Proceedings of the 28th Annual Hawaii International Conference on System Sciences - 1995

PACE: A Planning Advisor on Curriculum and Enrollment

Himawan Gunadhi, Kwang-Hui Lim & Wee-Yong Yeong

Department of Decision Sciences, National University of Singapore

Abstract
We present a framework for an intelligent advisory system for college students that combines object-oriented and knowledge-based paradigms. Such an approach results in representational efficiency and flexibility, improved performance, and ease of software development and maintenance. A decision model is presented for course advising based on students' need to know "what to do" and "how to do it". It consists of profiling a student's strengths and weaknesses, generating a personal curriculum customized to each person's needs, and producing a schedule for the courses chosen. A system is being developed based on this framework for advising students studying for the Bachelor of Business Administration (BBA) degree at the National University of Singapore.

1. Introduction

University student advising is a heavy responsibility with real consequences: guiding students in choosing and scheduling courses to maximize their potential requires considerable expertise on the part of the advisor. The increasing complexity of curricula being offered in educational institutions and the high cost of academic staff time, make student advising a good candidate for expert system implementation. University advising systems are good candidates for expert system implementation. In addition, the problem exhibits characteristics favorable to an expert system approach - it is restricted to domain specific knowledge, uses voluminous data, is difficult to characterize accurately, and is constantly changing [8, pp.8]. While an expert system will not replace the need for wise and sympathetic counsel from human advisors, it will focus students more clearly on issues to consider and let them simulate different scenarios before seeking advice, thus alleviating academic staff of part of their burden.

Considerable work has been done on intelligent planning systems in general [1], [7], [9]. However, there are few documented systems designed specifically for academic advising. Goloumbic [3] developed an expert system that comprises of student information, departmental knowledge base, planning facility and user interface. In particular, his planning facility describes succinctly the requirements for an academic plan: consistency with university regulations (e.g., prerequisites), consistency with student goals, and "schedulability". More recently, Cutright et al. [2] constructed a PC-based expert system that attempts to generate a schedulable list of courses consistent with prerequisites starting from an initial assertion list.

Due to the complexity of our curriculum, an expert system with a richer knowledge content than the ones described above is needed. The Planning Advisor on Curriculum and Enrollment (PACE), simplifies the process of ensuring consistency and schedulability by combining object-oriented (OO) with knowledge-based system (KBS) methodologies. This allows us to concentrate on the quality of the advise rather than the mechanics of the system during implementation. In our approach, objects hierarchies (trees) are used to represent data and knowledge structures (e.g., courses, curriculum requirements). Localized knowledge is embedded as methods within objects, while global knowledge is stored in a rule-base. An inference engine employs forward and backward chaining on global and the relevant local rules to generate advise and to guide students in choosing and scheduling courses. Combining OO and KBS approaches has distinct advantages:

- Efficient knowledge representation – production rules are stored as methods within objects. This provides superior knowledge representation of the curriculum and other structures as compared to non-KBS approaches, where all possible combinations of events are wastefully stored or unwieldy if-then conditions are used. It is also better than a KBS approach without OO methodology, where even the basic object hierarchy has to be represented as...
production rules, slowing down execution and making the tracing of object structures difficult.

- **Flexible knowledge representation** – separate concepts are stored as separate objects and linked dynamically; e.g., it is very easy to modify the relationships between curriculum, courses and staff which are separate but closely interrelated object trees.

- **Improved performance** – searching localized knowledge relevant to a specific object instead of the entire rule-base each time greatly reduces the search space.

- **Ease of software development and maintenance.**

In Section 2, we describe the background to our work. The conceptual framework of our decision model is introduced in Section 3. The model is explained in greater depth in Section 4, and the data structures to support it discussed in Section 5. Section 6 highlights implementation issues. Finally, we outline future plans in Section 7.

### 2. Background

A new modular system is being implemented at the National University of Singapore (NUS). The system offers great flexibility and makes better use of resources, but is much more complex than its predecessor. Hence, the need for student advising has increased tremendously.

