AN OBJECT ORIENTED INTELLIGENT TUTORING SYSTEM FOR TEACHING SET THEORY

A THESIS

Submitted to the Department of Computer Engineering and Information Sciences and the Institute of Engineering and Sciences of Bilkent University in Partial Fulfillment of the Requirements for the Degree of Master of Science

By
BABEL (KERIMOĞLU) CANKOCAK
JUNE 1991

AN OBJECT ORIENTED INTELLIGENT TUTORING SYSTEM FOR TEACHING SET THEORY

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND

INFORMATION SCIENCES

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

BY

EMEL (KERİMOĞLU) CANKAT

JUNE 1991

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. H. Altay Güvenir

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. David Davenport

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Petek Aşkar

Approved for the Institute of Engineering and Sciences

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

AN OBJECT ORIENTED INTELLIGENT TUTORING SYSTEM FOR TEACHING SET THEORY

EMEL (KERİMOĞLU) CANKAT

M.S. in Computer Engineering and
Information Sciences
Supervisor: Assist. Prof. Dr. H. Altay Güvenir
June 1991

In this thesis an Intelligent Tutoring System (ITS) to teach set theory is designed and implemented using an object-oriented approach. The program is implemented on an IBM PS/2 and PC compatibles using the Turbo Pascal 5.5 programming language.

The implemented system is an ITS that employs the features of set theory such as hierarchical structure, inheritably deductable operation and relations and set concept being the core of the theory to create a tutor that teaches the concept and monitors the user's state of knowledge. The system uses a distributed control strategy that allows four factors, namely student, teacher, student model and nondeterminism to possess the right to direct a session and its contents. Nondeterminism is used to generate the instructional content by randomly selecting different questions and examples each time the program is invoked. Finally, the system ends the tutorial session by giving a final examination to the user and monitoring any misconceived issues in order to repeat the related sections.

Keywords : Intelligent tutoring systems, object oriented programming, control strategy in ITS.

ÖZET

KÜME TEORİSİ ÖĞRETEN NESNEYE DAYALI BİR AKILLI YARDIMCI SİSTEM

EMEL (KERİMOĞLU) CANKAT

Bilgisayar Mühendisliği ve Enformatik Bilimleri Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. H. Altay Güvenir
Haziran 1991

Bu tez çalışmasında Küme Teorisi öğretmek için bir Akıllı Yardımcı Sistem (AYS) tasarlanmış ve nesneye dayalı yaklaşım kullanılarak gerçekleştirilmiştir. Bu program IBM PS/2 ve IBM uyumlu bilgisayarlarda Turbo Pascal sürüm 5.5 programlamlama dili ile geliştirilmiştir.

Sistemin, öğretme modülü, uzman modülü ve öğrenci modeli olmak üzere üç ana bileşeni vardır. Sistemin akış kontrolü öğrenci, öğretmen ve öğrenci modeli arasında paylaştırılmıştır. Tasarlanan ve gerçekleştirilen sistemin en önemli özelliği örnek ve soruların sistem tarafından rassal olarak üretilmiştir. Sistemin bütün modülleri nesnesel bir yaklaşımla gerçekleştirilmiştir.

Anahtar kelimeler: Akıllı Yardımcı Sistemler, Nesneye Dayalı Programlama, Akıllı Yardımcı Sistemlerde Kontrol Stratejisi.

ACKNOWLEDGEMENT

I would like to gratefully acknowledge the valuable assistance and help of my supervisor Assist. Prof. Dr. Altay Güvenir who made the completion of this work possible. Also, I would like to thank my manager at IBM Mr. Aydın Kolat and my colleagues at the IBM Computer Aided Instruction R&D Center. Furthermore my gratitude goes to the Dr. Zeki Kocabıyıkoğlu and Sinan Şenol for allowing me to use their facilities. I would also like to thank Mr. Mehmet Nadir Erhan for his assistance in printing the thesis.

Finally, I would like to express my endless thanks to my husband for his continuous support and assistance throughout all phases of this study.

TABLE OF CONTENTS

1. INTRODUCTION 1

2. INTELLIGENT TUTORING SYSTEMS 6

2.1. COMPARISON OF CAI AND ITS 7

2.2. PARADIGMS AND COMPONENTS 11

2.2.1. PARADIGMS 11

2.2.2. ITS COMPONENTS 12

2.3. ITS ARCHITECTURES 15

2.4. SURVEY OF SPECIFIC SYSTEMS 19

3. OBJECT-ORIENTED PROGRAMMING 25

3.1. INTRODUCTION 25

3.2. OBJECTS 26

3.3. CLASSES 28

3.4. MESSAGES 29

3.5. METHODS 30

3.6. ATTRIBUTES OF OOP 32

3.6.1. DATA ABSTRACTION 32

3.6.2. INDEPENDENCE 33

3.6.3. INHERITANCE 33

3.6.4. MESSAGE PASSING 35

3.6.5. POLYMORPHISM 35

3.6.6. HOMOGENEITY 35

3.6.7. DYNAMIC BINDING 36

3.7. CONCLUSION 36

4. IMPLEMENTATION 38

4.1. AN OVERVIEW OF THE SET THEORY TUTOR 38

4.2. THE MODULES OF THE SET THEORY TUTOR 39

4.2.1. EXPERT MODULE 39

4.2.1.1.GENERATION OF EXAMPLES 44

4.2.1.2.GENERATION OF QUESTIONS 55

4.2.2. THE STUDENT MODEL 64

4.2.2.1.ANALYSIS OF STUDENT'S RESPONSES 65

4.2.3. THE TEACHING MODULE 66

5. CONCLUSIONS AND FURTHER RESEARCH 81

6. REFERENCES 83

LIST OF FIGURES

1. ITS domains 7
2. ITS paradigms 12
3. General ITS Structure 16
4. Components of an ITS 17
5. Architecture of SEDAF 18
6. Self improving ITS architectures 19
7. An object universe 27
8. An object encapsulated by its message protocol 31
9. An object hierarchy 34
10. Structure of the implemented system 40
11. Definition of "setobj" 41
12. Definition of "ex_set" 42
13. Definition of method "ex_set.Make" 43
14. The source code segment to generate a set example 43
15. Definition of "example" 44
16. Set theory structure 45
17. Definition of "ex_sets" 45
18. Common attributes of relations 46
19. Definition of "ex_relation" 46
20. Definition of "ex_subset" 47
21. Definition of "ex_not_subset" 47
22. Definition of "ex_equal" and "ex_not_equal" 47
23. Definition of "ex_member" and "ex_not_member" 48
24. Definition of "ex_subset.make" 49
25. A source code segment to generate subset example 50
26. Common attributes of operations 50
27. Definition of examples for operations 51
28. Definition of "ex_intersection_make" 53
29. A source code segment to generate intersection example 54
30. Class of objects generating examples 54
31. Definition of "question" 55
32. Definition of all question generating objecs 58
33. The class of objects generating questions 59
34. Definition of "que_memberynQuemake" and"que_member.multQueMake" 60
35. A source code segment to generate member question 61

36. Definition of "que_difference.multQueMake" 63
37. A source code segment to generate difference question 63
38. Control Strategy in ITS 68
39. Definition of "topic" 71
40. Some question generating objects 73
41. Definition of "Relationtopic" 73
42. A source code segment for teaching relations 74
43. Teach method of "operationTopic" 74
44. Some objects teaching operations 75
45. Definition of "SetTheoryTopic" 76
46. Teach method of "SetTheoryTopic" 76
47. Mechanism for teaching set theory 76
48. The main program 77
49. An example of universe selection 78
50. Distribution of difficulty level in set theory 79

1. INTRODUCTION

The remarkable evolution of electronics within the last decades has stimulated the use of computers in nearly all aspects of our daily life. Following these enhancements, computers have been widely used for various educational applications throughout all phases of our academic life from elementary school to university. Consequently, the data manipulation, storage and presentation capabilities of computers have been employed for traditional applications for educational purposes, especially as *Tutoring Systems.*

The first systems were basically simple programs that run in a predetermined manner and aim at teaching a subject to the user. These programs lack any sort of intelligence and interactive decision making and thus are more like electronic books with attractive pages. It is also obvious that the fast pace of social and technological transformations has led to new educational needs, and thus AI techniques were more frequently consulted to aid educational activities. The systems that emerged as a consequence of these efforts are called *Intelligent Tutoring Systems* (ITS) or *Intelligent Computer Aided Instruction* (ICAI) [8]. A simple and brief explanation of ITS is given by Nwana [18];

> "ITS are computer programs that are designed to incorporate techniques from the AI community in order to provide tutors which know what they teach, who they teach and how to teach it."

In other words AI tries to imitate in a computer behaviour which, if done by a human, would be described as intelligent and thus ITS may similarly be described as an attempt to generate in computer behaviour which, if done by a human, would be called "good teaching" [18]. The motivation beyond producing computer-based tutors lies mainly in two factors;

- **Theoretical:** This factor is a consequence of the property of ITS being an multi-disciplinary area composing computer science, psychology and education. These attributes of ITS allow the scientists to test many theories from cognitive psychology.

- **Application:** There exists several advantages of ITS's over human tutors due to many economic and social reasons. But mainly the most important advantage of these systems is one-to-one tutoring that allows the student to regulate the pace of the session and repeat and practice as much as he desires.

Before going any further we would like to clarify any methodology confusion about the term ICAI and ITS. According to Wenger [25] the preference of ITS is motivated by the claim that, in many ways, the significance of the shift in research methodology goes beyond the addition of an 'I' to CAI. However, some researchers also use *Knowledge Based Tutoring System* (KBTS), *Adaptive Tutoring Systems* (ATS) and *Knowledge Communication Systems* (KCS) and recently Intelligent Education Systems (IES) instead of ITS. This work will use the terms ITS and ICAI as synonyms interchangeably.

The work described here deals with design and implementation of an ICAI system that teaches set theory in secondary school level. The overall objective of this system is to create an educational medium that stands between books and teachers. From an educatioal perspective the ICAI system implemented can also be classified as Generative CAI which can be defined as: a method that involves writing a computer program to generate material (i.e. problems, solutions and associated diagnostics) as and when it is needed during a teaching session [19].

The selection of *set theory* among the numerous topics available within the educational spectra can be justified by the following arguments;

- It is easy to teach (define) the computer the basics and operational mechanics of the theory. It is essentially important to be able to code the concept, that is intended to be taught, to the computer in order for the system to exhibit any sort of intelligence. This ability will allow the system to generate instructional material and evaluate the student's responses.

- The rules and operations of set theory are well defined and highly suitable for computer applications. For instance operations like intersection, union, difference, etc. are very easy to encode. One drawback of this subject is the difficulty of representing the concept of infinite set.

- Set theory is convenient for rapid and easy question and example generation. The significance of dealing with numerous questions in developing problem solving skills will be discussed later in this section.

- The *Set* concept forms a base for all operations within set theory. Therefore the structure of this theory heavily depends on sets and thus allows us to implement a hierarchical framework on which we can construct our ICAI model.

Under the light of above discussion set theory was selected as the topic to be taught.

One important point that should be kept in mind is that this work mainly focuses at intelligently generating examples and questions which allow the student to develop a knowledge base about a certain subject. The effect of experience in problem solving has been studied by many researchers [12], [3], [4]. One of the earliest studies about this topic has been performed by a Dutch Psychologist Adrian DeGroot in the 1960's [13]. DeGroot based his research on investigating why chess masters were better than skilled, yet less accomplished, players. DeGroot's initial assumptions were that masters are able to consider more future possibilities, judging all

potential strengths and weakness of each possible move. But contrary to these assumptions, experiments with masters and skilled players have shown that the masters superiority resulted from the choice of qualitatively better moves than less-skilled players.

These results initiated a second hypothesis by deGroot: masters due to their vast experience have developed a knowledge base that allows them to perceive various game positions and thus perform substantially better moves. To prove this DeGroot allowed both parties to view a chess scenario for a short time and then asked them to reproduce the scene with the new pieces. Results indicated that masters were excellent in short-term memory. However subsequent research [3] showed that when chess pieces were placed randomly without any meaning masters were no better than skilled players. The masters superiority emerged only if the configurations were meaningful. Further studies on bridge players, physics experts, cardiologists have also supported the conclusion that expert performance depends heavily on the capability to employ the knowledge acquired through past experience [3], [4].

On the other hand, educationalists accept as fact that the retention of the knowledge acquired by experience can be achieved through large number of examples. Therefore, in this research, one of the design decisions made was to develop an ITS which can generate intelligently a large number of different examples and questions in the hope of improving retention.

Another design decision focuses on the programming paradigm to be employed. Recently, *Object-Oriented programming* (OOP) has been successfully applied to programming projects in many different disciplines. Set theory, itself, seemed suitable for representation in object-oriented paradigm. Also, we wanted to investigate the applicability of OOP in the implementation of intelligent tutoring systems.

The selection of OOP is due to the suitability of the three main characteristic properties that are *Encapsulation, Inheritance* and *Polymorphism,* to the internal structure of set theory.

