# Case Study: A Course Advisor Expert System

**Conference Paper** *in* Lecture Notes in Computer Science · December 2003

DOI: 10.1007/978-3-540-24581-0_87 · Source: DBLP



**1 author:**

Ovidiu Noran
Griffith University
**91** PUBLICATIONS **637** CITATIONS

SEE PROFILE

# Case Study: A Course Advisor Expert System

**Abstract.** This article presents the development of a knowledge-based expert system for a non-trivial application domain, i.e. selecting a customised postgraduate study program from existing- and targeted sets of skills. Following a customised spiral development paradigm, the paper describes the problem domain, followed by the concepts and requirements involved in the expert system design. The main design constraints are stated and the design decisions are justified in light of the specific task. Attention is paid to the knowledge representation / acquisition and rule base design processes. Implementation and deployment are explained, together with a sample run and result evaluation. Finally, further work is detailed in light of the characteristics and limitations of the prototype.

**KEYWORDS**: Constraints, Expert Systems, Knowledge Acquisition, Knowledge Representation



## Introduction

The changes brought about by the Information and Communication Technology (ICT) revolution on the human society have also had an impact on the required set of skills of the workforce. At most levels of human activity, ICT skills are nowadays a prerequisite. While for the recently qualified professionals basic ICT knowledge has been built into the education process, the more mature workforce has to acquire this knowledge separately and in 'backwards compatibility' mode with their existing skills.

Particularly at higher levels of education, individuals may already possess some of the required skills. These abilities mainly exist in the form of tacit knowledge, formal/informal awareness, practical skills, etc that may have been acquired during past undergraduate studies, or current professional activity[1]. This knowledge, although useful, is not structured in a consistent way (as it would be e.g. as a result of a formal study program[2]). The potential exists however for this type of existing knowledge to be reused in the process of acquiring and formalizing the necessary ICT skills to the benefit of the individual undertaking the learning process.

There are several potential hurdles in the reuse of previous knowledge, such as:

[1] for example, most engineers have undertaken at least one course of structured programming during their undergraduate studies.

[2] in the following, the terms 'study program' and 'course' will be used interchangeably.

- eliciting and self-assessing the existing knowledge: the individual has to be aware of the skills it possesses (not trivial in all cases) and to unambiguously communicate them;
- devising a flexible learning process to take advantage of the identified skills;
- correctly *matching* a study program to a particular set of existing- and targeted skills.

This paper illustrates the problem through a case study describing the use of an expert system to customise the necessary study program for particular sets of existing- vs. targeted knowledge. The case study is limited to an expert system *prototype*, dealing with postgraduate study enrolment involving students with a non-IT background who wish to acquire ICT formal education in an accredited University environment. For the purpose of this paper, the study program will be called a *Conversion Course*[3]. The system may offer guidance to both potential students and course designers in regard to the necessary structure of a particular study program.

## The Problem Domain

The present University course description Web-based system exhibits some limitations. Occasionally, subject pages display erroneous or stale information. Students may be unintentionally misled in their enrolments / exams etc which could adversely affect the image of the teaching institution. Also, the present system does not verify that a student may enrol in subjects in a way that may contradict University policies[4]. Last but not least, the present web-based subject and course descriptions may be confusing for prospective students[5].

The advisory expert system is intended to provide preliminary assistance and advice for prospective postgraduates, extracting and inferring the necessary information from a current knowledge base within a consultation session. This should provide a realistic assessment of the alternatives, requirements and expectations for a particular student, and thus help prevent enrolment errors.

In the current teaching system, subjects are considered to be an indivisible entity, non-customisable, with a fixed number of points awarded on completion. Even if a student has prior knowledge covering part of the course, he / she still has to enrol in the *whole* course and will be awarded *all* of the points allocated to the course if successful. In order to enable and support the proposed expert system, the teaching system should allow course *modularity*, with points allocated to course components, separately assessed.



[3] the terminology used for a single study unit vs. a complete study program is University-specific. Therefore, in the following it is assumed that subject is a single study unit (e.g. 'Programming 2') while course is a collection of subjects leading to the awarding of a degree (e.g. 'Master of Information Technology').

[4] presently, students are responsible for the accuracy of their enrolment.

[5] e.g. which subjects are compulsory or elective for which course, or a particular order in subject enrolment, etc.

## Developing the Expert System.

As previously stated, in developing the expert system it is desirable to start with a *prototype* of the complete expert system, which would provide information on the feasibility of the project without full financial or resource commitment. This prototype may then be submitted for evaluation to users / stakeholders and so obtain their feedback and commitment. User and host institution acceptance is a multidimensional aspect [15], which ultimately decides the usefulness of the entire system development effort. It must be also emphasized that the prototype needed the elicitation of knowledge from at least one domain expert [6] (in this case, a subject convener). The development of the prototype could not be left to the knowledge engineer alone[6].

