YUCSA: A CLIPS EXPERT DATABASE SYSTEM TO MONITOR ACADEMIC PERFORMANCE

Anestis A. Toptsis, Frankie Ho[1], Milton Leindekar[2], Debra Low Foon, and Mike Carbonaro

Dept. of Computer Science and Mathematics
York University, Atkinson College
Toronto, Ontario M3J 1P3, Canada
Phone: (416) 736-5232
Fax: (416) 736-5773
Email: anestis@cs.yorku.ca

Abstract. This paper presents YUCSA (York University CLIPS Student Administrator), an expert database system implemented in CLIPS, for the monitoring of the academic performance of undergraduate students at York University[3]. The expert system component of the system has been already implemented for two major Departments and it is under testing and enhancement for more Departments. Also, more elaborate user interfaces are under development. We describe the design and implementation of the system and also discuss the problems encountered, as well as our immediate future plans. The system has excellent maintainability and it is very efficient (assessment of one student takes less than one minute).

---

[1]Also with Goodfellow Consultants, Inc., Toronto, Canada.
[2]Also with Royal Bank of Canada, Division of Management Information Systems, Toronto, Canada.
[3]York University is one of the largest in Canada, having a current enrollment of about 50,000 students.

INTRODUCTION

Student enrolment and degree requirement satisfaction checks are time consuming tasks for the University registration staff. Moreover, subjective judgements of the registration personnel may cause further conflicts in degree audits. On the other hand the rules posed for graduation and registration are numerous, frequently changing, and many of them quite complex. In the absence of an expert system, at least one human expert is necessary to assess a student's academic record for graduation and registration requirements satisfaction (as an example, a rule is "a student can only enroll in a fourth year course if he/she has completed at least 12 courses from which at least 4 in the third year level, and has obtained a grade of C or better in courses C1 and C2". Moreover, several prerequisites usually apply for any upper level course that a student wishes to enroll). As a result, a computerized academic performance monitoring package is always desirable for obvious reasons (speed, minimization of number of errors caused during the assessment process, ability for mass production of reports to be sent to students, etc).

Past experience has taught us that employment of "conventional" tools (such as operating system utilities, relational database management systems, and programming in a 3GL) is inadequate for the implementation of such a system (Gale 1990). The main drawback of such conventional tools is that the resulting system is very difficult to be maintained due to the constant changing of University curriculum, registration, and graduation rules. Therefore, even if all the rules are captured correctly during the system's development (which is not a typical case), it is very likely that the resulting product will be almost obsolete even by the time of its release. This is because some of the rules would have been already updated and/or deleted, while new rules would have been added. On the other hand, after a fairly short time development effort (less than one academic year, including problem setup, learning the details of the University rules, design, implementation, and limited testing) we have a *working* expert system which performs the above discussed task, *correctly*.

The system is designed to,

1. Replace manual effort and time spent in checking course enrolment pre-requisites and restrictions satisfaction;

2. Eliminate subjective judgements of the registration personnel;

3. Quickly determine the achievements of a student and give advice on the outstanding course works to complete his/her program.

The system is designed to perform various needs for different users. A registration personnel will use it to determine whether a student is allowed to enrol into a particular course or to determine whether a student fulfils all his/her requirements of the program. On the other hand, a student uses it to check the progress of his/her achievements and get advice from the system the outstanding course works he/she has to complete for graduation. Other staff may use it to update the student's record etc.

DESIGN AND IMPLEMENTATION

In order to understand the design concerns of the system, it is better to discuss the activities a student may face in his/her life in the University. Those activities are illustrated in Figure 1. Although the diagram is tailored for York University, it is generally true for any other institution if some rules has been changed to reflect different flow in the diagram. A new student registered into York will be evaluated for whether he/she has already completed some advanced standing courses from other institution, and if so, some advanced credits will be granted. Then, the student will take courses up to a maximum number of course that is allowable to take simultaneously. After the school term, he/she may pass or fail the courses. Credits will be given to the passing courses, and he/she will repeat those courses he/she fails. When a student fails too many courses or when he/she meets the debarment conditions, he/she will be debarred from York. Of course, a student has an option to repeat the same course once, in order to upgrade his/her marks even though he/she has passed that course. When a student accumulates enough credits and satisfies all degree requirements, he/she will graduate with a degree. At any