The undergraduate Bachelors in Business Administration (BBA) program was restructured in 1990 from a year-based system into a modular one based on semesters. Students take self-contained courses known as *modules*. There are three kinds of modules – essential, elective and enrichment – to ensure students get adequate breadth and depth while choosing them. Students may pursue a General Degree by completing 26 modules in 2 to 4 years or an Honors Degree by completing 35 modules in 3 to 5 years. In this paper, “module” and “course” are used interchangeably.

Courses are offered during two regular 17-week semesters each year and an 8-week summer semester. Most students complete the BBA General Degree within 3 years although it can be as short as 2 and as long as 4 years. Brighter students are offered the opportunity to pursue a 4-year Honors Degree program after completing their second year of study.

Our system differs from that of many universities because of the large number of constraints and rules involved. There are many groups of courses, such as concentration, distribution, electives, compulsories and research groups. Each group has its own set of courses to choose from and rules for choosing them; e.g., courses have to be taken in a semi-ordered sequence, i.e., Levels 1, 2, then 3; many courses have prerequisites, etc. The combination of courses available has expanded from a mere handful in 1990 to over 10 today (see Table 1). Students have to decide on course selection more often – three times a year, instead of once a year under the old system. In selecting courses, students must know which combinations are possible and be careful not to miss taking prerequisites needed later. These factors have increased demand for student advising.

<table>
<thead>
<tr>
<th>Level</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courses</td>
<td>Core(8)</td>
<td>Core(6)</td>
<td>Policy</td>
</tr>
<tr>
<td></td>
<td>English</td>
<td>Electives(2)</td>
<td>Distribution(4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Concentration(3)</td>
</tr>
<tr>
<td># modules</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td># of ways</td>
<td>2</td>
<td>156</td>
<td>478,800</td>
</tr>
<tr>
<td>Total # of combinations</td>
<td>$2^{156} \cdot 478,800 = 149$ million</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Possible Combinations of Modules

### 3. Conceptual Framework

#### 3.1 Scope of PACE

The role of a course advisor is sandwiched between that of a career counselor and of a timetabling system, as shown in Figure 1.

A career counselor helps students to decide what line of employment to pursue. A course advisor explores the best combination of courses that supports this goal, but takes into account other considerations, e.g., aptitude. It may suggest the pace and sequence for the courses, but its lowest level of detail is the semester or the year. A timetabling system prescribes in greater detail a schedule for taking these courses, specifying the time, duration, venue, and other information.

Our framework is consistent with that of O'Banion, who described advising in terms of the following steps [4,

---

Proceedings of the 28th Annual Hawaii International Conference on System Sciences (HICSS '95)
1060-3425/95 $10.00 © 1995 IEEE
1. Exploration of life goals, 2) Exploration of vocational goals, 3) Program choice, 4) Course choice, and 5) Scheduling courses. In some literature, course advising is referred to as academic advising, which "is concerned with the intellectual or cognitive components of the curriculum such as course selection, prerequisites, major cognates, requirements, and student performance and progress" [5, pp.136].

Course advising and career counseling are so closely related that many students find it difficult to separate them in an advising session [4, pp.71]. To create a usable system, the function of a course advisor in our definition is extended to act as a primitive career counselor by suggesting a set of courses from a predefined list of career opportunities.

3.2 Advise and Planning

According to Gordon [4, pp.52], a typical course advising interview has five elements: 1) Opening the interview, 2) Identifying the problem, 3) Identifying possible solutions, 4) Taking action on the solution, and 5) Summarizing the transaction. In our experience, steps (2) through (4) do not occur in such a clear-cut manner, but actually appear as a series of interrelated questions reflecting interacting subgoals. What is essential is the content of these steps, rather than the actual sequence in which they occur. Hence, our decision model deviates from this procedural approach (see Section 3.3).

Gordon cites several basic advising skills: information dissemination, referral, monitoring, decision making, counseling, mentoring and teaching. The first four are integral to our design. However, the remaining three are more difficult to implement on a machine as they involve many qualitative factors. Gordon's emphasis on good, clear communications underscores the need for a user-friendly interface in any computer system taking on the role of a course advisor.