### Life Cycle Models

Several life cycle paradigms have been considered such as waterfall, incremental, linear, spiral,, etc [6], [12], [13]. A modified *spiral* paradigm may be successfully employed, providing that some guidelines are followed regarding the newly added facts and rules. Such constraints may be: maintain integrity (must not contradict the existing facts and rules), avoid redundancy (must not represent knowledge already existent in the rule base), prevent scattering the knowledge over an excessive amount of rules / facts, etc. Thus, a balance must be struck between the facts / rules complexity, expressivity and number[7].

### Concepts of the Expert System

The expert system will be based on several concepts aiming to improve the subject selection and education processes. These concepts are *modularity* of subjects[8], *prerequisites* and *outcomes* for subject modules and *credit* for previous studies, applied to modules rather than subjects.

Hence, the aim is to establish *modules* within the subjects[9], with their own prerequisites, outcomes and credit points awarded on completion. The number of



[6] this may be allowable if the knowledge engineer is also a domain expert, i.e. a course / subject convener. In this case however, some method of triangulation should be employed (e.g. member checking).

[7] i.e. simpler rules are less complex and easier to understand / debug, but a lower level of complexity may result in a large number of rules. In terms of programming languages, this boils down to fewer, nested IF/THEN/ELSE clauses vs. more non-nested IF/THEN(/ELSE) clauses.

[8] although currently subjects are seen as indivisible entities, they do display a certain *structure* (e.g. as described in the subject syllabus) which may be used to assist decomposing the subjects into modules

[9] using knowledge acquisition methods as further detailed in this paper

modules within a subject must maintain a balance between flexibility and the amount of processing and development time required[10].

**User Requirements for the Expert System Prototype**

The user will provide a preferred type of occupation (targeted set of skills) and previous knowledge, usable to satisfy some of the prerequisites for the modules composing the study program. The system will provide a study course to achieve the targeted occupation and may also give correction advice if existing user skills are stated as too limited or too high.

**Design of the Expert System Prototype**

**System Requirements**

The system requirements represent a translation of the user requirements into the system domain[11]. For this particular case they may take the form:

- the expert system must rely on the user's tacit knowledge[12] rather than trying to model it;
- modules are seen as *objects* having *interfaces* - which are in fact the prerequisites and outcomes of a module. Prerequisites and outcomes are items contained in special *lists*, which are further contained within *module facts*;
- a module prerequisite may be satisfied by one (and only one) outcome, be it of another module or declared as 'known' by the user (previously acquired skills);
- the consultation starts with a set of initial facts, asserted at run-time according to the user's answers to 'job domain' and 'job type' queries. These facts provide the initial list of unsatisfied prerequisites.
- the system attempts to match these unsatisfied prerequisites: first, against outcomes declared 'known' by the user, and then (if unsuccessful) against the outcomes lists of the other module facts in the knowledge base;
- when a matching outcome is found for a prerequisite, the module owning the outcome is marked as 'needed' and its *prerequisites* list is then in its turn scanned for matches with other outcomes. The prerequisite for which the match has been found is marked 'done'.
- any prerequisites found to be not either declared 'known' by the user, or already satisfied by other modules' outcomes will trigger *additional questions* for the user;
- according to the user's answers, the prerequisites are either declared 'known' (if the user happens to have the required skill), or added to the list of unsatisfied prerequisites;

[10] a small number of modules defeats the purpose of the entire effort, while a large number of modules may require too many resources.

[11] the accuracy of this translation may be subsequently checked in the process of *validation* with the end user.

[12] i.e. basic knowledge assumed owned by a prospective postgraduate student - e.g. as in [5]

- the process is repeated until all prerequisites are satisfied - either by other modules' outcomes ('done') or by previous skills of the user ('known').
- the list of all modules marked as 'needed' (that is, modules the user will need to enrol in) is printed out[13]. **NB** some special modules are automatically added, such as the project modules (which will be present regardless of the user's prior knowledge)

The system functionality described above is shown diagrammatically in **Figure 1**.

**Knowledge Representation and Method of Inference**

Design decisions had to be made regarding the formalism to be used in representing the knowledge contained in the teaching system. A first criterion is related to the type of knowledge to be represented, namely the components of the various courses composing current study programs. As such, this knowledge matched the form of a collection of isolated facts, rather than a structured set of knowledge or a concise theory. Therefore, rules / assertions were preferred to frames or algorithms in a first decision round [16]. Production rules [21] have been chosen in preference to logic and semantic nets / frames [19], since they are more suitable for solving design and planning problems [17].

Bottom-up reasoning / inference, whereby a starting fact set (provided by the user) is matched against the conditions of the production rules (in the rule base, constructed upon the rules governing the study programs within the University) will also be used. The use of bottom-up inference leads to forward chaining as a preferred model of conflict resolution. However, prioritisation may also be involved by assigning production rules appropriate weights [17].