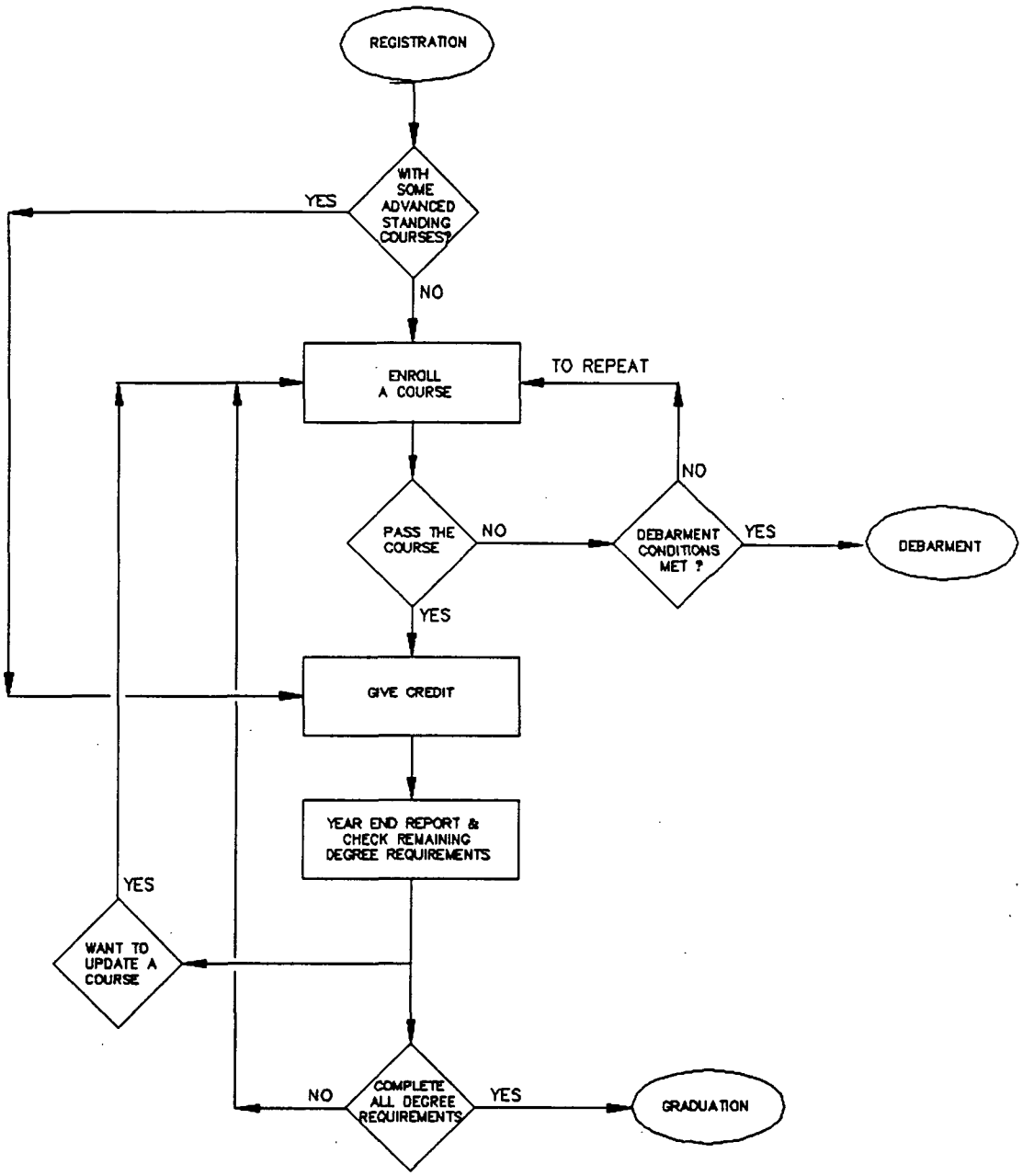

FIG 1: STUDENT LIFE CYCLE DIAGRAM

time, upon his/her discretion, a student can withdraw from the program which he/she is enroling.

The flow within the diagram is directed by the rule or a group of rules in a "box" of the diagram. Examples of those rules in the boxes are, the rules which determine whether a student is allowed to enrol into a particular course; what is the passing grade of a course (passing grade of a major course is not the same as that of a minor course); whether a student meets the debarment conditions, and whether a student satisfies all requirements for a degree; etc, etc. To illustrate this, a general rule, in the box "Enroled in a course", which governs the acceptance of a student to enrol a course is outlined below,

```
(defrule course-enrolment-accepted
   (enroling-student ?enroling-student)
   (enroling-course ?enroling-course)
   (or (permission ?enroling-student ?enroling-course)
      (and (prerequisites-satisfied ?enroling-student
               ?enroling-course)
         (course-restrictions-satisfied ?enroling-student
               ?enroling-course)))
   (student-registration ?enroling-student)
   (enroling-course-offered ?enroling-course)
   (not (enroled-in ?enroling-student ?enroling-course $?))
   (not (suspension ?enroling-student $?))
   (not (debarment ?enroling-student $?))
   (course-type ?enroling-course ? ?course-type)
   (enroled-in-fact-remark ?remark)
   (not (more-than-three-repeated-courses ?enroling-student)
=>
   (assert (enroled-in ?enroling-student ?enroling-course
      ?course-type ?remark))
   (assert (clear-old-enrolment-facts))
   (assert (update-student-records)))
```

This rule is translated in simple English as,

IF a person (wants);
to enrol into a course;
he/she has either permission from the chairman; or
he/she satisfies all pre-requisites of the course; plus
he/she fulfils all additional restrictions of the course;
he/she is registered in the college;
the course is offered;
he/she is not taking the course currently;
he/she is not under suspension;
he/she is not debarred;
he/she has not repeated more than three courses;
THEN
assert the fact that he/she is enroled;
assert the fact to trigger other rules to remove the
temporary facts;
assert the fact to trigger other rules to update his/her
records.