The second chapter will define intelligent tutoring systems in general, and compare them with CAI. The general structure of an ITS will be presented. Also some example ITS systems will be described.

Object-oriented paradigm will be presented in the third chapter. The characteristics of OOP will be explained with examples.

The fourth chapter describes the implementation of a system for teaching set theory combining the characteristics of ITS and object-oriented paradigm. The techniques used in the generation of examples and questions will be explained. It will be shown that the representation of both expert knowledge of set theory and the teaching strategy in OOP is clear and efficient.

The fifth chapter concludes with the results of the research, and provides some possible further improvements and research areas.

2. INTELLIGENT TUTORING SYSTEMS

Intelligent Computer Aided Instruction (ICAI) or Intelligent Tutoring Systems (ITS) are systems that use *Artificial Intelligence* (AI) techniques while teaching a subject to a student. There are a number of reasons behind calling these systems as intelligent. These can be briefly described as [5];

- Ability to solve the questions they ask the student.
- Capability of individualized instruction.
- Allowing branching.
- Decide what to do next by themselves.

ITS is an integrated field that involves Computer Science (CS), Cognitive Psychology and Educational Research which is generally known as "Cognitive Science" (Fig. 1).

The fact that ITS spans three disciplines, has important implications; Artificial Intelligence (AI), Computer Aided Instruction (CAI) and Cognition have to be perfectly coordinated in order to obtain improvements and successes. These distinct fields have major differences in research goals, terminology and theoretical frameworks. Therefore, ITS research requires mutual understanding of the three fields involved [12].

ITS systems are potentially more powerful than CAI systems, because they allow the student to explore the subjects according to his or her hypothesis and interests. An ITS is an open-ended system. Student can also control the flow trend of program which leads to reactive learning environment. In fact most ITS systems possess a mixed-initiative control property that distributes the flow of the learning session to both the user and the tutorial intervention part of the system [6].

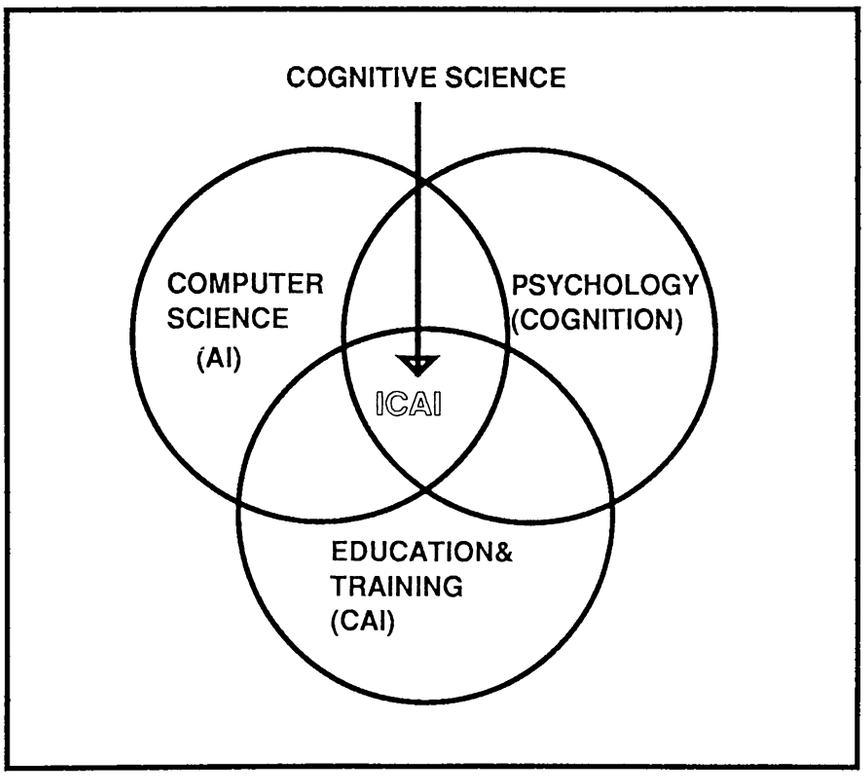

Fig.1. ITS domains (adopted from [12]).

2.1. COMPARISON OF CAI AND ITS

Education has been regarded as a major application field of computers since 1950's. Computer-assisted instruction (CAI) concept arose during this era and continuously evolved since then. A CAI program can be described as "a revolutionary successors of books" [8] which contain both the domain and the tutoring knowledge of expert human teachers. Generally, CAI programs are termed as branching programs since during their execution the program continuously follows a predetermined framework depending on the answers it receives from the student. For example; it is built into the program that if the answer is A go to section 1 or if the answer is B proceed to section 5 etc. Therefore, traditional CAI benefits from the experience of expert teachers and directly reflects it to the behaviour of the program. This capability is actually the main strength of CAI approach while it is also the main weakness. In reality, it is nearly impossible to consider all possible misconceptions a student can

acquire. Furthermore, it is practically not feasible to develop a software to handle all these misconceptions if they could be determined. Carefully developed and tested CAI programs can be safely used by a large number of people but they are hard to modify according to the evolving design principles.

CAI concept can be classified according to the level of computer control over the learning activity. This classification ranges from free learning environments such as LOGO microworlds to strict guided learning strategies. The central problem with most CAI systems is their deficiency in providing rich feedback and individualization resulting from their incapability of knowing what they teach, who they teach it and how to teach it [18]. The main disadvantages of CAI can be listed as [8]:

(1) Software quality is closely related to the designers capability to specify a high number of possible answers and their corresponding tutoring path.

(2) The selected tutoring strategy of the particular CAI program may not suit the student's specific needs.

(3) Autonomy provided in less directive systems is a handicap for students who can not exploit the opportunity.

(4) They are mostly not cost-effective in the sense that their construction and maintenance calls for considerable resources.

In the late 1960's to early 1970's CAI evolved into Generative CAI systems that were built upon the requirement that the teaching material could itself be generated by the computer. Generative systems have the capability of generating and solving meaningful problems. These systems were the ancestors of ITSs; laking the ability to possess human-like knowledge of the domain and the ability to teaching and answering serious questions like "why" and "how" that a student might ask [18].

Next stage within the development of CAI is the ITSs that employ AI techniques to create systems with human-like intelligence, judgement, inference and teaching capabilities.

In summary besides these general issues there are some very fundamental differences between ITS and CAI systems as pointed out by Kearsley [12]. These differences are the following:

Development Goals: CAI has been developed by educational researchers and teachers to solve their practical problems using computers while ITS has been specially developed by computer scientists to measure or see the capability of AI techniques in the process of learning and teaching.

Theoretical Basis: Generally, CAI programs are built upon some principles of learning and instruction. However, even with the adaptation of the systems approach, CAI development is limited by the system designer's knowledge of learning and instruction. On the other hand, ITS combines AI techniques and the instructional process mainly aiming at exploring the cognitive process behind learning and teaching specific tasks. Thus, ITS researchers based their system on theoretical notions of Cognitive Science which emerged from the information processing theory in cognitive psychology.

System Structures and Functions: Most CAI systems store and implement their instructional components in a single structure. Operational procedures of these systems are determined by previously entered specific pieces of information and algorithmic processes. This style of CAI that leaves little or no initiative to the user in the instructional process, is often named *"Ad-hoc Frame-oriented"*. ITS, on the other hand, use spontaneous inferential processes to diagnose the student's learning needs and prescribe instructional treatments. Therefore in contrast to CAI, ITS systems always allow the instructional process to be initiated by the system.

Instructional Principles: Since basically, CAI is an instructional delivery system, the main instructional methods used

within the system are not very much different from techniques used in schools and other training environments, except for interactive, individualized instruction capability of computers. This approach is also named "teacher-centered expository" approach which utilizes *Skinnerian Behaviourism*. In contrast, ITS systems adopt *Dewey's philosophy* which is "learning-by-doing". In this instructional approach the user is required to engage in interaction with the system and create what's called a *"reactive environment"*.

Methods of Structuring Knowledge: CAI, commonly utilizes task analysis to identify tasks and subtasks to be taught and content elements required to learn these tasks. This is a systematic method used to define tasks that utilizes either an algorithmic approach or a hierarchical approach. On the other hand, in ITS systems, the methods for structuring knowledge to be taught are determined from the AI knowledge representation technique which is determined by the system designer. This technique is rather a method to organize knowledge into a data structure.

Methods of Student Modelling: The methods used in CAI for student modelling were binary judgement in the beginning and quantitative methods later on. While ITS's student modelling method is solely qualitative where student's learning is judged from the responses or response patterns.

Instructional Formats: The most common CAI formats are *tutorial, drill* and *practice, games* and *simulations*. Games are further divided into two; *intrinsic games* and *extrinsic games*. Simulations can also be of various types like *physical, situational* and *process* simulations. ITS, on the contrary, can be classified into two; tutorials and games. CAI and ITS tutorials are quite different from each other in the sense that CAI tutorials emphasize the system's expository representation of instruction while ITS tutorials are based on question-and-answer driven interaction. Another sharp distinction also arises in both CAI's and ITS's use of games. CAI uses games either to teach gaming rules and skills (intrinsic games), or to maintain the student's attention (extrinsic games),

while ITS uses games to provide a reactive learning environment to allow students to explore their own interests.

Subject Matter Areas: CAI can be applied to a variety of subject matter areas ranging from mathematics to art. However ITS is rather limited within the scope of well-structured subject matter areas such as mathematics.

2.2. PARADIGMS AND COMPONENTS

The following sections describe the five major paradigms of ITS. Next, the three essential components of ITS are presented.

2.2.1. PARADIGMS

ITS domain contains five major paradigms, as shown in Fig.2.The first paradigm is the *mixed initiative dialogues*, which represent the original ITS paradigm. In this type of ITS the program engages the student in a two way conversation and attempts to teach the student via the socratic method of *guided discovery*. The paradigm best fits conceptual and procedural learning tasks.

The second paradigm is *coaches*. A coach observes the student's performance and provides advice that will help the student to perform better. Coaches are best suited to the problem solving types of programs.

A third paradigm is *diagnostic tutors* that debug a student's work. These programs are driven by a bug catalogue that identifies the misconceptions that students may have in solving a problem.

A fourth ITS paradigm is the *microworld* concept which involves developing a concept tool that allows a student to explore a problem domain such as geometry or physics.

The last one is *articulate expert systems* which can be used as job aids and provide practice in problem solving and decision-making skills.

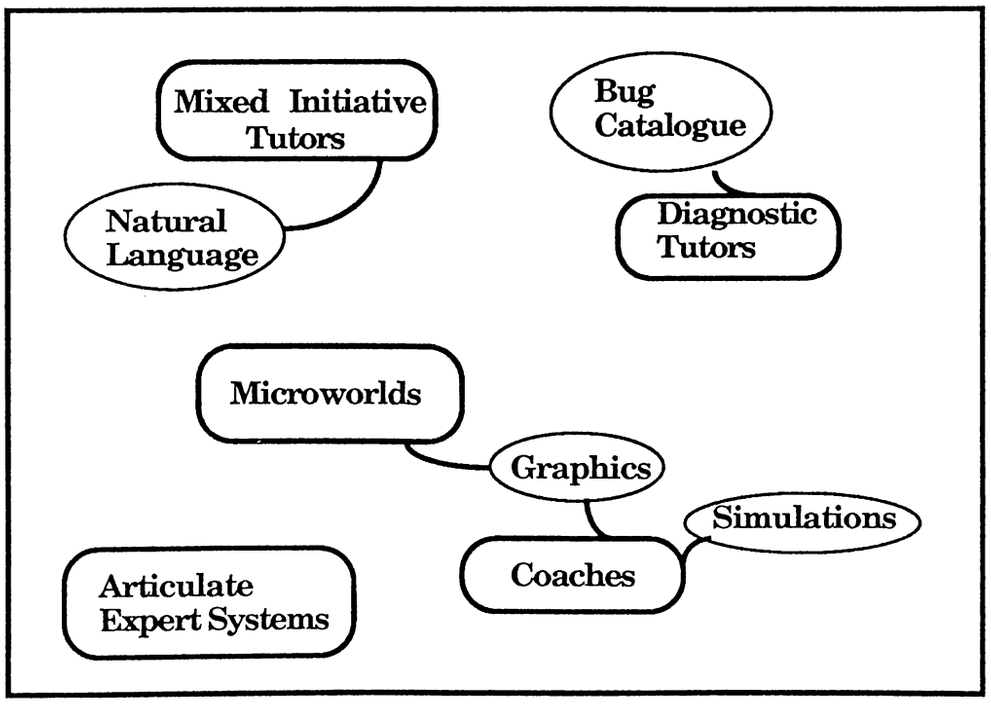

Fig.2. ITS paradigms (adopted from [12]).

When studying ITS programs that use different paradigms it is very important to keep in mind the fact that each paradigm is associated with only a certain set of cognitive science issues and ignore the rest. No paradigm covers all ITS concerns nor does any existing ITS program span more than one paradigm. But since this field is rapidly growing ITS programs are most likely to broaden in scope.

2.2.2. ITS COMPONENTS

Generally there are three basic units within an ITS system known as; Expertise Module, Student Model and Tutoring Module.