**Knowledge Acquisition**

The preferred methods of knowledge acquisition for the prototype have been chosen to be the use of questionnaires and structured interview. This decision owes to several factors such as the size of the prototype, the nature of problem domain and the availability of domain experts. Questionnaires are well suited to future automated processing which is likely to benefit the knowledge acquisition process [2]

The design of both questionnaire and interview have acknowledged the gap between the descriptions of domain specialists (subject conveners) and the resulting computational models [4, 18] and also the social issues underlying knowledge creation [10][14]. The interview design has mainly followed the COMPASS procedure [22] .



[13] at this stage, depending on the granularity of the subject / module decomposition, some 'needed' modules may have outcomes which are not used to satisfy any prerequisites. These outcomes come rather as 'bonus knowledge'.

[14] the items composing the questionnaire and the interview structure must also take into consideration the *culture* (e.g. the shared values, beliefs, etc) and policies of the institution hosting the domain experts and the system.

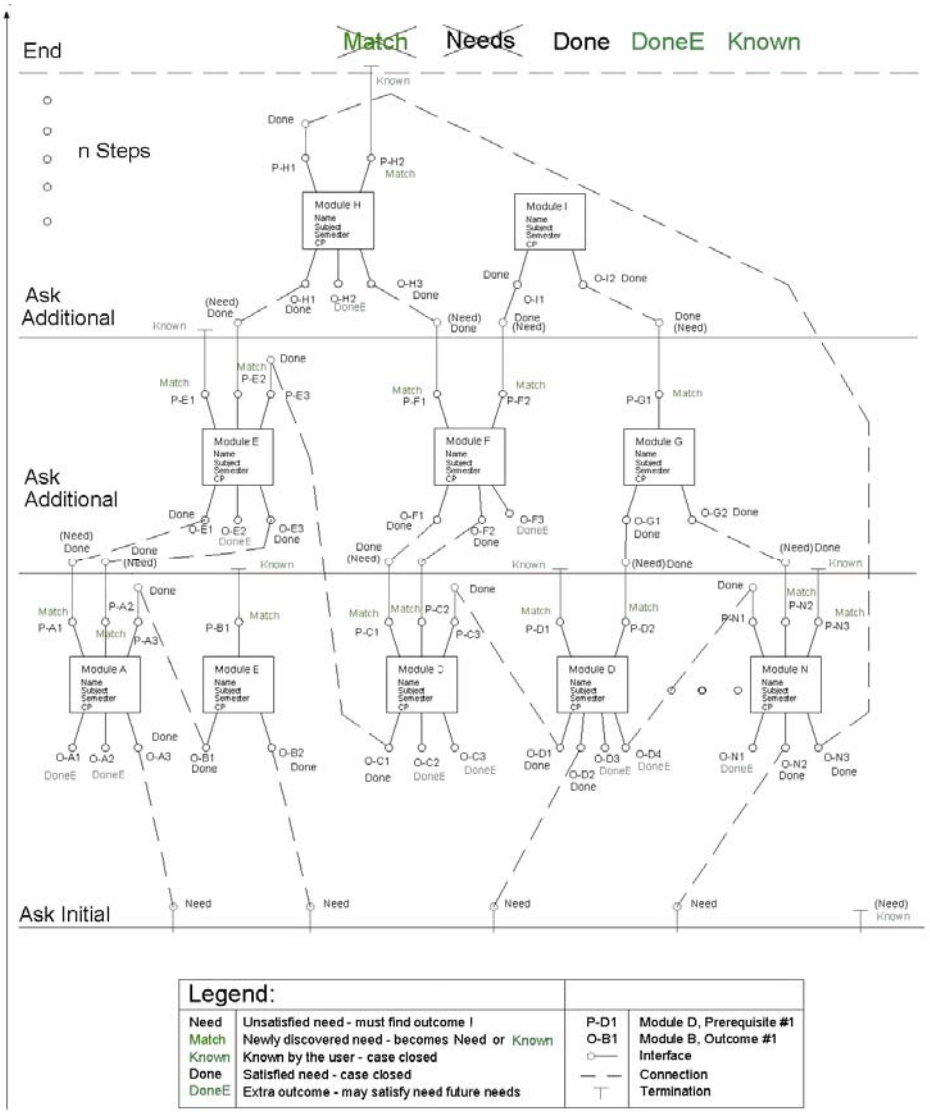



**Figure 1.** Expert system prototype functionality

**Design Constraints**

Constraints are necessary in order to enable a finite solution to be produced. Examples:

- the outcomes of a module must be distinct;

- two modules may *not* produce the same outcome: doing so would produce a redundancy in the teaching structure which needs to be resolved[15];
- all the module prerequisites contained in the knowledge base are satisfied by outcomes of other modules in the base[16].
- *nil* prerequisites for modules are allowed; however, they still require basic graduate knowledge, such as maths, physics, etc (the tacit knowledge);
- *cyclic dependencies* between any two modules are disallowed (e.g. if module A has a prerequisite satisfied by module B, module B must not have a prerequisite satisfied only by module A). Should this situation occur, the offending modules must be reassessed with regards to their prerequisites / outcomes[17];

Further constraints may be added in the developing the expert system, e.g.:

- maximum number of year **n** modules: may conflict with the concept of a Conversion Course and limit the flexibility of the expert system;
- maximum number of modules per Semester. The present prototype does not obey this constraint. In real life, subjects tend to be equally distributed in all Semesters;
- balanced number of modules in each Semester: see previous.

