NADA: NMSU Advising & Degree Audit System

J. E. Mendez-Mateos, G. Gupta, A. I. Karshmer, J. P. Brown
Department of Computer Science
New Mexico State University
Las Cruces, NM 88011

Abstract

Checking the fulfillment of graduation requirements for a degree from an American university is a complex problem. This paper suggests a logic programming approach to develop a software system called NADA for analyzing a student's course grade transcript to verify that all the conditions for graduation are met. If the student is not eligible for graduation, the NADA system then acts as his/her advisor and suggests possible courses that the student can take in order to graduate. In this paper, our logic programming based approach for solving the degree audit and advising problem is presented together with an Internet accessible user interface developed in Java. The system is fully functional and has been specialized for use by students in the College of Arts and Sciences at New Mexico State University (NMSU). This work shows the ease with which logic programming systems can be crafted for complex problems.

1 Introduction

Verifying the fulfillment of degree requirements (referred to as "degree check" or "degree audit" in this paper) at most American universities is a complex problem. The complexity arises because of the multiple requirements that a student has to meet and the various ways in which each of these requirements can be satisfied. These requirements are not only large in number, with myriad of ways to satisfy each, they can also be quite complex.

This paper presents a system, based on a logic programming approach, for advising and degree auditing specifically for the College of Arts and Sciences at New Mexico State University (NMSU). The system is general enough that other colleges and universities can also make use of it.

The system automatically accesses and analyzes a student's transcript (or gradesheet) and checks that all requirements for graduation have been satisfied. If the system cannot prove that all conditions have been met, it advises which courses the student can take in order to fulfill the requirements and obtain the desired degree. The systems permits the student to try our various what-if scenarios (e.g., what if I take this particular course? Or, what if I change my major subject of study).

2 The Degree Audit Problem

Verifying whether a student can graduate or not is a difficult task because the university, the college, and each department impose various requirements. Some of those requirements can be satisfied in different ways and often influence each other and prevent them from being satisfied independently. Thus, there is a combinatorial explosion in the number of different routes toward graduation. The problem is exacerbated by the fact that University publishes a new course catalog every year. In each new catalog the requirements get slightly modified. A student is given the option of choosing a catalog from among the several University catalogs: the chosen catalog can be the one published in the year the student joined the University, or the one published in any of the 5 subsequent years. This chosen catalog will be used to determine the student's program of study. It is also possible that a student may be able to graduate from the university given one major subject of study, but cannot graduate given another major subject. (In the rest of the paper we will refer to a student's major subject of study as his/her major and a university course catalog as just a catalog.) The chosen major (and minor, if applicable) and catalog are used as a basis for the degree audit. The choice of the catalog by a student is determined by the requirements it contains, e.g., some catalogs may have fewer requirements than others. Thus, determination of whether a student can graduate entails analysis of hypothetical scenarios using different university catalogs and majors/minors. After choosing a major (and a minor, if applicable) and a catalog, the student is ready to check if the courses he/she has already taken are enough to fulfill all the requirements for graduation stated in that catalog for that major/minor. The general problem of degree auditing is, thus, quite hard.

This paper describes the NADA system (NMSU Advising and Degree Audit system), a logic programming based system, developed as part of the Master's thesis of the first author [9], for performing on-line degree audits automatically. The NADA system is being deployed at NMSU for use by students.

2.1 Degree Audit

In order for a student to obtain an undergraduate degree, he/she has to satisfy the graduation, university, college, and department requirements. In this section, a summary of degree requirements imposed by New Mexico State University, the student's college and department, will be presented as described in the University's undergraduate course catalog [12]. This will help the reader appreciate the complexity of the degree-audit problem.

2.1.1 Graduation Requirements

To obtain a degree at NMSU, each student must complete at least 128 credit hours including no fewer than 55 credit hours numbered 300 or above and must also satisfy the following requirements [12]:

- A student must have an overall grade point average of C or better when all the courses taken at NMSU are considered.
- The student's last 30 credits towards a degree must be taken at NMSU (most American Universities allow transfer of courses from other institutions).
- The student is required to show proficiency in English in all of his/her classes.

2.1.2 University Requirements

University requirements are common to all students at the university and include the fulfillment of *basic academic skills* and *general education course requirements*. Basic academic skills in English and Mathematics have to be demonstrated by all NMSU students before enrolling in any upper-division courses (courses numbered 300 or higher). For this reason, and because a degree audit with no upper-division courses would be irrelevant, the system does not check for basic skills. The university General Education course requirement consists of 38 credits distributed in three parts as follows [12]:

- Part I. Developing Critical Thinking and modes of Expression: this includes courses taken from the following subcategories
  A. English Composition (7 Credits).
  B. Mathematics (3 Credits).
  C. Information Retrieval.
  D. Critical Thinking/Analysis (6 Credits).

- Part II. Establishing a Common Background: this includes courses taken from the following subcategories
  A. Historical Perspectives (3 Credits).
  B. Basic Natural Sciences (4 Credits).

C. Human Thought and Behavior (3 Credits).
D. Social Analysis (3 Credits).
E. Literature or Fine Arts (3 Credits).

- Part III. Viewing a Wider World (6 Credits).