One major result of the course advising process is a student plan. A plan is a representation of a course of action, consisting of a set of goals which can be ordered or unordered [1, pp.515]. A plan usually contains many subplans in a hierarchical structure. Planning is often difficult because of the presence of many interacting goals and subgoals. However, it is very useful and can be used to optimize a student's effort, open possibilities to alternative solutions, monitor progress, and catch errors early by comparing feedback to the plan. It is therefore a worthwhile exercise for students to undertake. There are four basic approaches to planning [1, pp.515-561]:

- **Hierarchical Planning** - generate a hierarchy of representations of a plan. The highest level representation is a simplification/abstraction of the plan, while the lowest level is a detailed plan. Refine vague parts into more detailed subplans until a sequence of problem-solving actions is clearly defined. This is a top-down approach.
- **Nonhierarchical Planning** - develop a sequence of actions that reduces goals into simpler ones. Unlike hierarchical planners, nonhierarchical planners only have one representation of the plan and so cannot distinguish between important and unimportant subgoals. Hence, they often get bogged down by unnecessary details.
- **Script-based Planning** - Find and apply a suitable skeleton plan from a pre-stored collection.
- **Opportunistic Planning** - Let "specialists" make suggestions on how a user should fill up a "blackboard" containing the plan. Each "specialist" operates synchronously, focusing on a particular kind of decision. Opportunistic planning is a bottom-up approach.

3.3 Modeling the Problem

The human process of advising students is highly subjective and variable. In order to provide consistent and usable advise, a more objective approach is required. Our two-prong approach is driven by student needs:

- Decide "WHAT TO DO" - choose a set of courses that fulfill the curriculum requirements, career goals, etc.
- Decide "HOW TO DO IT" - arrange the courses in the most desirable way.

Expert advise is used to guide the student in answering these two questions in the context of a planning horizon that the student sets. However, he/she should be warned if choices made during the planning horizon severely affect or limit future opportunities. The expert advise should maximize the student's development in areas that enhance his/her career plans and minimize time spent in the university and overlaps of course content.

The decision model is illustrated in Figure 2. Background questioning is used to profile the student's aspirations, capabilities and planning horizon. If the student's career objectives coincide with that from a predefined list, then a set of courses is recommended. The student is then guided through the process of selecting the courses he/she would like to take. Once enough courses have been selected, an initial schedule is proposed. The student is allowed to refine this schedule under expert guidance, revise the choice of courses, or modify his/her response to the background questions. This process is described in greater detail in Section 4.

Note the role of the expert system as an advisor/guide rather than as a persuader. The student takes primary responsibility for his/her career goals.
action and responsibility for deciding his destiny; the expert system provides a "blackboard" to help draw up his/her decision and furnishes advice as appropriate. The expert system's role is threefold: 1) Guide the student in creating a personal curriculum, 2) Guide the student in arranging a schedule, and 3) Ensure curriculum constraints/rules are never violated.

Our approach is a hybrid between hierarchical and opportunistic planning. Curriculum requirements are satisfied in a top-down manner driven by career choices, student abilities and other constraints. Advice is provided by "specialists" embedded as methods within objects. This occurs in an opportunistic manner depending on how students choose courses from the curriculum groups shown on the "blackboard".

3.4 Knowledge Representation and Object Structure

Course advising contains both global and local knowledge components. Global knowledge affects the choices of courses as a whole, and includes 1) Overall curriculum and course structures, 2) Semester structure (years and the semesters within each year), 3) General constraints (e.g., minimum candidature, number of courses per semester), and 4) Advice on courses to take based on career aspirations and general constraints. Local knowledge is specific to an object, e.g., that associated with choosing courses within a curriculum group or scheduling a particular semester. Components of local knowledge are: 1) Constraints on courses, e.g., prerequisites, course availability, etc., 2) Advice on suggested courses, 3) Advice on how to schedule the courses.