**The Expert System Conceptual Model**

Once the decision to build the prototype as a production rule-based expert system has been taken, the main elements of the system may be further specified.

The knowledge base should contain *facts* and *rules* referring to the prerequisites and outcomes of modules of the subjects offered in the University[18]. The facts are either 'fixed' (such as the modules information) or run-time asserted (e.g. the user's answers to the expert systems' questions). The inference engine must be chosen to match previous requirements and design decisions. The user interface, preferably graphical and integratable with currently and commonly used operating systems and enabling technology infrastructure (e.g. Internet), will preferably be implemented in the same language as the inference engine.

A Unified Modelling Language (UML, [24]) model of the expert system is presented in **Figure 2**. In this figure, the user-expert system interaction occurs through the user interface, which sends the problem conditions (answers to questions) to the work area that holds all the temporary data. The inference engine uses the knowledge base for the rules and fixed facts, and the work area for the dynamic facts (asserted at run-time). The knowledge base contains the fixed facts and the rules.



[15] this reveals a benefit of this system, namely identification of overlapping teaching efforts within different subjects

[16] this constraint does not include 'trivial' prerequisites, i.e. basic knowledge that a graduate is expected to have. This constrain may be relaxed if accepted that students may satisfy some prerequisites by enrolments outside the teaching institution .

[17] this should also be disallowed in real life, since there is no way for a student to enrol in either of the modules, *unless* he/she has previous skills covering the prerequisites of one of the offending modules.

[18] Supplementary information is also provided, such as semester, credit points, parent subject, etc. More information may be introduced later on as necessary (e.g. module convenor).

Finally, the solution is delivered to the user interface. The classes shown in the figure will be further specified in the Implementation section.

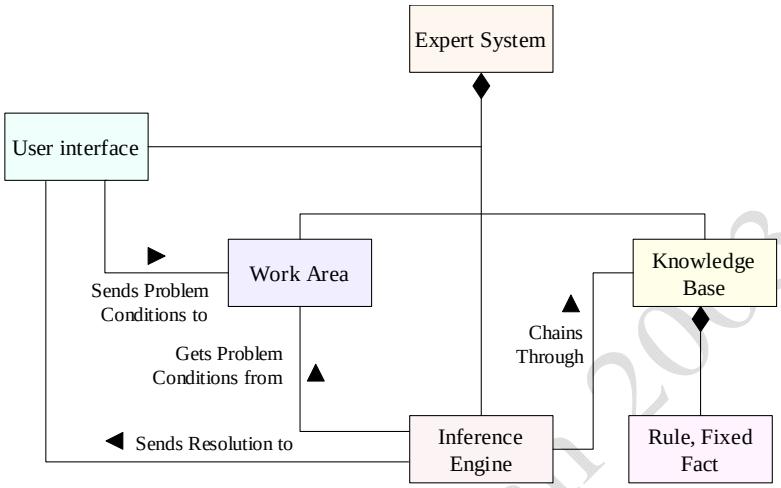



**Figure 2.** Object diagram of the expert system (based on [8])

**The Knowledge Base**

The knowledge base may be modelled using UML. In this representation, rules may be represented as objects, displaying *attributes* and *behaviour* and may be represented as shown in **Figure 3**.

| Start::InitialQuery1 |
| --- |
| ifPart = " job type Database_Admin "<br>thenPart = " ask for Query Optimisation Knowledge & assert the user answer " |

**Figure 3.** UML rule representation

The expressiveness of **Figure 3** may be improved by employing UML extension mechanisms, such as presented in **Figure 4**. In this version, each rule may also represent the rules that call- and that are called by the rule in question.

Such custom representations (which in fact create new modelling languages[19]) may be useful but they have to be based on a consistent and fully explained *metamodel*, unambiguously describing the structure of the modelling language that the specialised representation is using. Otherwise, such representations may impede, rather than facilitate information sharing.