The "Generic Rule" concept is used so that the similar rules can be consolidated into a single rule and this rule will be triggered by different facts. Such arrangement can avoid to hard code many similar rules and to render the maintenance easier. An example of this concept is the pre-requisites list check for enroling a course. Different course has different pre-requisites and the number of pre-requisites varies from zero to many. Hard coding method is to code for each offered course a rule such as,

(defrule pre-requisites-satisfied-course-xyz

```
    (passed pre-requisite1)
    (passed pre-requisite2)
     .....
    (passed pre-requisiteN)
=>
    (assert (pre-requisites-list-satisfied)))
```

To set the rules for all offered courses is time and main memory consuming. The generic rule method is to set up two rules and to let two rules call each other until some conditions happen. Rule 1 extracts from the facts one pre-requisite and increases the counter until no more pre-requisite. If it is no more pre-requisite, the rule will insert a fact that states "the pre-requisites list check is satisfied". Rule 2 checks whether the student has passed the pre-requisite, if so, it triggers Rule 1 again for next pre-requisite.

Another concern is the *modular programming* of the system. As indicated in the Life Cycle Diagram, each "box" of the diagram represents a procedure similar to that of a procedural programming language. CLIPS is a non-procedural programming language which does not render to partition the rules easily. Using the *small problem concept* is one of the alternatives, but it requires intensive I/O tasks. The system uses the "calling fact" concept which injects into the first pattern of each rule of a "procedure" a calling fact. And, this fact will be asserted into the fact base when this procedure is called and it will be removed right after the procedure is finished. The "calling fact" concept may require more time for matching the patterns of the rules, but it is justified as it cuts down the I/O requirements and it also provides a safety feature to avoid triggering other rules accidentally.

Portability is also a concern of the system design. The "generic rule" concept discussed above is a kind of the portability. Another important concept to improve the portability of the system is the "Degree requirements achievement" concept. The idea of this concept is to decouple the degree requirements into a group of small facts of which a small fact represents a simple degree requirement. The simple facts are stored in a fact base file. Whenever a student finishes a credit or fulfils a degree requirement, a "degree requirements achievement" fact will be

asserted into the fact base and the system will match it with the decoupled degree requirements facts mentioned above. When all degree requirements facts are achieved, the student has completed the program, otherwise, the outstanding requirements are obtainable easily. For different disciplines, it is only required to analyze and to decouple the degree requirements of that particular discipline and store them into the fact base file.

The facts using in the system can be classified into three categories,

1. The permanent facts are those relatively stable facts which will only be changed with the curriculum revision. Examples of such facts are the course-type, the degree requirements facts;
2. The dynamic facts are those facts changing with the progress of the student in his/her life cycle such as the degree requirements achievement, the enroled in course facts of a student at particular time;
3. The control facts are those facts asserted into the fact base to trigger some other rules or to perform some tasks temporary. Those facts for procedural concepts are one of those.

During execution, the permanent and dynamic facts are stored in the fact base and it may be down loaded to the file folders if necessarily. While the control facts will be removed from the fact base as soon as when this fact is no longer required.

COMPUTING ENVIRONMENT/COST

The development of the system was done using two Intel 80286 based microcomputers with conventional configurations. About 300 man hours were spent for the development of the existing system.

FUTURE ENHANCEMENTS

As it stands, the number of students that can be present in the database is (severely) limited by the available main memory. In addition, efficient retrieval of student records requires indexing that must be built during run time, in case that the entire database is loaded in main memory during each run. A solution to both of these problems (space limitations and expense of retrieval) is to use a conventional database management system to store the students' records, and interface the DBMS with CLIPS. Then, all data pertaining to a *particular* student can be retrieved from the database and brought into main memory upon any individual request. Using the indexing features available in most DBMSs, the retrieval would be fast; also, assuming that a small number of students' records (less than 10, for example) can be held in main memory, there would be no space limitations. Since the existing system works with a small number of students present in main memory, once such an interface is built, the system would require no modifications, but it can act as the intelligent module of a larger system.

REFERENCES

William Gale, Nick Chan, and Anestis Toptsis (1990), "CASA: An Expert Database System for Student Guidance", *Proc. Fourth UIC/CCC Partnership Program Conference*, Chicago, May 1990.