These knowledge components are easily represented using the OO-KBS approach. The curriculum, course and semester structures are represented by object trees containing CURRICULUM, COURSE and SCHEDULE objects respectively (see Section 4). The nodes of a tree represent various subclasses, and its leaves, the instances of objects. Object trees inherently contain knowledge in the way their nodes and branches are arranged, e.g., the arrangement of the curriculum tree reflects the actual curriculum structure of a university. Advice related to career aspirations and general constraints are stored as global rules. The rules are triggered as needed when the student attempts to choose courses. Apart from providing textual information to the user, they also add or remove items from an assertion list of courses to take and to avoid.

Local knowledge is stored as slots and methods within objects, which may be subclasses or instances of the object trees. Slots contain information specific to an object, while methods are procedures or rules related to that object. Thus, course availability is stored in a slot belonging to a COURSE object, while advice on choosing courses for a particular course group is implemented as a method within its CURRICULUM object. Both slots and methods can be inherited by the children of an object. Interacting subgoals are handled by allowing objects to communicate between themselves via messages. For example, a course object may check the status of a prerequisite by messaging the COURSE object representing the prerequisite, which then in turn triggers a method (local rule) to check if it is already taken.

3.5 Definitions

Important terms used in this paper are defined below:

- **Decision Space (D)** - the set of all possible combinations of courses that satisfies all curriculum constraints.
- **C** - a set of courses that satisfies all curriculum constraints as well as an individual student's needs. C is an instance of D for a given student.
- **CURRICULUM object (D)** - an object representing a group of courses. It includes a list of courses to choose from and the rules for choosing them.
- **Curriculum tree** - a hierarchy of CURRICULUM objects representing all the curriculum requirements of a university.
- **COURSE object** - An object representing a single course, detailing the course content, past performance and popularity, instructors, entry requirements, prerequisites, etc.
• **Course tree** – a hierarchy of COURSE objects containing all the courses offered.
• **Binding** – a CURRICULUM object is binding if all its courses have to be chosen before a usable schedule can be generated.
• **Container** – a non-binding curriculum object with courses not yet chosen.
• **Personal Curriculum** – a relaxation of C that may also include containers.
• **Planning horizon** – a semester in the future until which the student would like to plan.
• **SCHEDULE object** – An object representing one semester of study. It contains a list of courses, their status (taken, to be taken, already taken) and the grades scored if applicable.
• **Schedule tree** – a set of SCHEDULE objects stretching from the time a student enters the university till his/her planning horizon.

**4. Process Flows & Knowledge Structures**

This section describes the decision model in detail, along with its knowledge structures.

**4.1 Choosing Courses**

The curriculum in a university usually consists of a set of rules for choosing courses, along with a range of courses to choose from. The rules determine the number of courses that must be taken, prerequisites, limits to certain courses, and other constraints. For example:

**Distribution Requirement (3rd Year BBA Honors):**

**RULE:** students are expected to take four of the following list of modules of which one must be in his area of concentration.

**COURSES:** BH310 Financial Management  
          BH320 Marketing Management  
          BH330 Human Resource Management, etc...

Each curriculum requirement may have sub-requirements that must also be satisfied. This hierarchical structure is represented by the **curriculum tree** shown in Figure 3. Each node in the tree is a CURRICULUM object, with properties listed in Table 2. Production rules and constraints are stored as the ChooseCourses method in each D_i. They generate the search space D representing all possible combinations of courses such that all curriculum constraints are satisfied. Storing the **curriculum tree** instead of D provides compact and flexible knowledge representation, as discussed in Section 1.