[19] one may rightfully argue that new concepts added to the basic UML set will change its meaning, in effect creating a new modelling language

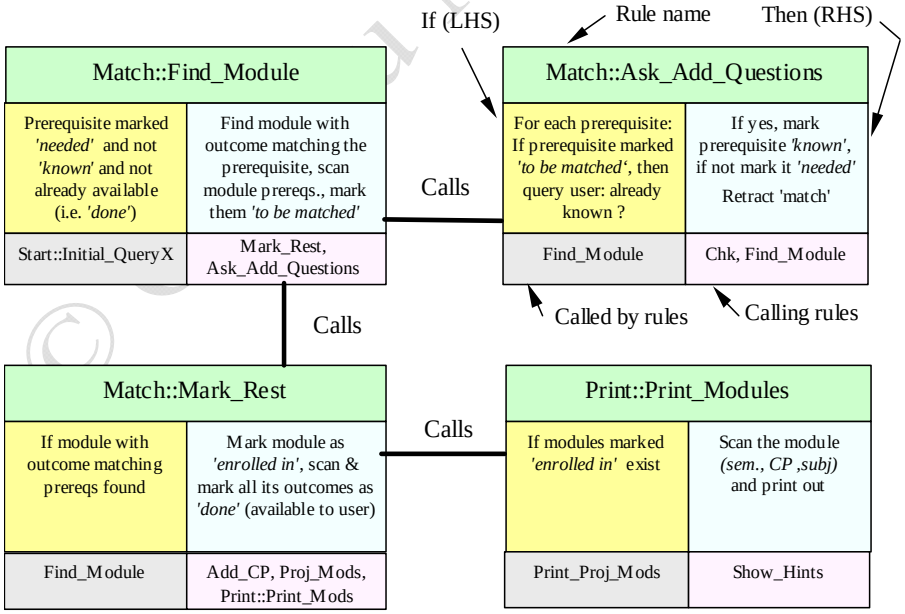
**Implementation**

Several decisions had to be made in the implementation of the expert system. the virtual machine hierarchy described in [10] provides a good guide in establishing the options. Several web-enabled expert system shells have been considered for the specific problem domain. While the shell may assist in the effort of knowledge representation, it has to be matched to the task [14]. Most shells impose a particular production rule formalism, chaining and structure[20] to the rule set.

Considering the size of the prototype, the resources available and the problem domain, the choice has been an expert system shell written in Java, which emulates the CLIPS [11] language, together with a simple graphical user interface. The Java implementation of CLIPS is called JESS - The Java Expert System Shell [9]. This provides the user with the power of the CLIPS language and the flexibility of the Java cross-platform concept. Most of the data structures provided by Java are available in JESS, including the AWT (Abstract Window Toolkit), which enables the user to construct graphical user interfaces to the JESS (CLIPS) engine. Jconsult [23] has been used as an off-the-shelf basic JESS graphical user interface.

The inference engine is indirectly provided CLIPS. The CLIPS language uses forward-chaining and the RETE fast pattern-matching algorithm [7]. CLIPS uses *lists* to process information, very similar to the LISP language.

In view of the implementation decision, **Figure 2** may now be further explained. Thus, the Work Area is the Java applet, the user interface is the applet's window, the inference engine is provided by the Java CLIPS implementation and the knowledge base resides in CLIPS file(s).




[20] or lack thereof - refer e.g. EMYCIN [25].

**Figure 4.** Possible customised rule representation

**The Knowledge Base Implementation**

JESS, similar to CLIPS (and LISP), uses the *list* construct to hold the data. The facts may be asserted manually (i.e. hardcoded in the knowledge base) or at run-time. They are implemented as lists containing one or more other lists. Example:

```
(attribute (type job)(value "Database Administrator))
```

*Templates* are similar to classes and contain models for the *facts*. For this particular system, three types of templates have been used: *goal* (a special type of fact, only used to start the expert system), *attribute* (a general purpose fact consisting of a *type* and a *value*) and *module* (a fact type describing the module information). Example:

```
(deftemplate module ; the module information template
   (slot name)   ; this is the equivalent of a class
   (slot CP (type INTEGER))
   (slot parent_subject)
   (slot semester (type INTEGER))
   (multislot prerequisites)
   (multislot outcomes)
); end deftemplate
```

A *slot* declares a field within the template - actually, a list in itself. A *multislot* declares a *multifield*, i.e. a list with more than two members. For example:

```
(module     ;a particular module - the equivalent of an object
   (name Query_Optimisation_Evaluation)
   (CP 2)
   (parent_subject Database_Management_Systems)
   (semester 1)
   (prerequisites SQL_Data_Definition_Language Data_Storage)
   (outcomes Query_Optimisation Relational_Operators)
); end module
```

This is a *module* object example where the *prerequisites* and *outcomes* list both have more than two members. This provides a very convenient way to accommodate a variable number of members within a list, without any special requirements.

The rules in the JESS environment take the following form: *if-part => then-part*. An example rule (taken from the actual program) would look like this:

```
(defrule RuleDB1a
   (attribute (type job)(value "Database
  Administrator"|"Database Designer"))
  =>
   (printout t "Databases Knowledge." crlf crlf
      "Do you have Database Architectures knowledge
  ?|explanatory|This
```

```
      type of job requires Database Architectures
  knowledge|yes|no|end")
    (assert (attribute (type Database_Architectures) (value
  (readline))))
); end RuleDB1a
```

An extensive coverage of the development environment and the user interface is beyond the scope of this document.