For each of the subcategories above (e.g., Historical Perspectives), there is a list of courses from which requisite number of courses must be chosen to satisfy the credit hour requirement of that subcategory. This list of courses can be quite long, and can change from one catalog to another. There also exists an alternative for meeting the General Education requirements: a student taking 9 or more credits in a specific subject area will have met the General Education requirement for that subject area.

2.1.3 College Requirements

In the NADA system proposed in this paper, the only college requirements considered are the ones from the College of Arts and Sciences. NMSU has five other colleges, which are not recognized by the system: College of Agriculture and Home Economics, College of Business Administration and Economics, College of Education, College of Health and Social Services and College of Engineering (the construction of the NADA system has been funded by the Dean of Arts and Sciences, hence the restriction). A student can meet the College requirements by [12]:

- earning a grade of C of higher in all courses taken to fulfill college requirements.
- passing three Liberal Studies [1] courses, each selected from any three different Liberal Studies areas (Communication Studies, English Literature, Fine Arts, History, Languages, or Philosophy).
- passing three Science courses [1], each selected from any three distinct science departments (Astronomy, Biology, Chemistry and Biochemistry, Computer Science, Geological Science, Mathematical Sciences, Physics or Physical Geography).
- taking three Social Science courses [1], each selected from any three areas of Social Science (Anthropology, Economics, Social Geography, Government, Psychology, or Sociology).

Knowledge of a second language is another college requirement [12] that can be fulfilled in several ways, depending on the language(s) that the student already speaks.

2.1.4 Departmental Requirements

The system proposed in this paper concentrates on the College of Arts and Sciences. Only the departments that belong to this college are considered by the system. These departments are: Aerospace Studies, Art, Astronomy, Biology, Chemistry and Biochemistry, Communication Studies, Computer Science, Criminal Justice, Economics and International Business, English, Geography, Geological Sciences, Government, History, Journalism and Mass Communications, Languages and Linguistics, Mathematical Sciences, Military Science, Music, Philosophy, Physics, Psychology, Sociology and Anthropology, and Theatre Arts.

Most of the departments offer majors with requirements that consist of having no fewer than 20 credits of upper-division courses in a specific field, and sometimes outside the field. Usually, most departments have the course requirements of their majors listed under three main categories: Departmental, Non-departmental and Electives. For a student to satisfy these requirements, he/she has to get a grade of C or above in each course taken, for that course to count towards a major.

There are numerous other rules regarding how transferred courses are to be treated, how one course can be substituted for another, etc. All these rules cannot be described here due to lack of space, however, they make the task of degree audit quite complex.

2.2 Current Solution & Previous Work

At present, the College of Arts and Sciences at NMSU relies upon one human expert who analyzes students' transcripts and determines whether the students can graduate. Students are entitled to only two degree requirement checks during their studies at NMSU. One of them can occur after the completion of 85 credits, and the other during the final semester [1]. An automated system, such as NADA, will allow the students to perform as many degree audits as they wish. If a transcript being audited does not fulfill all requirements imposed by the university, the college, and the department, the results of the degree audit will advise the student regarding which courses are still needed for him/her to graduate. The current solution depends heavily on the human expert, without whom the College would not be able to perform degree requirement checks. This is reason enough to automate the task.

In 1983-84, a system called DARS (Degree Audit Reporting System) was developed by Jack Southard of Miami University in Oxford. This system is currently being used by a number of schools throughout the United States. It appears to be very flexible and easy to use by the students. One big problem that schools working with this system have to worry about is the time it takes to adapt the system to the specific requirements of the institution. This process of adaptation is usually considered a project in itself.

Another system has been developed by the University of Florida system. This system uses SQL, a relational database language. This system is quite flexible, and is accessible through the Internet. The system is specifically designed for universities in the University of Florida system and is not meant as a general system. Moreover, it requires a team of about five programmers to keep it up and running. The amount of resources needed in this approach are thus prohibitive. One of the goals of our project was to devise a system which is cost-effective from the point of view of development and maintenance.

An earlier effort was made at NMSU itself as part of a course project [3], but it had to be abandoned due to its experimental nature and due to lack of financial backing. The NADA system can thus be thought of as a second generation system, and builds upon the lessons that were learned in the first system. The NADA system achieves a greater degree of automation (transcripts are automatically accessed from the University database) as well as has highly sophisticated and accessible WEB-based user interface (see Section 5.2).

There are several other products out in the market as well, but in a poll that we did of the Universities that are using these products we found that most of them were highly dissatisfied with the automated system they were using. From examining different approaches taken by other universities, it is obvious that the problem is adaptability. A system that adapts fast to degree requirements is desperately needed.

2.3 Objectives and Contribution of this Paper

A thorough study and analysis of the degree audit problem and the relationships between various requirements imposed made the design and construction of the NADA system possible. The NADA system presented in this paper will automatically and securely verify degree requirements and advise the students. It will give the students the opportunity to perform as many degree audits as they desire. These audits can be performed any time during the students tenure at the University. It allows them to analyze various "What if?" scenarios. That is, the students is in the position to experiment with the system, choosing from different catalogs (i.e., different requirements) to decide which one is more convenient, trying out different majors and/or minors, and asking the question: What would happen if I take these courses? By using the system, the College of Arts and Sciences would no longer be dependent on the human expert, and because the system will be accessible through the Internet, the students will be able to use it from anywhere and at anytime.