**Figure 3. Curriculum Tree**

<table>
<thead>
<tr>
<th>SLOT</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoursesChosen</td>
<td>List of courses chosen</td>
</tr>
<tr>
<td>Satisfied</td>
<td>Boolean. True when the student has satisfactorily chosen courses for D_i and all its child-objects</td>
</tr>
<tr>
<td>MinCoursesReqd</td>
<td>Minimum requirements for this curriculum object to be satisfied</td>
</tr>
<tr>
<td>CoursesNotWanted</td>
<td>List of courses not desired by the student</td>
</tr>
<tr>
<td>Binding</td>
<td>True if this CURRICULUM object must be satisfied before a schedule can be created</td>
</tr>
</tbody>
</table>

**Table 2. Structure of a Curriculum Object**

The student has to take a subset C of courses that satisfies all curriculum requirements and is tailored to his/her individual needs. This decision is based upon the characteristics of the student and of each course. Factors pertaining to the student include career goals, areas of interest and performance (past/expected). Those pertaining to the course include prerequisites, course availability, course instructors and selectivity (how easy it is to join a class).

All the curriculum constraints must be satisfied in choosing C. However, not all courses need to be chosen before a usable schedule can be generated from the chosen courses. For example, elective courses add complexity to the decision, but sometimes do not matter to a student's overall direction. CURRICULUM objects with such courses are considered non-binding, in contrast...
to binding CURRICULUM objects for which all courses must be chosen before a meaningful schedule can be created. For instance, CURRICULUM objects for compulsory and concentration courses are binding. A CURRICULUM object is said to be satisfied 1) When all its courses are chosen, or 2) It is non-binding. Only then would its Satisfied slot contain a TRUE Boolean value.

The expert system has to guide the student in producing a personal curriculum (Table 3) in which courses for all binding CURRICULUM objects have been selected. A personal curriculum is therefore a relaxation of the set C that satisfactorily meets the student’s needs. It may contain non-binding CURRICULUM objects which have not been selected, which are called containers as they “contain” decisions deferred till a later stage. The construction of a personal curriculum is completed when all the CURRICULUM objects in the curriculum tree have been satisfied.

<table>
<thead>
<tr>
<th>Level</th>
<th>CURRICULUM Object</th>
<th>Courses in Personal Curriculum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>Level 1 Compulsory</td>
<td>BK101, BK102</td>
</tr>
<tr>
<td></td>
<td>Level 1 Optional</td>
<td>None</td>
</tr>
<tr>
<td>Level 2</td>
<td>Level 2 Compulsory</td>
<td>BK201, BK207</td>
</tr>
<tr>
<td></td>
<td>Electives (2 courses)</td>
<td>BK211, One undecided</td>
</tr>
<tr>
<td>Level 3</td>
<td>(General Degree)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Policy (Compulsory)</td>
<td>BK300</td>
</tr>
<tr>
<td></td>
<td>Distribution courses (4)</td>
<td>Finance (BK310), Strategy (BK340), OM (BK350), MIS (BK360)</td>
</tr>
<tr>
<td></td>
<td>Concentration Courses (3)</td>
<td>Ranking (BK313), Risk (BK314), Insurance (BK312)</td>
</tr>
<tr>
<td></td>
<td>Up to student (1)</td>
<td>undecided</td>
</tr>
</tbody>
</table>

Note: Containers are marked in bold

Table 3. Sample Personal Curriculum

The biggest choices for a student are probably that of a major and whether to pursue an honors or general degree program. If the choice of major subject is known, then the expert should direct the student towards a personal curriculum that enhances his/her capability in the chosen field. Otherwise, the student should be allowed to choose from a broad range of courses based on inclination and performance, but notified if at any time he/she chooses courses that will limit future opportunities, i.e., to keep as many options open for as long as possible. Students are allowed to choose by elimination by indicating courses they do not want to pursue. These are stored in the CoursesNotWanted slot of each CURRICULUM object and affect future choices if they are prerequisites to other courses. The nature of advise is also affected, because the student should be encouraged to remain open-minded and not to exclude courses out of prejudice rather than genuine lack of ability and/or interest.

Courses for the CURRICULUM objects are chosen from the course tree shown in Figure 4. Each CURRICULUM object’s CoursesChosen slot points to the relevant COURSE objects from the tree. For each CURRICULUM object, courses can only be selected from specified branches of the course tree. This natural separation between the curriculum and courses provides great flexibility. Courses are treated as the building blocks of the curriculum. Therefore, curriculum changes are dealt with as rearrangements of the curriculum tree or as modification to the rules for selecting courses within CURRICULUM objects.