EXPERTISE MODULE: The domain knowledge that the system intends to teach is contained in this module. Furthermore, the actual model of the skills and concepts to be taught to the student is comprised within this module which furnishes it with a dynamic form of expertise within the specific area. There are mainly two functions performed by this component [8];

a) A means of developing questions, answers and explanations and thus behaving as a source.

b) A means of evaluating the users performance to set standards to determine the level of comprehension.

Since this module is employed in generation of the instructional content and evaluation of the student's performance, its domain knowledge needs to be organized within the framework of a computer program for the sake of easy manipulation. This organization is often time-consuming and complex and thus searching for methods of organizing and presenting knowledge is a major issue in developing an expertise module [18]. Some common AI methods used in domain knowledge organization are; *semantic networks, production systems, procedural representations and scripts-frames* [12].

Semantic networks contain all the factual information that is required for teaching the subject in a large, static database which is based on psychological models of human associative memory. *Production systems* are employed to form modular representations of skills and problem solving method. The knowledge database of these systems consists of productions which are rules in the form of condition-action pairs like

if <*conditions*> **then** <*actions*>.

Procedural representations contain the subskills that a student must possess in order to grasp the skill being taught in a well specified situation. *Scripts-frames* are data structures including declarative and procedural information in predefined internal relations.

STUDENT MODEL: A student model is used to measure the student's knowledge state and further try to guess his or her conceptions and reasoning methods employed to reach his or her current knowledge state. This is done by comparing student's performance to the computer-based expert's behaviour on the same task. Modelling the student's knowledge and learning behaviour uses basically two procedures.

a) Charting within the knowledge structure network, those areas which the student has mastered or has attempted to learn. In other words, the student's level of knowledge is compared to that of the expert and the resultant subset is used to measure the level of mastery. This technique is called overlay. The student's behaviour is recognized by the system simply using the correct and incorrect knowledge patterns present within the knowledge base [8].

b) Applying pattern recognition to the student's response history for making inferences about his or her understanding of the skill and the reasoning process used to derive the response. This phase is also termed as *diagnosis* and can employ method such as statistical analysis [8].

In the ideal case the student model should include all aspects of the student's behaviour and knowledge that may effect performance and learning, but in reality, forming such a model is obviously impossible considering that human behaviour is a composite topic which is a combination of all senses, sight, voice or even facial gestures. Thus, since in an ITS system the keyboard is the only means of communication, it lacks most state of mind detection capabilities of human tutors. Generally the functions for which a student model can be used are [18];

(1) **Corrective:** Guide to eliminate bugs in student's knowledge.

(2) **Elaborative:** Guide in completing the student's knowledge.

(3) **Strategic:** Help in managing major deviations in tutorial strategy other than mentioned in (1) and (2).

(4) **Diagnostic:** To aid in determining bugs in students knowledge.

(5) **Predictive:** To assist in determining the probable reaction of the student towards the tutorial action.

(6) **Evaluative :** To assist in evaluating the student or ITS.

TUTORIAL MODULE: The tutorial module consists of a set of specifications of what instructional material the system should present and the method and timing of the presentation. Generally in most of the present ITS systems, two methods of presentation exists: *Socratic and coaching method.* The first method employs a set of questions that direct the student through a process of debugging their own misconceptions. The latter one, on the other hand, creates an enjoyable environment like computer games, for the student so that he or she can learn related skills and general problem-solving abilities.

2.3. ITS ARCHITECTURES

ITS systems have taken on many forms, but essentially they have separated the major components of an instructional system in a way that allows both the student and system a flexibility in the learning environment that closely resembles what actually occurs when student and teacher sit down one-on-one and attempt to teach and learn together.

The number and variety of architectures used in existing ITS are surprisingly large mainly due to the experimental nature of the work in this area. Still, consensus is achieved in the literature that ITS contain four basic modules [18], [12], [8] ;

- Expert Knowledge Module
- Student Model Module
- Tutoring Module
- User Interface Module

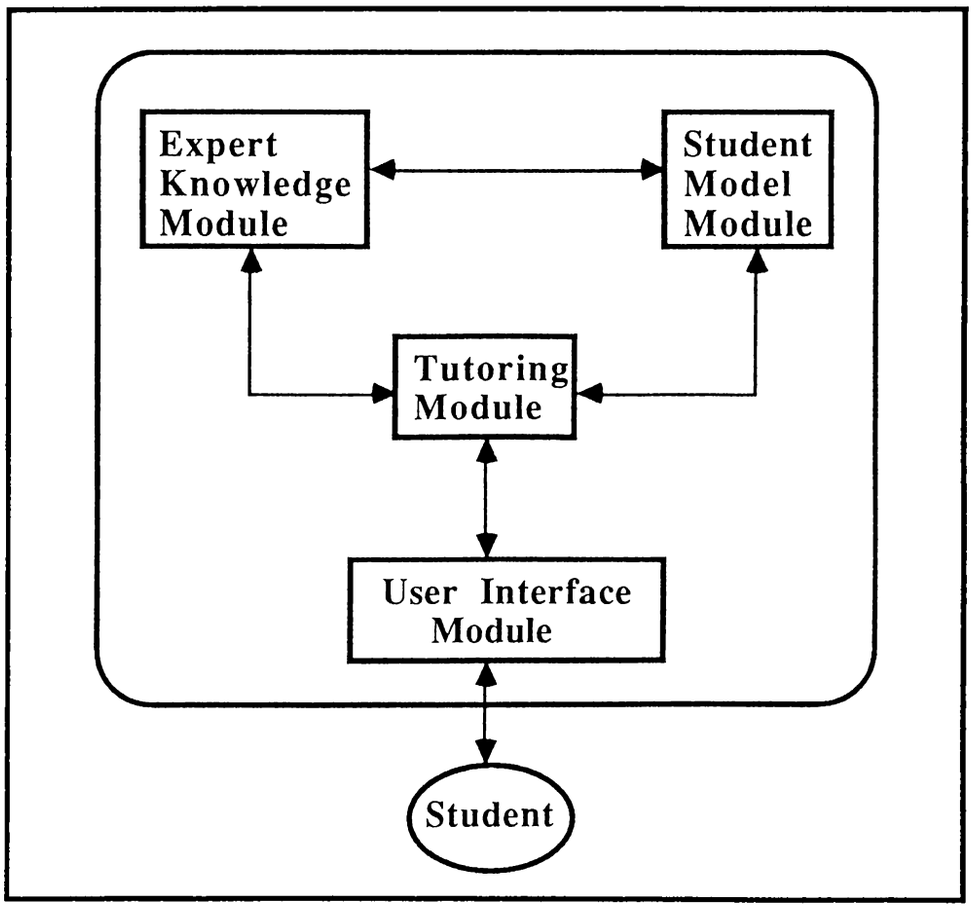

Fig.3. General ITS Structure (adopted from [18]).

The general structure of ITS given in Fig.3. This figure is a general representation and does not represent any specific system. In application, there exists systems that fit the general structure with slight modifications and systems that are totally different. A slightly modified structure is given in Fig.4 [5].

This architecture contains the "Modeller" as an addition to the general structure. The idealized information is input to the modeller from the expert simulator. In addition, information like types of students, student behaviour and what students know according to their background is also fed to this module from the knowledge base. These inputs are evaluated within the modeller and the student model is updated accordingly whenever necessary.

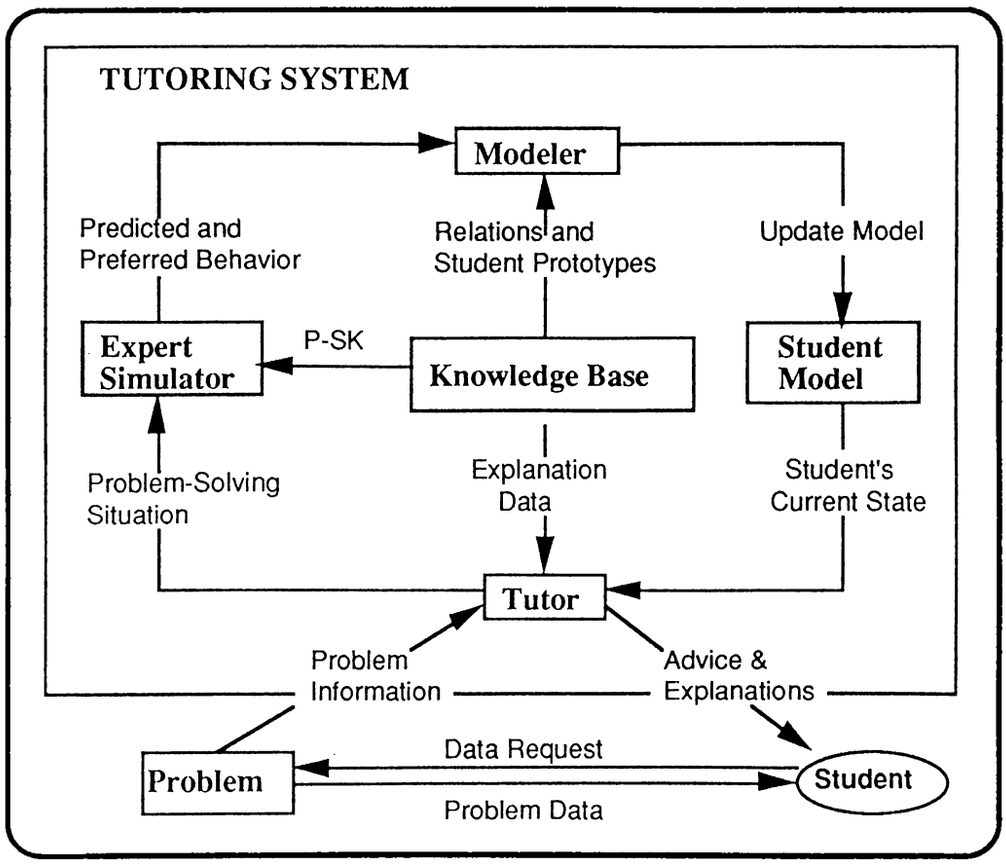

Fig.4. Components of an ITS (adopted from [5]).

One thing that must be kept under consideration is that the patterns that will be generated by the modeller can be too complex to manage. Therefore, this structure is an ideal structure that is hard to realize. An existing example that suits the first structure we have explained before is the "SEDAF" [2] which is an intelligent system for teaching users how to study graphs of mathematical functions (Fig.5). SEDAF architecture encompasses the following components [2] ;

- Expert Module contains two knowledge bases; correct knowledge and the misconceptions.
- Diagnosis Module attempts to find the causes for the user's errors.

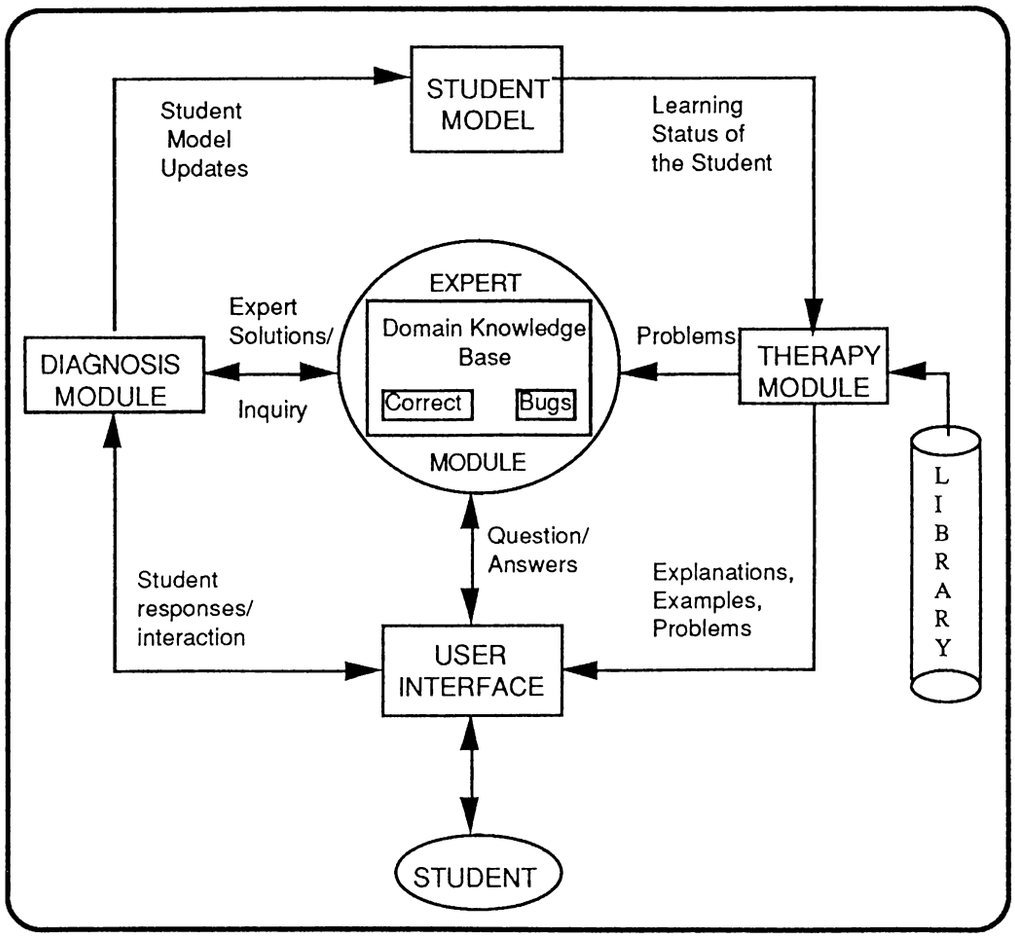

Fig.5. Architecture of SEDAF (adopted from [2])

- Student Model contains a description of the learning status of the students.
- Therapy Module embodies the teaching expertise of the system.
- User Interface.