3 Proposed Solution

This paper proposes a modular approach for solving the problem of developing a Degree Audit system that is easy to use and access as well as adaptable to the institution's requirements. There are two major modules proposed: the Engine or Degree Audit module, and the User interface. The Engine module handles the problem of verification of degree requirements, as well as keeping all the necessary student and requirement information to produce accurate results. The User Interface module takes care of the interaction between the user and the Engine module. It sends the user's input to the Engine module and returns the results from the Engine to the user.

3.1 Engine or Degree Audit Module

The problem of degree audit and advising consists of checking to see if the student fulfills all the University, college, and departmental requirements. This problem appears to be straightforward on surface, however, in reality and after careful analysis of the problem, it can be concluded that it is actually quite a difficult problem. In the worst case, it reduces to an uninformed search problem, also called blind search [11]. Essentially, it is an NP-hard problem.

The search space in the degree audit problem is determined by a set of requirements that have to be satisfied. Its complexity can be fully appreciated only when one notices that most requirements can be fulfilled in myriad of different ways and that, in the worst case, the system would have to traverse the entire search space to verify that the requirements have been met. This implies that the problem is of a highly non-deterministic nature. Thus, a tool or programming language that supports non-determinism (e.g., backtracking) is required. To successfully develop an automatic system that gives advice to students and verifies degree requirements, a tool that handles non-determinism has to be utilized. The tool proposed in this paper is PROLOG. In fact, after studying this problem closely, we can say with confidence that any effort to solve the degree audit and advising problem that is based on using a traditional imperative language will be deficient in one way or the other.

The requirements can be organized in a top-down hierarchy where the upper levels have the most general requirements and lower levels have more specific requirements. This notion can be represented as a tree structure, where the root represents 'degree requirements' and the leaves represent satisfied and unsatisfied requirements. The satisfied and unsatisfied requirements are obtained by accessing the student's transcript and checking fulfillment of very specific requirements.

3.2 User Interface Module

The interface and Internet-accessibility features that the system needs in order to be user-friendly can be developed as modules separate from the advising and degree auditing part of the system. For the user interface and Internet-connected module of the system, an object-oriented approach is suggested because of the great influence that object orientation has had on user interface design. To be more specific, the recommended object-oriented language to build the user interface and network connectivity of the system is Java [6].

4 Formal Specification and Requirements Of The NADA System

In this section, a more detailed definition of the system will be presented. A description of topics such as system functions, the role that the system will play in the university, and the planning of the system development will also be discussed.

4.1 Input and Output

There are three distinct forms of input and only one form of output. The three input forms are: one that comes from the end user (the student, also called user), one that comes from the system administrator, and the one that comes from the university databases.

The user input consists of a student ID, personal identification number (PIN), choosing a major and/or minor to base the verification on, and some classes that the system can use to answer the student's question: What if I take these courses?

The student ID is the student's Social Security Number, the same as in the other university systems. The PIN feature is provided only as a security precaution that hopefully will allow a student to gain access only to his/her own information.

Choosing a minor and/or major depends on what the student wants the system to verify. Usually the student will want to choose his/her actual major and/or minor, and only choose others to experiment.

The input needed from the administrator consists of degree requirements and course specification information found in the Undergraduate Catalog. It is recommended that this information be upgraded each time a new catalog is published. The format in which the course specification information has to be input by the administrator will be specified later. The rules for graduation are hard-wired in the program. If the graduation rules change significantly, then the program has to be modified. However, typically the rules for graduation do not change from one catalog to another, only the set of courses that can be used to satisfy a particular requirement change from one year to another. Thus, maintaining the program will not be a difficult task.

The university database information that is used as input essentially consists of the students' transcript records (or gradesheet). It is used to check if the courses a student has taken are adequate for fulfilling degree requirements of the major chosen.

The university database contains a record of each student previously or currently enrolled in the university (actually the only students the system currently considers are the ones from the College of Arts and Sciences) that shows information about the courses taken, transferred and waived, grades earned, etc.

The output of the system is directed exclusively to the student and consists of advising information in the form of a list of courses that the student should take in order to fulfill the requirements that are still outstanding for obtaining the desired degree.

4.2 Software Functions

Figure 1 shows the different parts of the system and the function each one is responsible for:

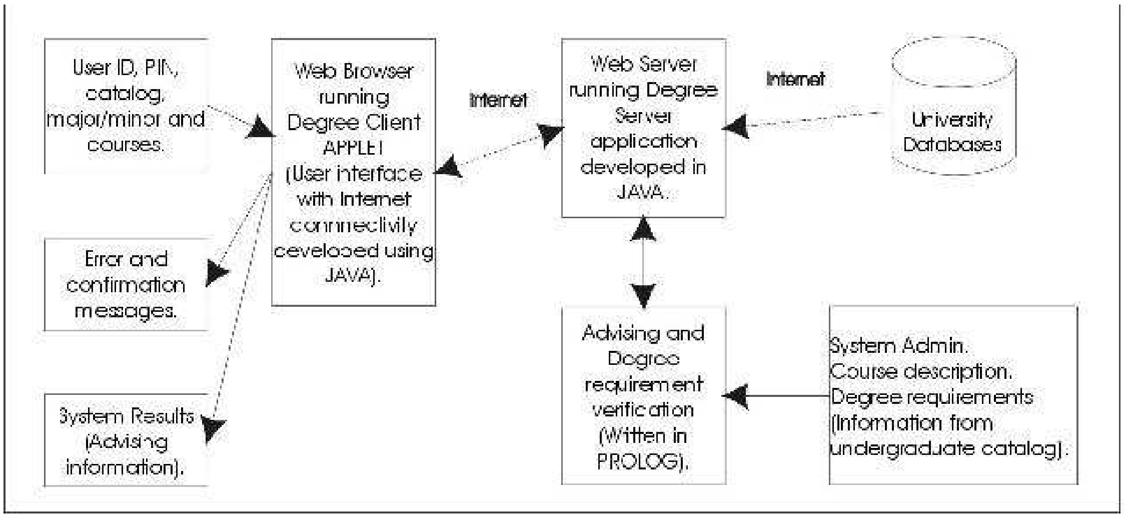

Figure 1: Data flow diagram.

5 System Overview

This section presents a data flow diagram of the whole system and a detailed explanation of the inner workings of each of the two main modules (see figure 2).

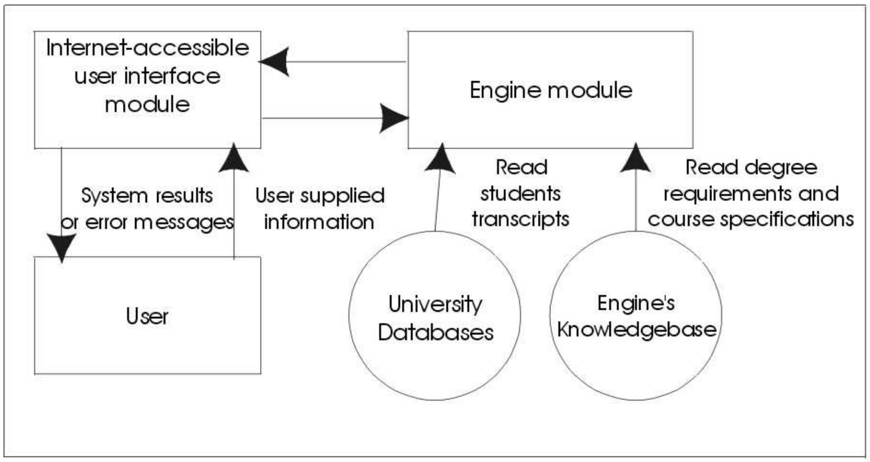

Figure 2: System data flow diagram.

5.1 Designing the Engine Module

The Engine module, which was developed in PROLOG, follows a logic-programming paradigm. It is made of three major parts that together check degree requirements (as shown in figure 3). These three parts check the university, college and departmental requirements respectively. Each part is organized in a top-down manner, and checks the requirements that its name indicates.

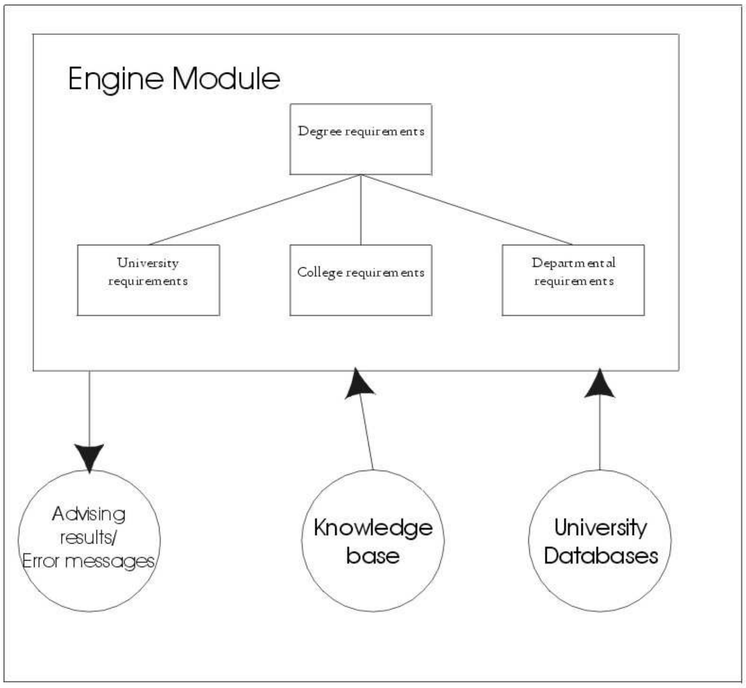

Figure 3: Engine module data flow diagram.

5.1.1 The Knowledgebase

The knowledge base consists of course information as found in the university undergraduate catalog [12]. It is organized alphabetically and it is assumed that the system administrator will provide it and update it each time a new catalog is published.

The format in which each course in the catalog is presented in the knowledge base is as follows:

```
class(department,course-num,suffix,name,
   [min-credits, max-credits,max-credits-with-repetition],
   count-credits-for-degree,count-credits-for-GPA,offered).
```

Where the variables are: department, course-num, suffix, name, min-credits, max-credits, max-credits-with-repetition, count-credits-for-degree, count-credits-for-GPA and offered:

Department: Is the variable that indicates the abbreviation of the department (e.g., Computer Science, CS etc.).