4.2 Arranging Courses

Once the student has created a personal curriculum, a schedule is generated (Figure 5). The schedule consists of a set of semesters stretching over the student’s planning horizon. Each semester contains a list of courses and their status: already taken, being taken, or to be taken. Courses already taken include the grades scored by the student. Apart from COURSE objects, a semester may also have containers. An initial schedule, called a normative schedule, is generated once all binding curriculum requirements are satisfied. It can be based on a fixed number of semesters (e.g., 3 years x 2 semesters each for a General Degree), or recommended based on the student’s planning horizon, past performance and course availability.
A student may refine or transform a schedule in the following ways (see Figure 6):

- **Move** courses from one semester to another.
- **Stretch/Contract** the schedule by adding/reducing the number of years.
- **Dilute/Concentrate** the schedule by changing the number of semesters in a year (there are up to three semesters in an academic year, the third being optional).
- **Replicate** courses the student is likely to fail or may want to re-take.

Courses already taken or being taken cannot be moved. Transformations can only be made from one allowable state to another. This leads to a very elegant implementation – the system only needs to check that all constraints and prerequisites are still satisfied after each transformation in order for it to be legal. Each time the student attempts a transformation, expert advice is provided. He/she can also access a help facility if unsure of what changes to make. Apart from transforming the schedule, a student may also change the personal curriculum and the responses to the background questions. This allows the student to experiment with various scenarios to iteratively refine his/her plans.

The schedule is internally represented by a schedule tree (Figure 7). This facilitates easy transformation and hence fast on-screen response. Each leaf (instance) points to a list that contains courses from the course tree or containers from the curriculum tree. Stretching/contracting the schedule involve adding/deleting subclasses, while concentrating/diluting it involves adding/deleting instances. Course and containers are moved or replicated by manipulating the list contained in each leaf.

Courses already taken or being taken cannot be moved. Transformations can only be made from one allowable state to another. This leads to a very elegant implementation – the system only needs to check that all constraints and prerequisites are still satisfied after each transformation in order for it to be legal. Each time the student attempts a transformation, expert advice is provided. He/she can also access a help facility if unsure of what changes to make. Apart from transforming the schedule, a student may also change the personal curriculum and the responses to the background questions. This allows the student to experiment with various scenarios to iteratively refine his/her plans.

The schedule is internally represented by a schedule tree (Figure 7). This facilitates easy transformation and hence fast on-screen response. Each leaf (instance) points to a list that contains courses from the course tree or containers from the curriculum tree. Stretching/contracting the schedule involve adding/deleting subclasses, while concentrating/diluting it involves adding/deleting instances. Course and containers are moved or replicated by manipulating the list contained in each leaf.

Key data elements and the information flows between them are illustrated in Figure 8. The student database contains information specific to a student's goals and performance. The administrative database is a relational database that stores statistical information about students, courses and staff. The knowledge base contains rules and structures for the courses, curriculum and schedule trees described above. It also contains both local and global advise. The student database is a local file available only to each student, while the administrative

---

Figure 5. Schedule

A student may refine or transform a schedule in the following ways (see Figure 6):

- **Move** courses from one semester to another.
- **Stretch/Contract** the schedule by adding/reducing the number of years.
- **Dilute/Concentrate** the schedule by changing the number of semesters in a year (there are up to three semesters in an academic year, the third being optional).
- **Replicate** courses the student is likely to fail or may want to re-take.

Courses already taken or being taken cannot be moved. Transformations can only be made from one allowable state to another. This leads to a very elegant implementation – the system only needs to check that all constraints and prerequisites are still satisfied after each transformation in order for it to be legal. Each time the student attempts a transformation, expert advice is provided. He/she can also access a help facility if unsure of what changes to make. Apart from transforming the schedule, a student may also change the personal curriculum and the responses to the background questions. This allows the student to experiment with various scenarios to iteratively refine his/her plans.