### Testing / Verification: Running a Consultation

The program may be run either as a standalone Java application (as shown in **Figure** 5 - used mainly for development) or as an applet embedded within a web page.

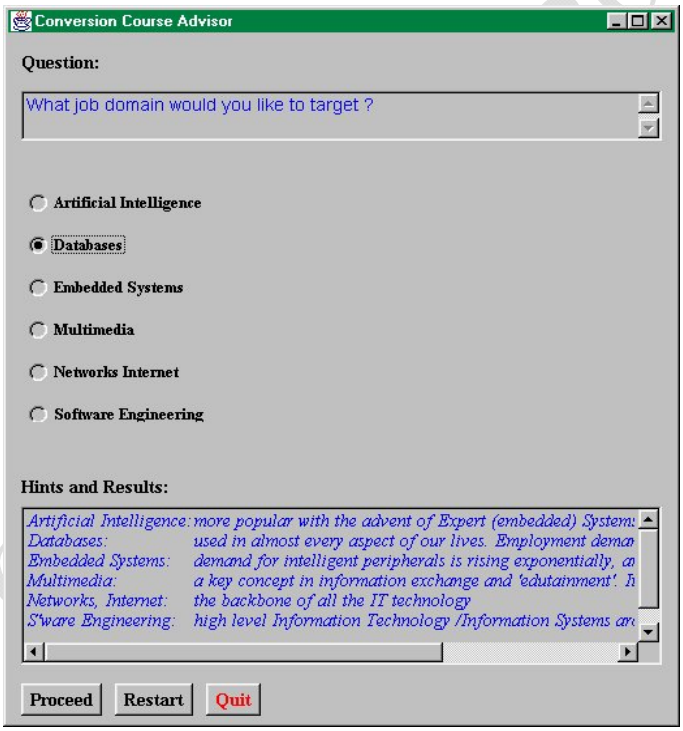



**Figure 5.** The expert system as a Java application

The system will initially require a target domain and specific employment opportunity (within the chosen domain), which will create the initial set of modules needed. The system will then query the user on previous knowledge usable to satisfy the initial set of modules' prerequisites. If the required knowledge is not present, the system will seek an appropriate module whose outcomes satisfy the prerequisites in question.

The majority of modules in the knowledge base have non-nil prerequisites. The system will seek to satisfy all the prerequisites of a module before enrolling the user in that module. This process will occur recursively - i.e. a module with **n** prerequisites may require **(n-k)** modules (where **k** represents outcomes provided by the user) with outcomes that satisfy those prerequisites. These **(n-k)** module(s) may also have **p** prerequisites that need to be satisfied, and so on. Every time a new prerequisite is discovered, firstly the user is queried whether he/she has the knowledge to satisfy it. If not, a suitable outcome of another module is searched to satisfy the prerequisite. Although this would seem to create a larger pool of prerequisites each time, in reality there are many prerequisites being satisfied by one same module, and the a knowledge base constraint states that there can be *no* unsatisfied prerequisites (in real world, meaning that the University may cater for all the needs of a postgraduate enrolled in a Conversion Course).

At the end of the consultation, the expert system will produce an output containing: Stated (existing) knowledge, Necessary modules with Parent Subject, Semester, Credit Points, Project modules (the Conversion Course must include a 40CP Project) and Total Credit Points necessary for the course[21]. Boundary values are as follows: needed CP < 70 - existing knowledge may be overstated; 70 < CP < 105 - Ok.; 105 < CP < 150 - existing knowledge may have been understated; CP> 150: the envisaged occupation / skills may be unsuitable for that person (too little knowledge).

A sample run of the expert system for the job of *'Artificial Intelligence Researcher'* (with prior knowledge) has produced the output shown in **Figure 7**.

### Deployment

**NB** at this stage there is no mechanism to ensure a *balanced* distribution of the modules within the Semesters[22]. **Figure 6** shows the current method of deployment of the prototype[23], which was deemed appropriate for the restricted initial scope of the problem domain.

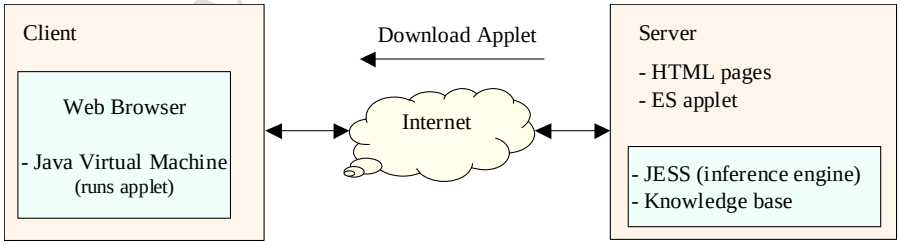



**Figure 6** The Web-based expert system.

[21] In a customised Course, the number of total credit points depends on the student's prior knowledge.

[22] The algorithm for that kind of function involves careful subject planning and further knowledge elicitation.