Course-num: As its name indicates, it represents the number of the course (e.g., CS 101).

Suffix: Indicates the type of requirement that a course can fulfill. The different suffixes are: G for general courses, N for developmental courses, H for honor courses, and Z for normal courses.

Name: Is the name of the course (e.g., CS 110 Computer Literacy).

Min-credits: Is the minimum number of credits that a course can be taken for.

Max-credits: Is the maximum number of credits that a course can be taken for.

Max-credits-with-repetition: Is the maximum number of cumulative credits that a student can have when taking a course multiple times.

Count-credits-for-degree: Indicates whether the credits of a course should be counted for the degree.

Count-credits-for-GPA: Indicates whether the credits of a course should be counted in the GPA (Grade Point Average) calculation.

Offered: Is a variable that indicates if the course is currently being offered or not.

A portion of the knowledge base that shows courses from the Biology, Chemistry and Computer Science departments is shown below for illustration:

```
class(biol,110,g,"Contemporary Problems in Biology",[4,4,4],y,y,of).
class(biol,111,g,"Natural History of Life",[3,3,3],y,y,of).
class(biol,111,l,"Natural History of Life Laboratory",[1,1,1],y,y,of).
class(che,101,z,"Introduction to Chemical Engineering",[3,3,3],y,y,of).
class(che,101,h,"Introduction to Chemical Engineering-Honors",[3,3,3],y,y,of).
class(cs,110,g,"Computer Literacy",[3,3,3],y,y,of).
class(cs,137,z,"Pascal Programming",[3,3,3],y,y,of).
class(cs,167,z,"C Programming",[3,3,3],y,y,of).
```

5.1.2 Information from the University Databases

Information from the students' transcripts is kept in the university databases and then transformed into facts that the Engine module can understand. These facts represent the courses that a student has taken, transferred, or waived. The structure of the facts is as follows:

```
class_taken(Student_SSN,Dept,Class_number,
   Suffix, Semester,Year,Credits,Grade,Location)
```

Where the variables are: `Students_SSN`, `Dept`, `Class_number`, `Suffix`, `Semester`, `Grade` and `Location`:

Student_SSN: Is the Social Security Number of the student, which is used as an ID in the university.

Dept: Is the department that offers the course.

Class_number: Indicates the number of a course (e.g., CS 110).

Suffix: Indicates the type of requirement that a course can fulfill. The different suffixes are: G for general courses, N for developmental courses, H for honor courses, and Z for normal courses.

Semester: Is the semester in which the course was taken. Only fall, spring, summer1, and summer2 are accepted.

Year: Indicates the year in which the course was taken (e.g., 1997).

Credits: Indicates the number of credits for which a course is taken. Using the default value 'cr' tells the system to check the knowledge base for the maximum number of credits that the course can be taken; otherwise, a value can be specified.

Grade: Is the grade that the student got after taking the course. This grade goes from four, which indicates a grade of A, to one, which indicates a grade of D.

Location: Is the location of the institution where the course was taken. This is very useful with transferred courses. The default for courses taken is the main campus of New Mexico State University (NMSU).

Below we show some of the facts from the transcript of student ID=999396532 who is currently pursuing a CS major:

```
   class_taken(999396532,engl,111,g,spring,89,cr,3,nmsu).
class_taken(999396532,engl,218,g,fall,90,3,3,nmsu).
class_taken(999396532,math,180,z,fall,88,cr,2,nmsu).
class_taken(999396532,math,185,z,fall,88,cr,3,nmsu).
class_taken(999396532,math,279,z,fall,90,3,3,nmsu).
class_taken(999396532,cs,110,g,fall,88,cr,3,nmsu).
class_taken(999396532,stat,210,g,fall,89,cr,3,nmsu).
```

5.1.3 The Engine Parts

To construct the engine module and its parts, it is important to distinguish first the structure of the four most basic types of requirements that exist in the university:

1. In the first type of requirement, a certain number of courses from a supplied list have to be taken and passed with a certain grade by the student. It does not matter which courses are chosen from the list, as long as the requisite number of courses have been selected from the specified list (see figure 4).

2. In the second type of requirement, it is necessary that the student takes and passes all of the courses specified in a list. The difference between this requirement and the previous one is that in this one, all courses in the list are needed to meet the requirement, not just a subset of them (see figure 5).

3. In the third type of requirement, a specific number of sub-requirements in the lower level are to be satisfied. These sub-requirements are supplied as a list. It is not important which particular sub-requirements are chosen to be fulfilled as long as the requisite number of them are selected from the supplied list (see figure 6).

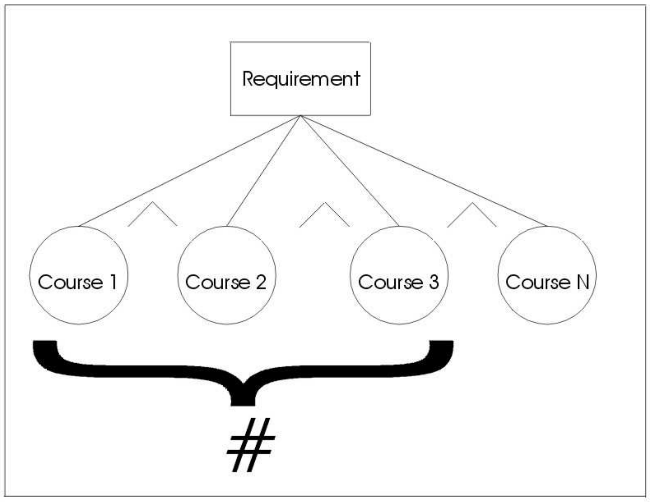

Figure 4: Basic type of requirement # 1.

