exploration of the Web,* 402-417, Physica-Verlag, ISBN: 3-7908-1529-2, Heidelberg.

Ciancarnini, P., Rizzi, A. And Vitali, F. (1998). An extensible rendering engine for XML and HTML, *Computer Networks and ISDN Systems,* Vol. 30, Issues 1-7, (April, 1998), 225-237, ISSN: 0169-7552.

Dunstan, N. (2008). Generating Domain-specific Web-based Expert Systems, *Expert Systems with Applications*, Vol. 35, No. 3, (October, 2008), 686-690, ISSN: 0957-4174.

Eriksson, H. (1996). Expert systems as knowledge servers, *IEEE Expert: Intelligent Systems and their Applications*, Vol. 11, No. 3, (June, 1996), 14-19, ISSN: 0885-9000.

European Union (EU) (2004), ECTS User's Guide – European Credit Transfer System for Lifelong Learning, European Commision, 2004.

Ghislain, F., Pilman, M., Floresu, D., Kossmann, D., Kraska, T. And MacBeath, D. (2009), Xquery in the browser, *Proceedings of the 18th International Conference on WWW,* pp. 1011-1020, Madrid, April, 2009.

Huntington, D. (2000), Web-based expert systems are on the way: Java-based Web delivery, *PC AI Intelligent Solutions for Desktop Computers,* Vol. 14, No. 6, (June, 2008), 34-36.

Grzenda, M. And Niemczak, M. (2004). Requirements and solutions for web-based expert systems, In: *Lecture Notes in Artificial Intelligence 3070,* Rutkowski, L. et al. (Ed.), 866-871, Springer-Verlag, Berlin.

Maltz, E., Murphy, K. E. And Hand, M. L. (2007)). Decision support for university enrolment management: implementation and experience, *Decision Support Systems*, Vol. 44, No. 1, (November, 2007), 106-123.

Moody, G. (2001). *Rebel code: Linux and the open source revolution,* Perseus Publishing, ISBN: 0738206709.

O'Mahony, M. P. And Smyth, B. (2007). *A recommender system for online course enrolment: an initial study, Proceedings of the 2007 ACM Conference on Recommender Systems,* pp. 133-136, ISBN: 978-1-39593-730-8, ACM, Minneapolis.

Riva, A., Bellazzi, R., and Montani, S. (1998). A knowledge-based Web server as a development environment for Web-based knowledge servers. In *EIEE colloquium on web-based knowledge servers (Digest No 1998/307).*

W3C (2000). XHTML 1.0 The Extensible Hypertext Markup Language (second edition) : A Reformulation of HTML 4 in XML 1.0. Retrieved from www.w3.org/TR/xhtml1/ in Ocober, 2009.

Wall L., Christainsen, T. and Schwartz, R. L. (1996). Programming in Perl, O'Reilly and Associates, ISBN: 1-56592-149-6, USA, Second Edition, 1996.

Wielemaker, J. (2003). *An overview of the SWI-Prolog programming environment (Vol. CW371, pp. 1-16).* WLPE Report. Katholeike Univiersiteit Leuven, Department of Computer Science, Celestijnenlaan 200A, B3001 Heverlee (Belgium).