The schedule is internally represented by a schedule tree (Figure 7). This facilitates easy transformation and hence fast on-screen response. Each leaf (instance) points to a list that contains courses from the course tree or containers from the curriculum tree. Stretching/contracting the schedule involve adding/deleting subclasses, while concentrating/diluting it involves adding/deleting instances. Course and containers are moved or replicated by manipulating the list contained in each leaf.

Figure 6. Transforming A Schedule

Courses already taken or being taken cannot be moved. Transformations can only be made from one allowable state to another. This leads to a very elegant implementation – the system only needs to check that all constraints and prerequisites are still satisfied after each transformation in order for it to be legal. Each time the student attempts a transformation, expert advice is provided. He/she can also access a help facility if unsure of what changes to make. Apart from transforming the schedule, a student may also change the personal curriculum and the responses to the background questions. This allows the student to experiment with various scenarios to iteratively refine his/her plans.

The schedule is internally represented by a schedule tree (Figure 7). This facilitates easy transformation and hence fast on-screen response. Each leaf (instance) points to a list that contains courses from the course tree or containers from the curriculum tree. Stretching/contracting the schedule involve adding/deleting subclasses, while concentrating/diluting it involves adding/deleting instances. Course and containers are moved or replicated by manipulating the list contained in each leaf.

Figure 7. Schedule Structure

Key data elements and the information flows between them are illustrated in Figure 8. The student database contains information specific to a student's goals and performance. The administrative database is a relational database that stores statistical information about students, courses and staff. The knowledge base contains rules and structures for the courses, curriculum and schedule trees described above. It also contains both local and global advise. The student database is a local file available only to each student, while the administrative
database and knowledge base are maintained by administrators and shared by all students.

An object-oriented system that supports hierarchical knowledge representation with inheritance of object properties and local as well as global knowledge (rules) representation is needed. Local rules can be implemented by default methods which may be overwritten/overloaded down the object hierarchy. An inference engine is required to drive the decision model, using both forward and backward chaining. Finally, a database package is required to manage information (e.g., courses, student information, etc.), and to interface with other systems. Based on these considerations, the integrated tool set shown in Table 4 was chosen.

6. Implementation

Our objective is to build a system that is, in descending priority: 1) User friendly & easy to use, 2) Adaptable to changes in curriculum, 3) Able to provide satisfactory advise and explanations, 4) Easy to maintain (program and documentation), and 5) Strong in performance - response and turnaround time.
The system will be implemented on the campus-wide PC network in the university (NUSNET), as shown in Figure 9. The knowledge base and administrative database are accessible to all users via NUSNET, but are transparent to students who can only access them through PACE. The administrative database is periodically updated by an office administrator. The knowledge-base can only be modified by a system developer familiar with the inner workings of the entire system. The student database is stored as a local file accessible only to each student.

The key issues that have to be addressed during implementation are: 1) How to acquire and incorporate "real-life" expertise, 2) What data is accessible, versus what is confidential, 3) How to ensure security of the administrative database and knowledge-base on the network, 4) How much of the reasoning process should be revealed to students by the explanation facility, and 6) How to interface to other systems in the university (online registration system, academic and student databases, automated timetabling system).

7. Future Directions

PACE is currently being coded by a team of students programmers in the university. The curriculum module should be completed by June 1994, along with an application to update the administrative database. The schedule and background questioning modules should be completed by August. We expect to complete a working model on the network by the end of 1994.

Once PACE is implemented and tested, the quality of advice can be refined to take into account special populations such as transfer, high-ability and high-at-risk students [4, ch.5]. Feedback from PACE can be used for planning courses and vacation schedules for academic staff based on anticipated demand.

We have shown the object-oriented, knowledge-based methodology to be suitable for academic student advising. The approach provides efficient and flexible knowledge representation, improved performance and smooth implementation. Many other real-world problems may benefit from applying this approach.

References