Finally, an architecture that is radically different from the common structure that introduces a self-improving module is given in Fig.6.

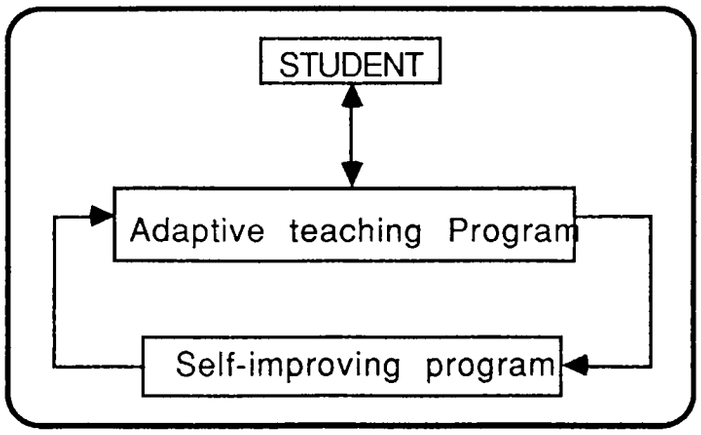

Fig.6. Self improving ITS architectures (adopted from [18])

2.4. SURVEY OF SPECIFIC SYSTEMS:

There has been a considerable amount of research and actual work on ITS in the past. Among these systems there are certain distinguished ones that need to be analyzed and investigated to a large extent. This section aims at reviewing some systems we believe, that either play a historical role or possess the basic principles of ITS mentioned previously. These systems can be listed as;

* SCHOLAR (1970)
* SOPHIE (1975)
* WEST (1978)
* DEBUGGY (1978)
* MENO (1981)
* WHY (1982)
* PROUST (1982)
* STEAMER (1984)
* GUIDON (1987)

The *SCHOLAR* (Carbonnel, 1970) was the first pioneer in the area of intelligent tutoring [5] [18]. Carbonnel, considered the founder of ITS's, developed a new paradigm which was called *Information-structure-oriented* (ISO) as opposed to traditional *Ad-hoc-frame-oriented* (AFO) structure that was being used in CAI. He realized that the domain knowledge to be taught can be separated from the

teaching knowledge. Some other important characteristics embedded in SCHOLAR are;

- A complex but well defined database structure in the form of a network of facts, concepts and procedures.
- Network containing information-defining words and events in the form of multi-level trees.
- Socratic style of tutoring dialogue.
- Inference strategies for answers are independent of the content of the semantic net.

SCHOLAR was a mixed-initiative ITS that aimed at teaching *South American geography* to students. The expert knowledge module of SCHOLAR contained geography of South America preserved in the form of a semantic network. This structure relieves the system from memory problems since it allows answering questions whose answers were not stored. Besides these revolutionary improvements, there were certain disadvantages or weak points in SCHOLAR simply because it was one of the first systems in ITS area. The disadvantage can be listed as;

- Tutorial strategies were rather primitive, mostly depending on local topic selections once an agenda was input to the system by the teacher.
- Language processing capabilities were rather constrained since text was produced by sentence and question templates selected from the network. Consequently, it was not able to understand wrong answers and thus could not determine the student's level of understanding.

Even though SCHOLAR had some shortcomings it has introduced many concepts that are vital for ITS design and implementation.

The *SOPHIE* (Sophisticated Instructional Environment) followed SCHOLAR and yielded more promising results [5] [18]. SOPHIE (J.S Brown) also used the mixed-initiative CAI concept to create an environment where a user learns-by-doing as opposed to learning-by-being-told. SOPHIE's expertise laid in the area of electronic trouble shooting. The four main modules of SOPHIE set very good examples for well-designed ITS systems;

a) **Expert Module:** It contains a strong module of the selected topic which not only solves problems but it can generate tactical approaches and high level strategies for attacking the problem.

b) **Tutorial Module:** This module contains a complex structure which contains heuristics for answering, critiquing and generating alternatives against student's hypothesis.

c) **Student Model:** The tutorial module defined above naturally implies that there must be a student model at least as sophisticated to allow proper system operation.

d) **User Interface:** SOPHIE contains a well established, efficient, robust natural language capability designed by Burton using semantic grammars.

WEST is one of the first examples of a coach (Burton and Brown) that aims at teaching students to play a game called *"How the West was won"* [5] [18]. The game is based on numbers that are determined from three dials used by each player. The three numbers obtained can then be manipulated using four elementary operations and parentheses. WEST is a program that exhibits perfectly how more emphasis on various modules of ITS can yield totally different systems. Therefore, WEST is called a *"Coach"* rather than a tutor due to the informal teaching atmosphere it establishes.

WEST uses comparison to evaluate the student's ability to write algebraic equations against that of expert solution present in the

database. WEST's expert module encompasses a simulated board game and an articulate expert that is capable of monitoring and evaluating the user's moves. The student model, on the other hand, is built upon a method that is a simple overlay model called *"Differential Modelling"* which is based on the assumption that the student's moves are never wrong but only poor. The model uses the difference of the student's calculated total to that of the expert's to advise on the topic.

DEBUGGY is a program developed for modelling a user's knowledge about the subtraction procedure [5]. DEBUGGY generates a procedure about how a particular user performs subtraction simply by tracing through a set of example problems solved by the user. The main hypothesis behind DEBUGGY is the idea that there are common systematic errors that people make in most cases and these errors can be determined by analyzing problems.

MENO is an ITS which tries to determine a user's programming bug to reason about his/her misconceptions (Soloway et al. 1981). MENO carries this a step further and also advices the student about the misconception [5]. To achieve this MENO uses a library of misconceptions, a semantic net, a parse of the program and then relates the variables and procedures to an internal model of the code. The next step is associating certain bugs with this model. Unfortunately, the program is not generative, that is it is not able to analyze any code, thus the program to be analyzed must be an average program that is already available within the domain.

The *WHY* program is work conducted by Collins and Stevens (1980) following SCHOLAR [5]. In this system the domain of SCHOLAR which consisted of purely factual reasoning methods was replaced by casual reasoning. The reasoning rules that are used to generate questions for the students to answer are stated abstractly, including certain predictions, case studies, general rules and previous causes.

The general approach that Collins tried to use in the dialogues was the Socratic approach, found in the writings of Plato. In addition

to these, a tutoring plan which determined the subjects that the student did not understand was included within WHY. As a result misconceptions, one of the misunderstood topics was selected and the tutorial rerun. This feature of WHY which allowed determination of the misconception was the most important characteristic.

PROUST is a knowledge-based system designed for finding non-syntactic bugs in Pascal programs written by novice programmers. It was implemented by Soloway et al., in 1983 [12]. It finds all kinds of bugs. In addition to this, it determines how the bugs could be eliminated. PROUST analyzes programs using an analysis-by-synthesis approach. PROUST investigates the program requirements that are previously defined in order to suggest methods for satisfying these requirements. Then the system compares each possible method with the programmer's method and thus requires programming knowledge. The two kinds of knowledge; goals and plans, are frame-based. Goals are problem requirements while plans are stereotypic methods for implementing goals. PROUST, as an ITS, was quite successful and managed to find most bugs in code written by inexperienced programmers. The next logical step that can be built upon PROUST is an automated programming course that not only corrects student's mistakes but also provides them with examples to give student practice whenever needed.

STEAMER was a project (Hollan, Hutchins and Weitzman 1984) which aimed at evaluating the potential of new AI hardware and software technology, especially in the construction of computer-based training systems . The subject selected was steam propulsion engineering for a number of reasons [12] [5]:

1) A critical need for improvement in this topic.
2) Relative costs of alternative training methods quite high.
3) A mathematical steam propulsion system model was available.
4) A non-tactical subject.
5) Graphical interfaces highly applicable.

This project was a good example of the importance of the user interface which employed graphical representations that add a visual dimension to the ITS.

GUIDON is an ITS that aims at teaching diagnostic problem solving skills (Clancey 1984) [18]. This project is unique since it represents one of the first attempts to adapt an existing expert system into an ITS. It had been heavily influence by SCHOLAR and SOPHIE but after undergoing long and tedious stages of planning yielded nearly as important findings.The expert system used was *MYCIN* (Shortliffe, 1976) which was a well established medical expert system for treating bacterial infections. The way GUIDON operated was mainly by case dialogues in which students are presented with a sick patient and were required to ask questions relative to the case. These questions were compared with those MYCIN would have asked and the student evaluated accordingly. Therefore this shows a different tutoring strategy than SCHOLAR and SOPHIE. Also it can be derived from the previous sentence that student modeling method is overlay modelling. GUIDON also separates its tutorial strategies from its domain knowledge just like in SOPHIE. Furthermore, natural language capabilities of GUIDON was far less developed than SOPHIE's but better than SCHOLAR's. Still GUIDON project provided an insight into intelligent tutoring while producing valuable hints and guidelines for designing ITSs.

3. OBJECT-ORIENTED PROGRAMMING

This chapter describes the Object-oriented (O-O) programming technique used to develop the Intelligent Set Theory Tutor. A brief introduction to the basic principles of O-O approach, its main attributes and structure is contained within this section.

3.1. INTRODUCTION

Object-oriented programming is a method which leads to software architectures based on the objects every system or subsystem manipulates. O-O design is the process of decomposing a problem into objects and establishing the relations between them. The technique aims at identifying those objects in the real world that must be manipulated to reach a solution to the selected problem. Then, these determined objects are simulated within the computer to arrive at a program that performs the desired actions. There exists a number of notions that are currently associated with the O-O approach such as;

(1) Data abstraction (Encapsulation)

(2) Independence

(3) Inheritance

(4) Message-passing

(5) Overloading (Polymorphism)

(6) Homogeneity

(7) Late binding

The object-oriented method employs a data or object-centered approach to programming which is different from data-procedure paradigm used by many programming languages. In this approach objects are asked to perform operations on themselves rather than

passing data to procedures. The notions listed above originate from this different paradigm used in OOP.

Object-oriented programming is a new technique that has been widely used in a variety of disciplines like management science, information systems, software development and management, database management, artificial intelligence and educational applications. The popularity of OOP is believed to come from the benefits listed below:

- O-O techniques allow the use of methods to formalize most methods of reality.
- O-O methods allow complex systems to be modelled.
- O-O systems allow handling of structures that are inherent in character.
- O-O systems allow for cost reduction both in programmer productivity and maintenance costs.
- O-O programs resist, to a degree, accidental and malicious corruption attempts.

Before going into characteristics of OOP, it is better to examine the building blocks of this approach that are;

1. Objects
2. Classes
3. Methods
4. Messages

3.2. OBJECTS

In OOP an object is an entity which is a package of information and descriptions of its manipulation that combine the attributes of procedures and data (Fig. 7). They are the primitive element of OOP.

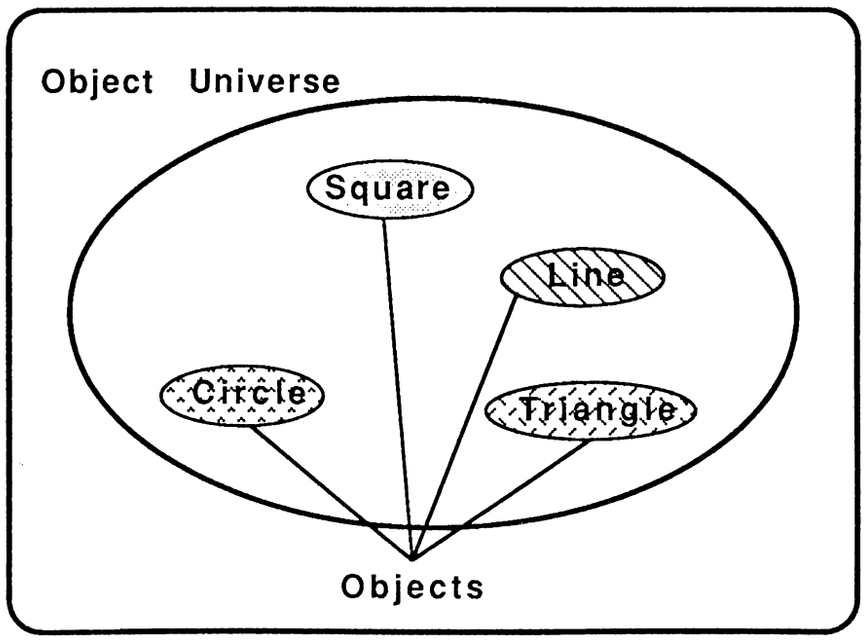

Fig.7 An object universe

The basic functions of objects are to store data in variables and respond to incoming messages by performing methods.