[23] At the time of writing, the expert system prototype is available for testing on: http://www.cit.gu.edu.au/~noran.

**Maintenance / modification.**

Deployment is not the end of the spiral development paradigm. In fact, another spiral comprising periodic maintenance and updating will be necessary to keep the knowledge base current. All additions / modifications must preserve the initial knowledge base constraints[24] (unless e.g. University policies change, etc) and be properly validated [1]. Querying the reasoning of the expert system is also available to the knowledge engineer.

## Further Work on the Prototype and Beyond

Knowledge acquisition proved to be the major bottleneck in this case study. Thus, the knowledge elicitation techniques will need to be improved so as to shorten the turn-around time from changes in the real world to their reflection in the knowledge base. The questionnaire should be available on-line so the experts can access it regardless of their location. The process of questionnaire assessment and input in the knowledge base may be (partially) automated. Also, some interview techniques have been proposed that allow automation - e.g. [20]. Low-level automated knowledge acquisition may also be derived from CLIPS[25].

Substantial improvements may be made in the user interface[26]. A further improvement would be to implement an algorithm to evenly distribute the modules by the semester they are offered in (if possible, or else extend the study period) and to implement a mechanism to limit the number of modules per semester.

Another bottleneck in the existing prototype is the fact that the user's browser must download the expert system applet, which then runs in its own sandbox on the host machine. A more efficient way would involve data / input processing on the server side, using servlets. The two main possibilities with servlets are implying user - program interaction occurring either via HTML pages containing e.g. FORM requests, or through applet / servlet interaction. Thus, the applet is much smaller in size[27].

Although the presented prototype is reusable and scalable to a certain degree, a fully featured expert system may have different needs in terms of the set of development and knowledge acquisition tools[28].



[24] A mechanism to check for unsatisfied prerequisites for modules per total is included.

[25] new rules are built as the program learns new knowledge [11]

[26] the newer versions of JESS make it easier to construct a graphical user interface due to built-in Java reflection.

[27] since all it provides is a user interface to send user input and receive and display servlet output

[28] refer e.g. [3] for a survey of knowledge acquisition tools

```
********** Consultation Output Started **********

Job targeted: Artificial Intelligence Researcher .
-------------------------------
Prior (existing) Knowledge stated:
-------------------------------
Java_API .
Information_Systems_Concepts .
Entity_Relationship_Model .
Inference_Nets .
Artificial_Intelligence_Trends .
Knowledge_Representation_Principles .
==================================================
The following Modules are needed:
-------------------------------
Semester I:
-----------
Programming_I_2, subject Programming_II, Sem. 1, 5 CP.
Advanced_Information_Systems_Development, subject Introduction_to_Information_Systems_Development,
Sem. 1, 5 CP.
Programming_Language_Implementation_1, subject Programming_Language_Implementation, Sem. 1, 5 CP.
Programming_Language_Implementation_2, subject Programming_Language_Implementation, Sem. 1, 5 CP.
Introduction_to_Artificial_Intelligence_1, subject Introduction_to_Artificial_Intelligence, Sem. 1, 5 CP.
Programming_I_1, subject Programming_II, Sem. 1, 5 CP.
Natural_Language_Processing_Basics, subject Natural_Language_Processing, Sem.1, 5 CP.
Natural_Language_Processing_2, subject Natural_Language_Processing, Sem.1, 5 CP.
------------
Semester II:
------------
Database_Management_Systems_2, subject Advanced_Topics_in_Database_Management_Systems, Sem. 2, 5 CP.
Database_Basics, subject Data_Modelling, Sem. 2, 3 CP.
Intelligent_Decision_Support_Systems, subject Intelligent_Decision_Support_Systems, Sem. 2, 5 CP.
Intelligent_Decision_Support_Systems, subject Intelligent_Decision_Support_Systems, Sem. 2, 5 CP.
Expert_Systems_Advanced, subject Expert_Systems, Sem. 2, 3 CP.
Artificial_Intelligence_in_the_Future, subject Current_Issues_in_Artificial_Intelligence, Sem. 2, 5 CP.
Expert_Systems_Design, subject Expert_Systems, Sem. 2, 4 CP.
Expert_Systems_Basics, subject Expert_Systems, Sem. 2, 3 CP.
----------------------------------------
The following Project Modules are needed:
----------------------------------------
Artificial_Intelligence_Project_Part_1, subject Artificial_Intelligence_Project, Sem. 1, 20 CP.
Artificial_Intelligence_Project_Part_2, subject Artificial_Intelligence_Project, Sem. 3, 20 CP.
==================================================
Total Credit Points = 113 .
**************************************
WARNING: you seem to require over (the required) 100 Credit Points.
----------
-> It is possible that you haven't stated ALL your existing knowledge.
-> You may also examine various courses of study by ASSUMING you had some of the knowledge.
---------
ADVICE: *Restart* and answer 'yes' to all existing / assumed knowledge.

********** End of Consultation Output  **********
```

**Figure 7**. Expert system output

## Conclusions

This paper has presented the application of an expert system to a non-trivial problem domain that involves producing a customised study program for acquiring a desired set of skills. The design and implementation decisions have been highlighted and justified, and sample run results have been provided. The development and testing of the prototype have shown that the concept of a knowledge-based advisory expert system is feasible, provided that a set of essential guidelines are followed, and due attention is paid to the knowledge acquisition process design and implementation.