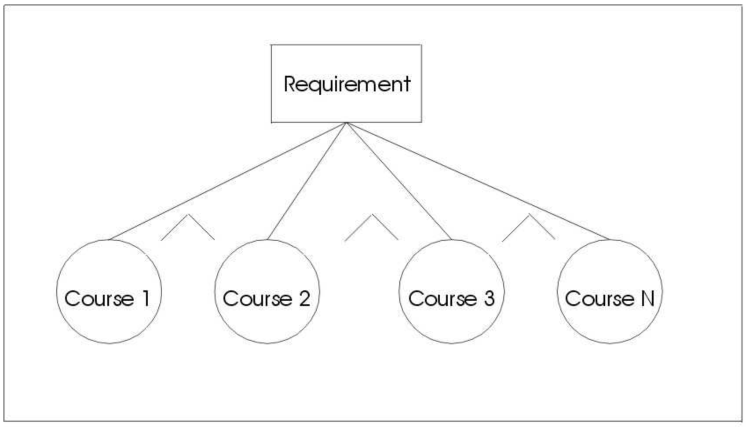

Figure 5: Basic type of requirement # 2.

4. In the fourth and last type of requirement, at least one of the lower level requirements has to be satisfied. The difference between this requirement and the last one is that in this one, only one of the sub-requirements has to be met, not a subset of them (see figure 7).

The three major parts of the Engine module make use of three utility predicates. These predicates are very general and their use corresponds exactly to the four basic structures of requirements previously shown:

choose_from_classes: This predicate corresponds to the first and fourth cases. In the first case (see figure 4) a student is given a list of courses from which he/she has to take a specific number of them in order to fulfill the requirement. For example:
`choose_from_classes(Student,2,1,[[engl,111,g],[engl,111,h],[spcd,111,g ]],Chosen).`
indicates that the student has to take at least two courses from the supplied list and obtain a grade of at least D. An example that corresponds to fourth case (see figure 7) is:

```
ct_computer_science(Student) :-
        ( choose_from_classes(Student,1,1,[[age,250,g], [bcs,110,g],
                [cs,110,g], [et,152,z], [me,160,z], [ce,151,z],
                [aen,151,z], [gen,151,z], [ie,151,z], [hon,151,z]],
                Chosen) )
        ;
```

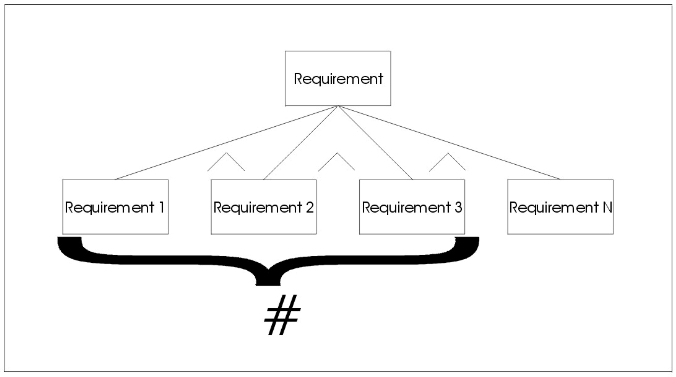

Figure 6: Basic type of requirement # 3.

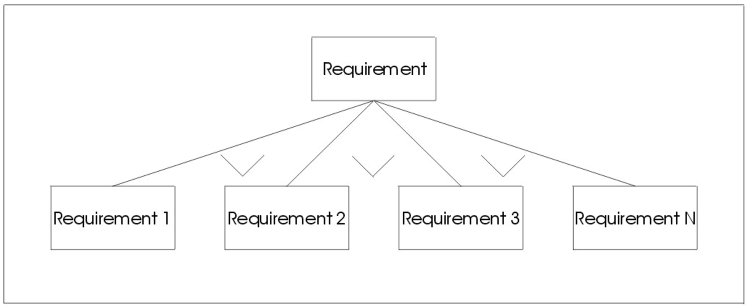

Figure 7: Basic type of requirement # 4.

```
( choose_from_classes(Student,2,1,[[ee,162,z], [ee,162,l]],
        Chosen) ), !.
```

This indicates that the student has to take either one course, from the first list of courses or two courses from the second list in order to fulfill the `ct_computer_science` requirement. The courses that fulfill the requirement have to be passed with a grade of at least a D. If either requirement is met, the variable 'chosen' will have a list of the courses that satisfy the `ct_computer_science requirement`.

class_taken: This predicate corresponds to the second basic structure of requirements (see figure 5). It works by checking if all courses specified by the class_taken predicates have been taken by the student:

```
alternative_to_general_mathematics(Student) :-
     class_taken(Student,math,Course1,Suffix1,Sem1,Year1,Cr1,Grade1,Where1),
     class_taken(Student,math,Course2,Suffix2,Sem2,Year2,Cr2,Grade2,Where2),
     class_taken(Student,math,Course3,Suffix3,Sem3,Year3,Cr3,Grade3,Where3),
     grade1 >= 1, Course1 >= 115, Grade2 >=1, Course2 >= 115,
     grade3 >= 1, Course3 >= 185,
     check_difference([Course1,Course2,Course3]),
     !.
```