The internal structure of an object contains two sections: a public and a private part. This structure assures the information hiding principle which states:

"All information about an object should be private to the object unless it is specifically declared public."

This structure allows the object to be known to the rest of the universe through some official public interface which represent some of the module's properties while keeping the rest in its private section. The main use of this structure is to preserve the continuity of the structure which allows a change in the private part to be performed without affecting any client modules.

A number of actions can be performed on each object such as; instantiating an object which is similar to giving proper names to real world object to be able to distinguish them. In most applications this process is related with the external part of an object independent from its internal relations and properties. As a consequence, there

might be any number of instances of otherwise identical objects. For example an object 'fruit' can have an instance such as 'apple', which can be said as apple is an instance of object fruit. Similarly, orange can be an instance of object fruit. Another property is changeability of object. Since the identity of an object is independent of its internal properties, an object can have a set of properties at one time and a different set another time. For example, if the object fruit had a color variable which was 'red' at one time, the apple instance can have its internal color variable modified as 'green' later on. Still another related notion is sharing of object which is a direct consequence of object being able to have various instances.

In systems that are completely object-oriented there exists a strong degree of homogeneity which implies that every element of the system is an object. Thus, the system treats every item a procedure, file, program etc. as an object which is both active and persistent.

3.3. CLASSES

A definition of object-oriented design can be given as the construction of software systems as structured collections of abstract data type implementations. This definition brings along the issue of classes which is a collection of a whole set of related objects. The classes in OOP must be formed as units which are interesting and useful on their own, independent of the systems which they belong. Each object that belong to a class is called the instance of that class. In OOP every object is an instance of a class. The class concept represents the similarities of its instances and each instance contains some information particular to itself which distinguish it from other instances. This information is a subset of its private part variables called the *instance variables*. All instances of a class possess the same number of instance variables, but the values of each variable differ from an instance to another.Similarly, there are certain variables of the object's private part that are shared by all other instances of its class. Such variables are known as *class variables* and belong to that class. Naturally, there are important

relationship between classes such as the client and descendant relationship.

* A class is a client of another when it uses the other class' services.

* A class is a descendant of one or more other ones when it is designed as an extension or specialization of these classes.

The class concept is a primary element of OOP and has many uses and applications that can be listed as;

- Generating new objects.
- Describing the representation of instance.
- A tool that facilitates differential programming.
- Serve as locations for methods for receiving messages.
- A method that allows updating numerous objects simultaneously and dynamically.
- A group of all instances of a class.
- In a running system it describes how objects behave in response to messages.
- In systems under development serve as an interface for the programmer to interact with the definition of objects.

Finally, as a summary OO systems are built as collections of classes with each class representing a particular abstract data type implementation. Thus, classes should be designed as general and reusable as possible since the process of combining them into systems is often bottom up.

3.4. MESSAGES

The objects communicate to one another in order to fulfill their required task using *messages*. Generally, a message is a specification of one of an object's manipulations; such as requests to access, modify or return a part of its private part.

When a message is received by an object, it determines how to react. The object that reacts is called the receiver of the message.

Mostly the message includes a symbolic name that indicates the desired reaction. This name is known as the *message selector*. The distinguishing feature of messages is that the selector not only shows the desired reaction but also describes what the programmer wants to happen and how it should happen. The receiver object knows how to respond to the request. This process of invoking a particular manipulation is similar to procedure calling, but differs in the sense that in OOP a message can be interpreted differently by different receivers. A set of messages to which any given object will respond is termed as the *message protocol* for that object. OOP technique employs a message-passing paradigm as a model for object communication. In this scenario, as mentioned in the previous paragraph, an object is not allowed to "to operate" on another object but it can only invoke a manipulation in that object. This structure allows *independence* of objects by leaving the interpretation of the message to the internal rules of the receiver.

Message-passing in OOP can either be synchronous in which the sender blocks until the message is delivered or asynchronous in which the message is put on a queue and the sender is free to perform another task. From the object independence point of view asynchronous message passing seems preferable but it is not a pre-requisite of an O-O system.

3.5. METHODS

Objects use *methods* to define their behavioural actions. A method is a procedure-like entity that describe either a single type of manipulation of an object or a sequence of actions to be performed by the processor. The distinguishing feature of methods is that a method can not call another method and they can not be separated from object.

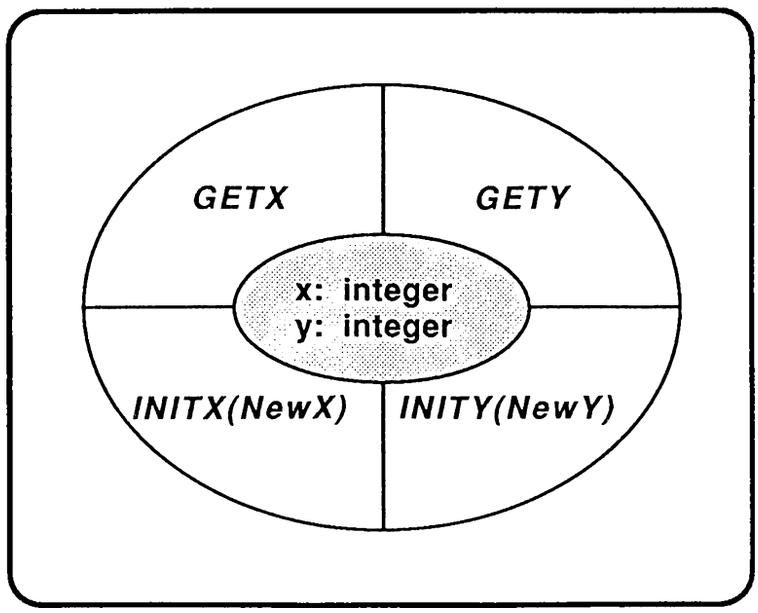

Fig.8. An object encapsulated by its message protocol.

An object has both private variables and a set of methods that describe what to do when a message is received. When a message is received by an object, the values of its private variables serve as data and methods serve as procedures. This segregation of data and procedures is totally localized to the private part of the object. Like procedures, methods are required to know about the form of the data they manipulate.

Lets analyze the mechanics of communication among objects. When a message is sent to an object, a message search is initiated to determine the corresponding method. In most cases, this search starts from the class of the receiver object. If search is successful then the method is executed, otherwise, look-up continues in a super class of the receiver class. This procedure continues through the class hierarchy until the appropriate method is found or root is reached in which case an error occurs. The message passing is usually implemented as function calls or in certain applications remote procedure calls.

Since methods of an object are the only tools that can be used to alter an object, O-O system, form a natural barrier around the object where methods control any undesired access and changes to these reserves. It can also be said that the method-message couple add data abstraction property to O-O programming (Fig.8).

3.6. ATTRIBUTES OF OOP

The basic attributes of OOP approach is a result of the combination of the basic elements like the object, class, messages, method of OOP. These attributes institute a special notion to this technique. These characteristics are:

Data Abstraction

Independence

Inheritance

Message Passing

Polymorphism

Homogeneity

Dynamic Binding

3.6.1. DATA ABSTRACTION

This is by far the most important concept in OOP approach. The main interest of this technique is focused on the behaviour of an object rather than its representation [15] [20]. An abstract type consist of an external interface which contain a set of procedures (methods) used to access and manipulate the data and an internal representation. Since objects are the main building blocks of OOP and by definition they possess the state and the behaviour simultaneously, the O-O system supports data abstraction. Furthermore, the class concept and message passing paradigms assist by providing data

and procedure abstraction consecutively to assure overall abstraction or information hiding. In other words, an O-O system constructs a protective barrier around an object where its methods prohibit any unwanted accesses or changes to its internal data.

Data abstraction is a valuable tool that allows the compositions of large systems into smaller encapsulated subsystems that are relatively easier to handle. Besides, this property ensures software reliability and modifiability by reducing interdependencies between software components. Yet another benefit is providing the programmer enough freedom to employ high levels of abstraction and encouraging the composition of the complex problem into collections of cooperating objects with different complexity levels.

3.6.2. INDEPENDENCE

There are primarily two different notions of independence in OOP approach [15] [20] [26]. The first one is object independence which is a consequence of objects possessing control over their own states. Once established, an object will continue to exist even if its creator dies. Thus, persistent objects eliminate the need for files which makes a system more independent. The second notion is the independence of the system to add or create new object types during run time. This capability is especially important if O-O environment is also the development environment eliminating the need of creating new types outside the actual system and thus making it more independent.

3.6.3. INHERITANCE

Inheritance in OOP is the ability to acquire structural and behavioural information from certain other objects in the object universe [14] [15] [20] [22]. This property allows programmers to create classes that enable specializations of objects to be established. A specialized object inherits the properties of its parent type and is free to add more properties if it is desired. Creating a specialization of an existing class is called subclassing and the existing class is then termed as the super class. More specifically the subclass inherits the instance

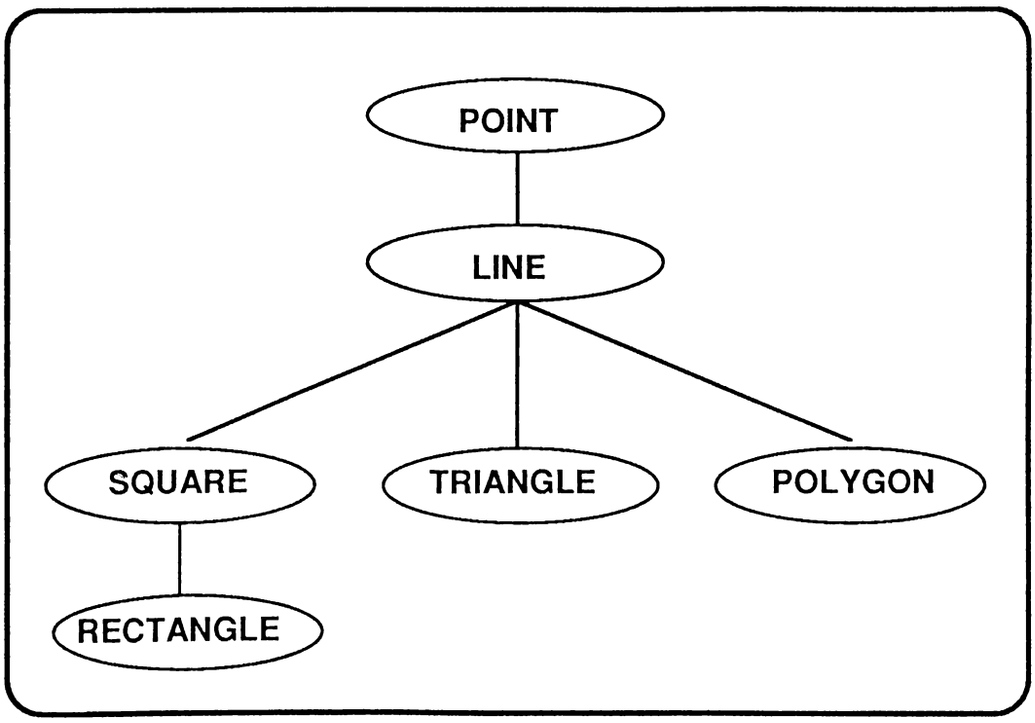

Fig.9 An object hierarchy

variables, class variables, and methods from its super class but it can add or override this inherited properties to provide a more specialized object (Fig. 9). This property structures the object within an object universe into acyclic directed graph which is known as the inheritance hierarchy. Moving down within this hierarchy it can be observed that each subclass has a special characteristic which distinguish it from its super class.

The mechanics of inheritance is straightforward; when a class object receives a message it first scans through its class methods to determine whether it can satisfy the request. If it can not, it passes the message upward through the inheritance hierarchy searching for the suitable method. This search continues until the top class or until a suitable method is found. If no method is found then an unknown message error occurs. It is always possible to override or redefine methods and instance variables during inheritance. This process is known as *subtraction* and *addition* consecutively. Inheritance constitutes flexibility and reusability to OOP.

3.6.4. MESSAGE PASSING

The message passing paradigm that is used for object communication in OOP is another characteristic that supports independence of objects [15] [26]. This model had been previously discussed in detail so its sufficient to realize that it is not an implementation requirement but a tool that attaches advantageous characteristics to the technique.

3.6.5. POLYMORPHISM

Polymorphism is a Greek word which means "many shapes" [14] [16] [20] [22] [26]. This concept can be defined as a way of giving an action a name that is shared up and down an object hierarchy with each object implementing the action according to a polymorphic function. During this process each object is sent the same message selector. But this selector can elicit a different response on the receiver object. In general polymorphism can occur in two ways:

- Same operation preserves its behaviour for different arguments.

- Two operations have the same name but behave completely different.

In OOP class inheritance is closely related to polymorphism and widely used to pass common properties of superclass to subclasses and creating generic subclasses by parametizing the unknowns.