Owing to its design and similar to other rule-based expert systems, (notably EMYCIN [25]), the expert system prototype described in this paper may be reused by (1) establishing that the new problem domain suits the type of design decisions taken for the current system (e.g. pattern matching, forward chaining, Web enabling, etc) and (2) replacing the knowledge base with one specific to the new problem domain.



## References


1. Benbasat, I. and J.S. Dhaliwal, *A Framework for the validation of knowledge acquisition.* Knowledge Acquisition, 1989. **1**(2): p. 215-233.
2. Bergadano, F., A. Giordana, and L. Saitta, *Automated vs. Manual Knowledge Acquisition: A Comparison in Real Domain*, in *Knowledge Acquisition for Knowledge-based Systems*, H. Motoda, et al., Editors. 1991, IOS Press: Amsterdam, The Netherlands.
3. Boose, J.H., *A survey of knowledge acquisition techniques and tools.* Knowledge Acquisition, 1986. **1**(1): p. 3-37.
4. Boose, J.H. and B.R. Gaines, eds. *Knowledge Acquisition Tools for Expert Systems*. 1988, Academic Press.
5. Collins, H.M., G. R.H., and R.C. Draper, *Where's the expertise ? Expert systems as a medium of knowledge transfer*, in *Expert Systems 85*, M. Merry, Editor. 1985, Cambridge University Press.
6. Diaper, D., *An Organizational Context for Expert System Design*, in *Expert Systems: Human Issues*, D. Berry and A. Hart, Editors. 1990, Chapman and Hall: London. p. 214-236.
7. Forgy, C., *RETE: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem.* Artificial Intelligence, 1985. **19**: p. 17-37.
8. Fowler, M. and K. Scott, *UML Distilled*. 2nd ed. 1999, Reading, MA: Addison-Wesley.
9. Friedman-Hill, E., *JESS - The Java Expert System Shell*. 1998, Distributed Computing Systems, Sandia National Laboratories: Livermore, CA.
10. Gaines, B.R. and M.L.G. Shaw, *Foundations of Knowledge Acquisition*, in *Knowledge Acquisition for Knowledge-based Systems*, H. Motoda, et al., Editors. 1991, IOS Press: Amsterdam, The Netherlands.
11. Giarratano, J., *CLIPS User's Guide*. Vol. 1: Rules. 1992: NASA.
12. Giarratano, J. and G. Riley, *Expert Systems: Principles and Programming*. 1998, Boston, MA: PWS Publishing Company.
13. ISO_JTC1/SC7, *ISO/IS 12207: Software Engineering - Life cycle Processes*. 2000.
14. Jackson, P., *Introduction to Expert Systems.* 3rd ed. 1999, Harlow, England: Addison-Wesley.
15. Jagodzinski, P., S. Holmes, and I. Dennis, *User-Acceptance of a Knowledge-Based System for the Management of Child Abuse Cases*, in *Expert Systems: Human Issues*, D. Berry and A. Hart, Editors. 1990, Chapman and Hall: London.

16. Kline, P.J. and S.B. Dolins, *Designing Expert Systems*. 1989, New York: John Wiley & Sons.
17. Lucas, P. and L. Van der Gaag, *Principles of Expert Systems*. 1991, Wokingham, England: Addison-Wesley.
18. Marcus, S., ed. *Automating Knowledge Acquisition for Expert Systems*. 1988, Kluwer Academic Publishers: Boston.
19. Minsky, M., *A framework for representing knowledge*, in *Phychology of Computer Vision*, P.H. Winston, Editor. 1975, McGraw-Hill: New York.
20. Mizoguchi, R., K. Matsuda, and N. Yasushiro, *ISAK: Interview System for Acquiring Design Knowledge*, in *Knowledge Acquisition for Knowledge-based Systems*, H. Motoda, et al., Editors. 1991, IOS Press: Amsterdam, The Netherlands.
21. Newel, A. and H.A. Simon, *Human Problem Solving*. 1972, Englewood Cliffs, NJ: Prentice-Hall.
22. Prerau, D.S., *Developing and Managing Expert Systems*. 1990, Reading, MA: Addison-Wesley.
23. Reichherzer, T., *JConsult v1.3*. 2000, Institute of Human and Machine Cognition, University of West Florida.
24. Rumbaugh, J., I. Jacobson, and G. Booch, *The Unified Modelling Language Reference Manual*. 1999, Reading, MA: Addison-Wesley.
25. van Melle, W., et al., *The EMYCIN Manual*. 1981, Computer Science Department, Stanford University.