In this case the student has to take three different math courses and pass them all with a grade of at least a D to fulfill the alternative_to_general_mathematics requirement.

choose_from_predicates: This predicate corresponds to the third basic structure of requirements presented earlier in this chapter (see figure 6). It works by checking the fulfillment of a number of requirements from a given list:

```
critical_thinking(Student) :-
        choose_from_predicates(Student,2,[
                ct_oral_communication(Student),
                ct_philosophy(Student),
                ct_computer_science(Student),
                ct_statistics(Student)],
        chosen),
        !.
```

In this example, the student has to fulfill at least two requirements from the supplied list in order to fulfill the `critical_thinking` requirement. The variable 'chosen' will have a list of the requirements that have been met and that satisfy the `critical_thinking` requirements.

In practice, checking a requirement may imply checking a combination of the four basic types of requirements. The Engine module works by grouping requirements into three groups: university, college, and departmental requirements (as shown in figure 3), and checking them for fulfillment one by one. If any of the three groups of requirements is not met, the corresponding parts of the engine will return a list of courses needed to fulfill those requirements; list which serves as advise to the student. The following example shows how the system checks university, college, and departmental requirements for a student pursuing a Bachelor of Science in Chemistry:

```
   bachelor_of_science_chemistry(Student,Recommendations,Language) :-
   bachelor_of_science_chemistry(Student),
   university_requirements(Student,R1),
   college_core_requirements(Student,R2,Language),
   append(R1,R2,Recommendations),
   !.
bachelor_of_science_chemistry(Student,Recommendations,Language) :-
   university_requirements(Student,R1),
   college_core_requirements(Student,R2,Language),
   append(R1,R2,Temp),
   R_Chem = ['Bachelor of Science in Chemistry',
   ['Nondepartamental Requirements'],
   ['Take all these',
      [math,191,z], [math,192,z], [math,291,z], [phys,215,z],
      [phys,216,z], [phys,215,l], [phys,216,l] ],
   ['Choose 1 from these',
      [math,391,z], [math,392,z], [cs,171,z], [stat,371,z] ],
   ['Choose 1 from these',
      [phys,315,z], [phys,390,z] ],
   ['Departamental Requirements'],
   ['Choose the first or second pair',
      [chem,115,z], [chem,116,z], [chem,111,z], [chem,112,z] ],
   ['Take these',
      [chem,242,z], [chem,313,z], [chem,314,z], [chem,315,z],
      [chem,356,z], [chem,371,z], [chem,421,z], [chem,433,z],
      [chem,434,z], [chem,435,z], [chem,443,z], [chem,452,z],
      [chem,456,z] ]
   ],
   append(Temp,R_Chem,Recommendations).
```

This example has two predicates; both of them check if the university and college requirements have been met. The first one gets used when all requirements, including departmental requirements, are met. The second one is used when there exists at least one departmental requirement that has not been met. In this case, a list of different courses that can be used to satisfy the unfulfilled requirements is given to the user as advice.

5.2 Designing the User Interface Module

The User Interface module consists of two main programs developed in Java; one that runs in a web browser on the client machine (the applet), and the other that runs on a web server (the server application) where the Engine module resides (see figure 1).

The applet and the server application are connected through the Internet. The applet collects information from the user, transmits it to the server application, and waits for an answer. The server application gets the user information sent from the applet, requests student transcript information from the university databases and transforms it into facts understandable to the Engine module. The Engine module processes the information and writes the result to a file. The server application reads the file and transfers it to the applet along with the student transcript information. In case an error occurs throughout this process, the server application notifies the applet so the user knows the nature of the problem. The applet receives the student's information and gives him/her the opportunity to specify to the system the courses that he/she is planning to take or is currently taking. The student can also specify to the system any waived or transferred course. After the student enters all his/her additional information, the applet sends it to the server application so that it can be submitted to the Engine module again. This process can continue as long as the student wants to keep trying out different alternatives (e.g., hypothetically taking different courses).

The following figure shows the different classes used to developed the applet and server application:

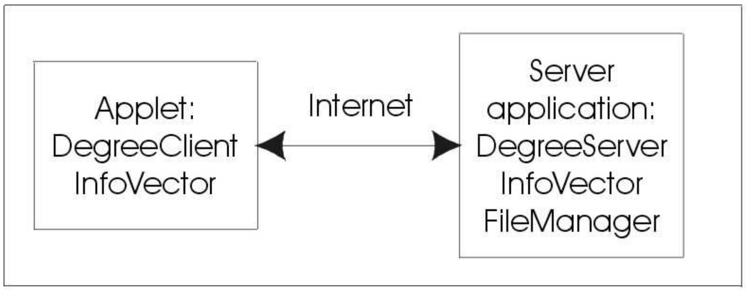

Figure 8: User Interface module data flow diagram.

The applet uses two classes: one to handle the user interface (see figure 9) and the other to handle information:

DegreeClient: This class takes care of network connectivity to the server application and the user interface of the system.