This property allows programmers to use the same name when requesting different implementations of a given operation, allows for more readable code, extending the flexibility and reusability of the software.

3.6.6. HOMOGENEITY

In O-O systems homogeneity refers to "every thing" within the universe being an object [15] [17]. The degree of homogeneity within a system can vary depending on whether elements like active objects,

object types, messages, classes are objects or not. If homogeneity is kept at a high level the environment is rather consistent and interpreted code can be totally analysed in terms of communicating objects.

3.6.7. DYNAMIC BINDING

In most conventional languages the code is bound to a specific name at compilation and to a specific address at link time, called early binding [14] [20] [26]. On the other hand, in O-O systems any object identifier can assume any object identity and object can receive any message at any time which is known as dynamic or late binding. The messages are only evaluated when it is actually sent and thus a priori decisions about which objects will be invoked can be avoided during initial development phases. So crucial design decisions do not have to be made in the early design process.

Dynamic binding uses previously defined concepts of polymorphism, data abstraction and inheritance to allow systems that are highly resistant to change to be built. Dynamic binding is usually applied to unstable environment where initial bindings need to be changed after they have been established.

3.7. CONCLUSION

When the basic elements of OOP like object, classes, messages are combined with characteristics such as inheritance, polymorphism, data abstraction, etc., the resulting technique can provide external software qualities such as;

- Maintainability
- Robustness
- Reliability
- Reusability
- Portability

- Integrity
- Ease of use

Obviously, above listed qualifications offer significant advantages to OOP approach. But it should be kept in mind that in order to utilize these factors to high extent, object oriented concepts need to be thoroughly understood and well analyzed.

4. IMPLEMENTATION

This chapter provides a detailed analysis of the implemented system beginning with an overview and covering the modules of the system. The modules are explained by giving segments from the source code supported by definitions of their internal representations and methods. Another topic discussed is the methods employed for nondeterministic generation of examples and questions.

4.1. AN OVERVIEW OF THE SET THEORY TUTOR

The ITS developed within the scope of this work aims at teaching *Set Theory* to secondary school students. The system was implemented using the object-oriented Turbo Pascal (version 5.5) programming language. It runs on DOS and MS-DOS operating systems on all models of IBM PC compatibles and IBM PS/2. The system is implemented using object-oriented programming approach, which improves overall homogeneity of the system by employing objects and classes of related objects.

The design and implementation stages of the system have been successfully completed and the resulting ITS performs teaching of the entire contents of Set Theory while generating all questions and examples on a random basis. To be more specific, the system generates *different* examples and questions each time it is invoked. On the other hand, the system is also able to produce both *true/false* and *multiple choice* types of questions, trying to select tricky and confusing options for the second type.

The framework of the system is formed by classes under which objects are collected. Also every element of the system is an object which contributes an overall homogeneity to the system.

There are certain auxiliary inputs that the system required from the teacher; this information comprises the teaching approach that the teacher would like to administer. These are;

- Selection of **universes** for the entire system.
- Selection of the **universe** for the questions and examples.
- For each topic:

    * Text explaining the topic,
    * Number of questions to be generated,
    * Number of examples to be generated.

After feeding the system with the data listed above it is ready for operation. One point that should be kept in mind is that the system's objective is teaching Set Theory to the user and evaluating his/her performance. Thus, a history file is not kept to record the performance of the user.

4.2. THE MODULES OF THE SET THEORY TUTOR

As discussed in the previous sections most ITSs comprise three main modules namely ;

1. Expert module
2. Student model
3. Teaching module

Similarly, the ITS implemented incorporates the three basic modules (Fig. 10). The following sections will discuss, in detail, the structures and attributes of these modules of the actual system.

4.2.1. EXPERT MODULE

The expertise module is a control center which encompasses the entire domain knowledge, generates instructional content and evaluates the student's performance. Although, in some systems, this module is made up two sections , the *knowledge base* and the *bug catalogue* , in the system implemented in this work only the knowledge base is used ignoring the bug catalogue.

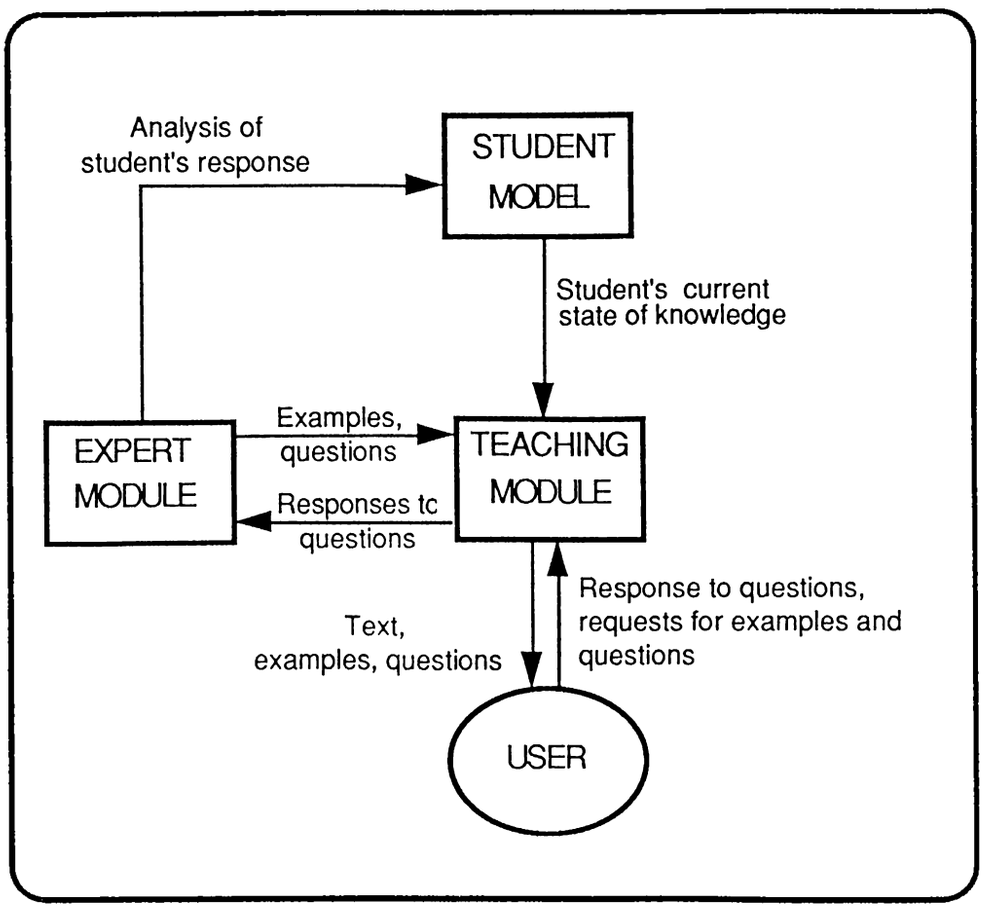

Fig.10 Structure of the implemented system

In the set theory tutor all the knowledge is hard-coded in a procedural representation scheme. The core of Set Theory is the concept of *Set* which is the starting point of introducing the theory to the computer. For the program to yield any positive result it should know *what* a set is and *how* it can be generated. To achieve this an object "setobj" is defined to represent a set that contains two data elements: universe and value, which point to a universe linked list and a set linked list respectively (Fig. 11.). Furthermore, some additional methods are defined such as; *default* to initialize the set with default values, *cardinality* to find the cardinality of set, *nthelement* to find the nth element of a set, *initUniv* to set the universe of a set, *shuffle* to change the sequence of set elements.

```
setobj = object
     universe : universePtr;
     value : setType;

     constructor default;
     procedure inituniv(univ : universePtr);
     function cardinality : byte;
     function nthElement (n : byte) : WordType;
     procedure shuffle;
     procedure SetCopy (OriginalSet : setobj);
          {Copies OriginalSet }
     function SelectElt : WordType;
          {Randomly returns an element from the set}
     function IsEmpty : boolean;
     procedure delfromSet (elt : WordType);
          {Delete an element from that set}
     procedure addtoSet (elt : WordType);
          {Add an element to the set}
end ; {setobj}
```

Fig.11 Definition of "setobj"

The "setobj" represents a basic set. But in order to generate a set with the desired requirements many additional parameters are needed. For example, it could be the case that a set with certain elements is required to be generated or a set with a fix number of elements is needed. Therefore, a new object "ex_set" is defined on the object "setobj" which is insufficient on its own (Fig. 12).

```
ex_set = object (setobj)
     min, max : byte;
          {determines the cardinality of set}
     incl, excl: setobj;
          {the elements to be included and excluded}
```

```
    constructor default;
        {sets default values}
    procedure initMin (newMin:byte);
        {sets minimum value}
    procedure initMax (newMax:byte);
        {sets maximum value}
    procedure initIncl (newIncl:setobj);
    procedure initExcl (newExcl:setobj);
    procedure copy (orgset:ex_set);
        {copies orgset}
    procedure make;
    function equal(otherset:ex_set):boolean;
end; {ex_set}
```

Fig.12 Definition of "ex_set"

The make method of object "ex_set" is as follows(Fig. 13)

```
procedure ex_set.Make;
var
    TmpUniv : setobj;
    anelt :WordType;
    i : shortint;
    more : shortint;
begin
    tmpUniv.default;
    initialize(nil);
    setCopy (incl);{include the desired elements}
    more := min - incl.cardinality + random(max - min + 1);
    TmpUniv.setCopy(universe^.value);{contains all the
        elements of universe}
```

```
    for i := 1 to  incl.cardinality do
              TmpUniv.delfromSet (incl.nthelement (i) );
              {deletes the contents of include from
              TmpUniv}
    for i:=1 to excl.cardinality do
              TmpUniv.delfromSet ( excl.nthelement (i) );
              {deletes the contents of exclude from
                   TmpUniv}
    for i := 1 to more do begin
         anelt := TmpUniv.selectElt;
         addtoSet(anelt);
         TmpUniv.delfromSet(anelt);
         {exclude the selected elements from tmpUniv}
    end;
end; {ex_set.Make}
```

Fig.13 Definition of method "ex_set.Make"

Assume that ASetExample is an instance of "ex_set" that is to be generated from the universe of animals, having 'lion' and 'hawk' as elements excluding 'cat' with a cardinality of maximum 5 minimum 3. The following are the messages sent to object ASetExample and tmp, which is an instance of "setobj", to have the desired example generated (Fig. 14).

```
ASetExample.default;
tmp.default;
tmp.addtoSet('cat');
ASetExample.initExcl(tmp);
tmp.initialize(nil);   {clear tmp}
tmp.addtoSet('lion');
tmp.addtoSet('hawk');
ASetExample.initIncl(tmp);
ASetExample.initUniv(Univ);
ASetExample.initMin(3);
ASetExample.initMax(5);
ASetExample.Make;
```

Fig.14 The source code segment to generate a set example

After initializing the object AsetExample with proper data it will function as described in procedure "make" and generate a set including 'lion' and 'hawk', excluding 'cat' with number of elements between 3 and 5 such as {bear,dog,hawk,lion}.

4.2.1.1.Generation of Examples

As previously mentioned, the system generates examples and questions on each topic of Set Theory. Fortunately, some processes are the same for each topic. So an object called "example" is defined (Fig.15). This object gathers all common features of examples.

```
example  =  object
     universe : UniversePtr;
     x,y : integer;

     constructor default;
          {sets default values}
     procedure inituniv (univ : UniversePtr); virtual;
          {sets the universe of example}
     procedure initxy ( x1,y1 : integer);
          {determines the coordinates of display}
     procedure make; virtual;
          {generates an example}
     procedure show; virtual;
          {displays the example}
end; {example}
```

Fig.15 Definition of "example"

The keyword virtual implies the inherited objects will define this procedure using their own method. Refering to the structure of set theory in Fig.16, there are mainly 3 groups of examples:

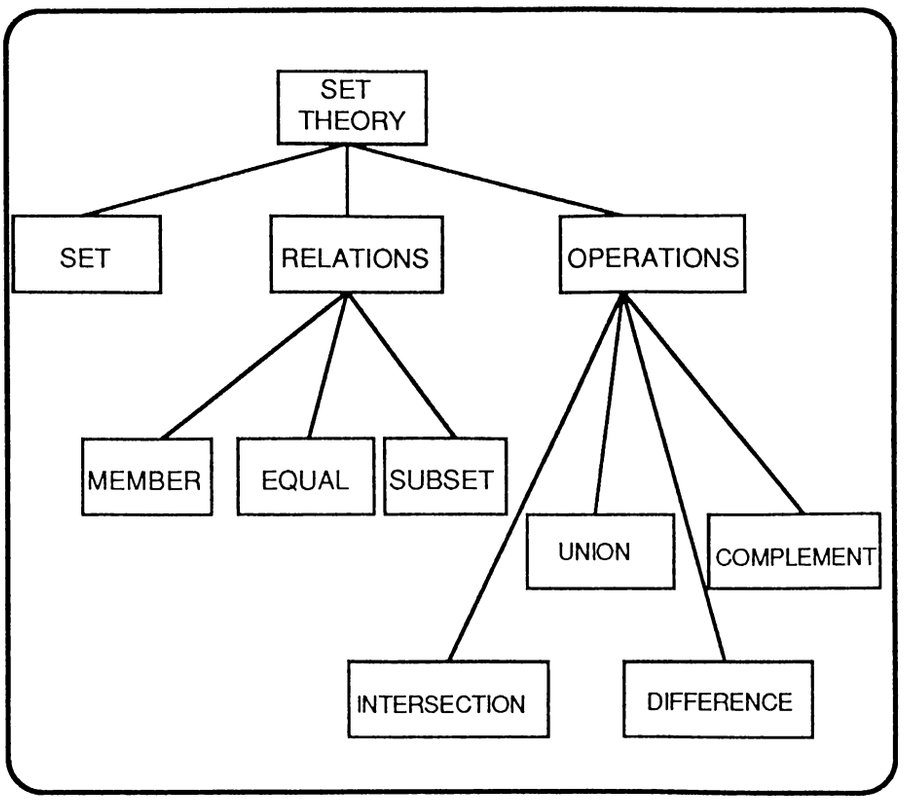

Fig.16 Set theory structure

- Examples for Set
- Examples of Relations
- Examples of Operations

The next step is to define a new object called "ex_sets" to generate the set example (Fig.17).

```
ex_sets   = object (example)
     Aset : ex_set;

     constructor default;
     procedure initUniv (univ : universePtr); virtual;
          {sets universe of example and Aset}
     procedure make; virtual;{generates  Aset}
     procedure show; virtual;{displays the example}
end;
```

Fig.17 Definition of "ex_sets"

Relation	Universe	Element	SetL	SetR
Member	√	√		√
Subset	√		√	√
Equal	√		√	√

Fig.18 Common attributes of relations

Also, the process of generating examples on relations use similar data and procedures. To achieve this, a new object called "ex_relation" for common relation properties can be built(Fig.19).

```
ex_relation = object(example)
    SetR:ex_set;
        {All relations share SetR and universe}

    constructor default;
    procedure initSetR(Aset:ex_set);
    function getUniv:universePtr;
    procedure initUniv (univ: universePtr); virtual;
end; {ex_relation}
```

Fig.19 Definition of "ex_relation"

On the other hand, in order to generate subset examples, the system requires a second set called "setL"(Fig.20). The system also can easily generate an example where SetL is notsubset of SetR. Therefore defining a new object "ex_not_subset" by changing the "make" procedure of "ex_subset" is sufficient, reasonable and easy (Fig.21).

```
ex_subset = object(ex_relation)
     SetL:ex_set;
     constructor default;
     procedure initUniv (univ:universePtr); virtual;
     procedure initSetL (Aset:ex_set);
     procedure make; virtual;
     procedure show; virtual;
end; {ex_subset}
```

Fig.20 Definition of "ex_subset"

```
ex_not_subset = object (ex_subset)
     constructor default;
     procedure make; virtual;
end; {ex_not_subset}
```

Fig.21 Definition of "ex_not_subset"

The objects "ex_equal" and "ex_not_equal" have the same logic as "ex_subset" and "ex_not_subset"(Fig.22).

```
ex_equal = object (ex_relation)
     SetL : ex_set;

     constructor default;
     procedure initUniv(univ:universePtr); virtual;
     procedure initSetL(Aset:ex_set);
     procedure getSetL(var aset:ex_set);
     procedure make;virtual;
     procedure show;virtual;
end; {ex_equal}

ex_not_equal = object (ex_equal)
     constructor default;
     procedure make; virtual;
end; {ex_not_equal}
```

Fig.22 Definition of "ex_equal" and "ex_not_equal"

The object "ex_member" needs an element data in order to generate member examples. Again by changing "make" method of "ex_member", the system can generate examples where an element is not an element of a proper set. This is possible via defining "ex_not_member" object on top of "ex_member"(Fig.23).

```
ex_member = object (ex_relation)
     Anelement : WordType;
     constructor default;
     procedure make; virtual;
     procedure initAnelement ( elt : wordType);
     function getAnelement : wordType;
     procedure show; virtual;
end; {ex_member}

ex_not_member = object (ex_member)
     constructor default;
     procedure make; virtual;
end; {ex_not_member}
```

Fig.23 Definition of "ex_member" and "ex_not_member"

Furthermore, the make method for object "ex_subset" can be defined in Fig.24 .

```
Procedure ex_subset.make;
var
     TmpUniv, tmp: setobj;
     SetLcard,i,j,k,extra: integer;
     anelt: WordType;

begin
     tmpuniv.default;tmp.default;
     if SetL.isEmpty and SetR.isEmpty then begin
          {if SetL and SetR is not settled before}
          SetL.make;{Generate SetL}
          SetR.initIncl(SetL);
          SetR.make;
          {the system is sure that SetL is subset of SetR
               as SetR has all the elements of SetL}
          SetR.shuffle;
          end
     else if SetL.isEmpty then begin
          {If SetR is adjusted before, SetL can be generated
          as having random elements of SetR}
          SetL.copy(SetR);
          for k:=1 to random(SetR.cardinality) do
               SetL.delfromSet(SetR.nthElement(k));
          SetL.shuffle;
          end
          else begin
          {if SetL is adjusted before, then generate SetR
          having elements of SetL}
               SetR.InitIncl(SetL);
               SetR.make;
               SetR.shuffle;
          end;
end; {ex_subset.Make}
```

Fig.24 Definition of "ex_subset.make"

Operation	Universe	SetL	SetR	Result
Union	√	√	√	√
Intersection	√	√	√	√
Difference	√	√	√	√
Complement	√	√		√

Fig.26 Common attributes of operations

ASubsetExample is an instance of "ex_subset". Assume that a subset example is required with SetL={apple,plum} and within the universe of fruits. The followings are the series of messages sent to ASubsetExample to have the system generate the required example (Fig.25).

```
ASubsetExample.default;
tmp.default;
tmp.addtoSet('apple');
tmp.addtoSet('plum');
     /* tmp is a set including desired SetL elements.*/
ASubsetExample.initSetL(tmp);
ASubsetExample.initUniv(Univ);
ASubsetExample.make;
```

Fig.25 A source code segment to generate subset example

As a result of the above listed messages the system can generate the following example.

{apple,plum} ⊂ {orange,apple,plum,pear}

Up to this point, the way that the system generates relation examples has been demonstrated. Operation example generation employs the same logical steps (Fig. 26)

```
ex_operation = object (example)
     SetL : ex_set;
     SetR : ex_set;
     resultSet : ex_set;

     constructor default;
     procedure initSetL (aset:ex_set);
     procedure initresultSet(aset:ex_set) ;
     procedure initSetR (aset:ex_set);
     procedure show; virtual;
end; {ex_operation}

ex_union = object (ex_operation)
     constructor default;
     procedure make; virtual;
end; {ex_union}

ex_intersection = object (ex_operation)
     constructor default;
     procedure make; virtual;
end; {ex_intersection}

ex_difference = object (ex_operation)
     constructor default;
     procedure make; virtual;
end; {ex_differrence}

ex_complement = object (ex_operation)
     constructor default;
     procedure make; virtual;
     procedure show; virtual;
end; {ex_complement}
```

Fig.27 Definition of examples for operations

The following procedure is the "make" method that generates the intersection examples (Fig.28).

```
Procedure ex_intersection.make;
var
     tempset,tempset2 : ex_set;

procedure randomset( Orgset : ex_set ;var newset : ex_set);
begin
     /* generates newset using elements of Orgset*/
end;
begin
tempset.default;tempset2.default;
     if SetL.isEmpty then
          if SetR.isEmpty then
               if ResultSet.isEmpty then begin
                    SetL.make;
                    SetR.make;
                    intersection(SetL,SetR,ResultSet);
                    end
               else begin
                    SetL.initIncl(ResultSet);
                    SetL.make;
                    SetL.shuffle;
                    tempset.initialize(empty);
                    difference(SetL,ResultSet,tempset);
                    SetR.initIncl(ResultSet);
                    SetR.initExcl(tempset);
                    SetR.make;SetR.shuffle;
                    end
          else
               if ResultSet.isEmpty then begin
                    randomset(SetR,tempset);
                    SetL.initIncl(tempset);
                    SetL.make;
                    SetL.shuffle;
                    intersection(SetL,SetR,ResultSet);
                    end
               else  begin
                    tempset.initialize(empty);
```

```
                tempset2.initialize(empty);
                intersection(SetR,ResultSet,tempset);
                difference(SetR,ResultSet,tempset2);
                SetL.initIncl(tempset);
                SetL.initExcl(tempset2);
                SetL.make;SetL.shuffle;
                end
    else
        if SetR.isEmpty then
            if ResultSet.isEmpty then begin
                randomset(SetL,tempset);
                SetR.initIncl(tempset);
                SetR.make;
                SetR.shuffle;
                intersection(SetL,SetR,ResultSet);
                end
            else begin
                tempset.initialize(empty);
                tempset2.initialize(empty);
                intersection(SetL,ResultSet,tempset);
                difference(SetL,ResultSet,tempset2);
                 SetR.initIncl(tempset);
                SetR.initExcl(tempset2);
                SetR.make;
                end
        else
            if ResultSet.isEmpty then
                intersection(SetL,SetR,ResultSet)
            else ; {nothing is needed}
    tempset.clear;tempset2.clear;
end; {ex_intersection.Make}
```

Fig.28 Definition of "ex_intersection_make"

AnIntersectionExample is an instance of "ex_intersection". If an intersection example using the digits universe is going to be generated the procedure will be as follows (Fig.29):

```
AnIntersectionExample.default;
AnIntersectionExample.initUniv(univ);
AnIntersectionExample.make;
```

Fig.29 A source code segment to generate intersection example

And a possible generation can be as follows;

$\{1,3,7,2\} \cap \{5,7,3,1\}=\{1,3,7\}$.

The tree below will provide a brief explanation of class of examples. It should be noticed that Fig.30 below is nearly the same as the set theory structure given in Fig.16.

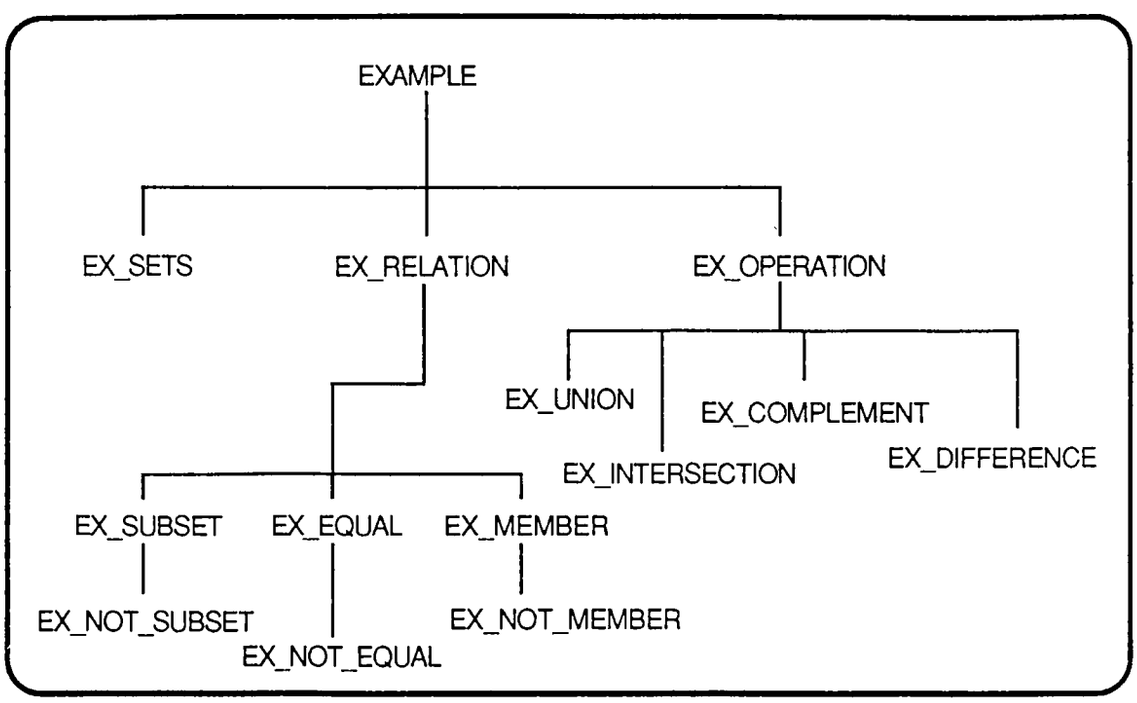

Fig.30 Class of objects generating examples

4.2.1.2. Generation of Questions

After the system has the capability to generate examples, the next step is the generation of questions. Questions are obtained by changing some parts of the examples. Therefore, the system uses example objects in order to generate questions. Again all features of questions are gathered under the "question" object (Fig.31).

```
question = object
     booleanAns : boolean;
     multAns : char;
     queType : shortint;
     noofanswers : integer;
     x,y : integer;

     constructor default;
     procedure initxy (x1,y1 : integer);
     function  getbooleanAns : boolean ;
     procedure setmultAns (a : char );
     procedure setqueType (i : integer);
     procedure makequestion;
     procedure ask;
     procedure ynQueMake;virtual;
     procedure multQueMake;virtual;
     procedure ynQueShow;virtual;
     procedure multQueShow;virtual;
end; {question}
```