InfoVector: This class handles the data structures involved when transferring information between the applet and the server application.

The server application uses three classes:

DegreeServer: This class handles network connectivity with the applet, access to the university databases to get students' transcripts and making calls to the Engine module.

FileManager: This class takes care of reading the Engine module result file into the server application and translating transcript information into facts that the Engine module can understand.

InfoVector: This class handles the same data structures that the InfoVector class from the applet uses.

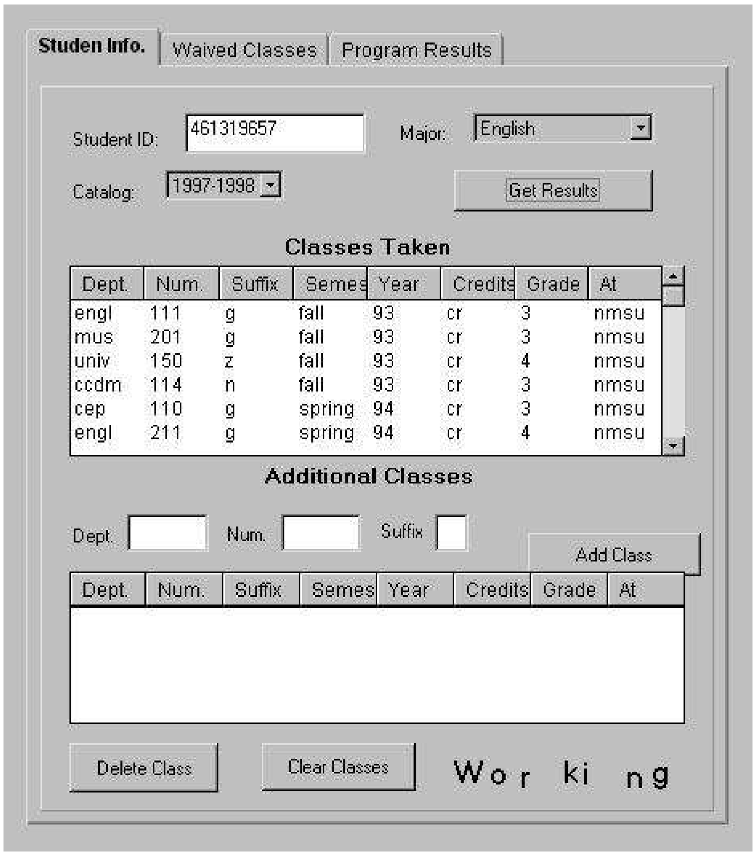

Figure 9: User Interface.

6 Conclusions

This paper presented the NADA system for advising and degree verification and suggested tools such as PROLOG and Java for its construction. It can be concluded from this experience that the assumptions and recommendations made when planning this system were in fact correct. This follows, in particular, from the construction of a working prototype that can be easily maintained and updated by the system administrator, and more importantly, any student enrolled in the College of Arts and Sciences can use it. This system highlights the ease with which logic programming technology can be used for solving decision problems that involve a large amount of data and manipulation of complex constraints, and the adequacy of using object-oriented languages for user interface design.

Work on the NADA system is still continuing. The eventual goal is to finally put the system in production use at NMSU, so that students have access to an automated degree audit and advising system. Currently, it is being evaluated by the human experts who are responsible for degree audits at NMSU. Plans are underway for the system to be deployed for use by students at NMSU.

Acknowledgements

We are grateful to Dr. René Casillas, Dean of the College of Arts & Sciences for providing financial support, and to Cuca Saenz and Juanita Herrera, the degree audit human experts, for their assistance in evaluating the NADA system. Thanks also to David Rocks and the Computing Center Staff for providing and helping with access to the University databases.

References

[1] Advising Manual (1996). College of arts and sciences new mexico state university.

[2] Barfield, L. (1993). *The User Interface, Concepts and Design.* Addison-Wesley Publishing Company.

[3] Bassetti, F. et al. (1996) . A Logic Programming System for Checking Graduation Requirements Technical Report, New Mexico State University, Jan. 1996.

[4] Booch, G. (1994). *Object-Oriented Analysis and Design with Applications.* The Benjamin/Cummings Publishing Company, Inc.

[5] Clocksin, W. F. and Mellish, C. S. (1994). *Programming in Prolog*, page 239. Springer-Verlag.

[6] Edith Au, D. M. and the Pencom Web Works (1996). *Java Programming Basics.* MIS: Press.

[7] Graham, I. (1993). *Object Oriented Methods*, page 31. Addison-Wesley Publishing Company.

[8] Khoshafian, S. and Abnous, R. (1995). *Object Orientation*, page 388. John Wiley & Sons.

[9] Mendez-Mateos, J. E. (1998). *NADA: NMSU Advising and Degree Audit System.* Master thesis. Lab for Logic and Databases, Department of Computer Science, NMSU. 1998.

[10] McGraw, K. L. (1992). *Designing and Evaluating User Interface for Knowledge-based Systems.* Ellis Horwood.

[11] Rusell, S. and Norvig, P. (1995). *Artificial Intelligence A Modern Approach.* Prentice Hall.

[12] Undergraduate Catalog (1997). New mexico state university undergraduate catalog. NMSU.