Fig.31 Definition of "question"

The choices of questions are kept in an array. In order to generate member questions, choices are kept as elements where in remaining topics choices are sets. Therefore, two objects defined upon "question" as "questionA" and "questionB" ,where questionA has choices as array of sets and questionB has choices as array of elements (Fig.32).

```
que_set = object (question)
     Set1 : ex_sets;

     constructor default;
     procedure ynQueMake ; virtual;
     procedure ynQueShow ; virtual;
     procedure Explain(methodnum:integer; answer,
                    correct : char);
end;
que_subset = object (questionA)
     subset : ex_subset;

     constructor default;
     procedure ynQueMake;virtual;
     procedure multQueMake;virtual;
     procedure ynQueShow;virtual;
     procedure multQueShow;virtual;
     procedure Explain(methodnum:integer; answer,
                        correct : char);
end; {que_subset}

que_member = object (questionB)
     member : ex_member;

     constructor default;
     procedure ynQueMake; virtual;
     procedure multQueMake;virtual;
     procedure ynQueShow; virtual;
     procedure multQueShow;virtual;
     procedure Explain(methodnum:integer; answer,
                    correct : char);
end; {que_member}

que_equal = object (questionA)
     equal : ex_equal;
```

```
    constructor default;
    procedure ynQueMake; virtual;
    procedure multQueMake;virtual;
    procedure ynQueShow; virtual;
    procedure multQueShow;virtual;
    procedure Explain(methodnum:integer; answer,
                  correct : char);
end; {que_equal}

que_union = object (questionA)
    union1 : ex_union;

    constructor default;
    procedure ynQueMake; virtual;
    procedure multQueMake;virtual;
    procedure ynQueShow; virtual;
    procedure multQueShow;virtual;
    procedure Explain(methodnum:integer; answer,
                  correct : char);
end; {que_union}

que_intersection = object (questionA)
    intersection1 : ex_intersection;

    constructor default;
    procedure ynQueMake; virtual;
    procedure multQueMake;virtual;
    procedure ynQueShow; virtual;
    procedure multQueShow;virtual;
    procedure Explain(methodnum:integer; answer,
                  correct : char);
end; {que_intersection}

que_difference = object (questionA)
    difference1 : ex_difference;
```

```
    constructor default;
    procedure ynQueMake; virtual;
    procedure multQueMake;virtual;
    procedure ynQueShow; virtual;
    procedure multQueShow;virtual;
    procedure Explain(methodnum:integer; answer,
                  correct : char);
end; {que_difference}

que_complement = object (questionA)
    complement1 : ex_complement;

    constructor default;
    procedure ynQueMake; virtual;
    procedure multQueMake;virtual;
    procedure ynQueShow; virtual;
    procedure multQueShow;virtual;
    procedure Explain(methodnum:integer; answer,
                  correct : char);
end;{que_complement}
```

Fig.32 Definition of all question generating objecs

The tree in Fig.33 gives a brief explanation of class of questions. After examining the object hierarchy of question generation process a number of examples on the topic will be demonstrated supplying the related "make" procedures in relation "member" (Fig.34) and operation "difference" (Fig.36).

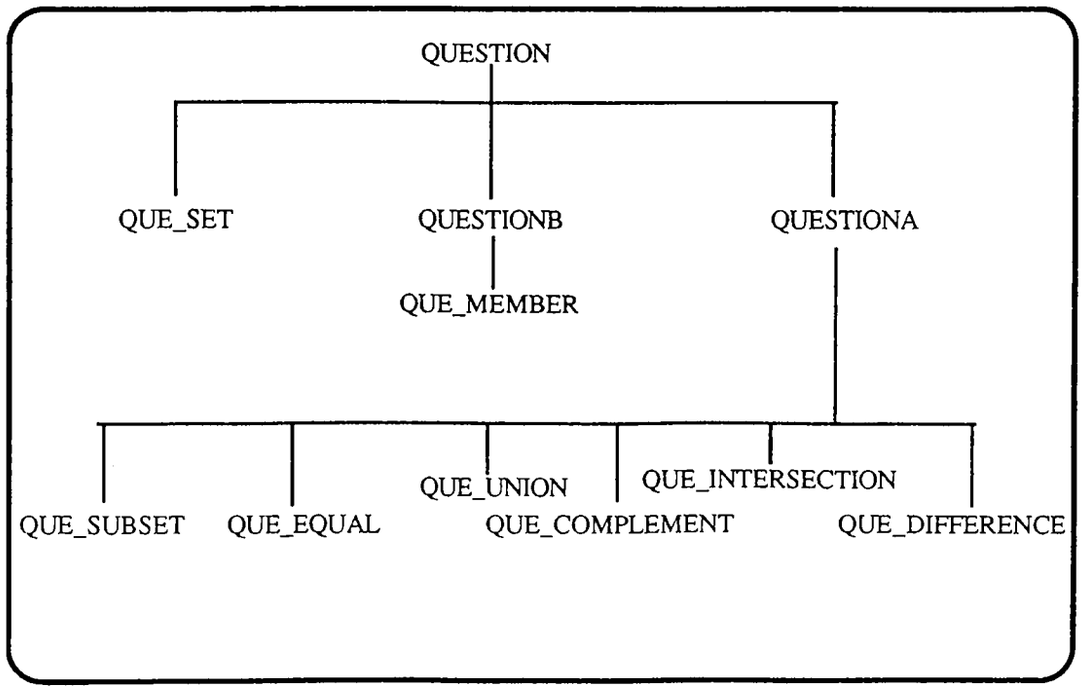

Fig.33 The class of objects generating questions

```
procedure que_member.ynQueMake;
var
     tempset : ex_set;
begin
     tempset.default;
     member.make;
     if booleanAns = false then begin
          complement(member.SetR,tempset);
          member.initAnelement(tempset.selectElt);
     end;
end; {que_member.ynQueMake}
```

```
procedure que_member.multQueMake;
var
     i,j : integer;
     tempelt : WordType;
     sorusayisi,trycount : integer;

begin
     member.make;
     tempelt:=member.getAnelement;
     addtoAnswers(tempelt,'x');
     multAns:='a';
     notmemberex.default;
     notmemberex.SetR.copy (member.setR) ;
     notmemberex.SetR.setnotvalued(false);
     notmemberex.inituniv(member.getuniv);
     sorusayisi := noofanswers;
     trycount:=0;
     for i:=1 to sorusayisi do begin
          notmemberex.make;
          tempelt:=notmemberex.getAnelement;
          if not(inanswers(tempelt)) then
               addtoAnswers(tempelt,'y')
          else begin
               i:=i-1;
               trycount :=  trycount+1;
               if trycount > 10 then i:=sorusayisi;
               end;
          notmemberex.initAnelement('');
     end;
     notmemberex.clear;
     shuffleanswers;
end; {que_member.multQueMake}
```

Fig.34 Definition of "que_member.ynQuemake" and "que_member.multQueMake"

The object AMemberQuestion is an instance of "que_member". If a multiple choice question is to be generated the procedure will be as follows:

```
AMemberQuestion.default;
AMemberQuestion.setQueType(2);
AMemberQuestion.Make;
```

Fig.35 A source code segment to generate member question

The universe of the question will be randomly determined within the method "default". Assuming that the universe of courses is randomly selected a typical question will look like:

A={math,physics,chemistry}

Then which one of the following is a member of A?

a) english

b) turkish

c) physics

d) history

Past experience has shown that students have a tendency to confuse operations union, intersection and difference. Besides, A \ B and B \ A are also often confused. Therefore, this information is used to generate difficult choices for multiple choice questions. The following is a method to generate a difference question.

```
procedure que_difference.multQueMake;
var
    i,j : integer;
    tempSet : ex_set;
    sorusayisi,trycount : integer;
```

```
begin
methodused := 3;
tempSet.default;notEqualEx.default;
difference1.make;
tempSet.copy(difference1.ResultSet);
addtoAnswers(tempSet,'x');
multAns:='a';
tempSet.clear;
sorusayisi := noofanswers;
union(difference1.SetL,difference1.SetR,tempset);
if not(inanswers(tempset)) then begin
     addtoAnswers(tempSet,'u');
     sorusayisi:=sorusayisi-1;
     end;
tempSet.clear;
Difference(difference1.SetR,difference1.SetL,tempset);
if not(inanswers(tempset)) then begin
     addtoAnswers(tempSet,'d');
     sorusayisi:=sorusayisi-1;
     end;
tempSet.clear;
Intersection(difference1.SetL,difference1.SetR,tempset);
if not(inanswers(tempset)) then begin
     addtoAnswers(tempSet,'i');
     sorusayisi:=sorusayisi-1;
     end;
tempSet.clear;
notEqualEx.SetR.copy(difference1.ResultSet);
notEqualEx.SetR.setnotvalued(false);
notEqualEx.inituniv(difference1.universe);
trycount := 0;
for i:=1 to sorusayisi do begin
     notEqualEx.make;
     tempSet.copy(notEqualEx.SetL);
     if not(inanswers(tempset)) then
          addtoAnswers(tempset,'y')
```

```
        else begin
             i:=i-1; trycount:=trycount+1;
             if trycount > 10 then i:=sorusayisi;
             end;
        notEqualEx.SetL.clear;
        tempSet.clear;
    end;
    notEqualEx.clear;
    shuffleanswers;
    end; {que_difference.multQueMake}
```

Fig.36 Definition of "que_difference.multQueMake"

The object ADifferenceQuestion is an instance of object "que_difference". The procedure to develop difference questions is given in Fig.37.

```
ADifferenceQuestion.default;
ADifferenceQuestion.setQueType(2);
ADifferenceQuestion.Make;
```

Fig.37 A source code segment to generate difference question

A possible multiple choice question from the universe of letters can be formulated as :

A={a,b,c,d,e}

B={c,d,f,g,h}

Then what is A \ B ?

a) {c,d}

b) {a,b,c,d,e,f,g,h}

c) {f,g,h}

d) {a,b,c}

e) {a,b,e}

Up to this point, the methods that the system uses for generating the instructional content has been demonstrated including examples. Yet another function of the *expert module* is evaluating the performance of the student. This task is achieved by comparing the answer that the teaching module passes to the expert module to the *correct* answer. The correct answer is known by the expert module since the question is generated by this module. The result of the comparison process is then sent to the student module that updates itself accordingly.

4.2.2. THE STUDENT MODEL

The function of the *student model* is to determine the student's current state of knowledge, his/her misconceptions and reasoning strategies. The system that was implemented uses this module for determining the current knowledge state of the student. The student's misconceptions are determined in the final examination phase. The reason behind not performing the reasoning earlier is that the bug catalogue is not included within the framework of this system.

In some cases, certain subjects can be divided into subtopics that bring a flexibility to the teaching approach that could be used to instruct it. Set Theory is a good example of such a subject. If set theory is carefully analyzed it could seen that it can be broken down into independent subgroups. This capability allows the topic to be approached using a well structured point of view. Set theory can be represented in a *tree structure* that makes it highly viable for object oriented programming (Fig.16).

The program treats each node as an independent topic (actually subtopic) and teaches each separately. The main logic behind the operation of the program is that for a tutorial session to be concluded the question section must be successfully completed. Therefore, a threshold level must be exceeded, for each node of the set theory tree, for the entire tutorial to be successful. If the level of the user is not

sufficient then the last tutorial about a node is repeated until the threshold is passed. The teaching loop employed by the system is given below.

```
        SHOW RELATED TEXT
LOOP:   GIVE EXAMPLES
        ASK QUESTIONS
        CHECK USER LEVEL
        IF NOT SUFFICIENT GO TO LOOP
```

The check user level step in the above pseudo code examines the student model and extracts the user's current success from that model. Then the system compares the current level to the desired level and decides on whether to repeat or complete the session.

Another important role that the student model has is towards the end of the tutorial of the entire set theory. The sequence that the system follows during this session is to teach the definition of a set, then the relations within set theory and finally the operations of set theory. It is known that the most confusing concepts within set theory are the operations and most novice learners seem to confuse these operations. Therefore, after teaching all operations in set theory the system conducts a final test by asking mixed question on all operations. This combined final examination is different from the previous tests that contained only a single topic. Up to this point the user faced questions about a single topic whereas now a variety of questions must be handled that monitor the knowledge level of the student. If during this final stage a topic is determined to be misconcepted then the tutorial about that topic is repeated.

4.2.2.1.Analysis of Student's responses

To be able to model the misconceptions of the student the system keeps an array containing information about the knowledge level of the user on four fundamental operations of set theory. The initial state of the elements of the array are zero.

As the questions are asked one by one, the system determines the operations used within that question and decreases the relative array element by one if the answer is wrong and does not provide any hints about the misconception. If the wrong answer provides any hints about the misconception of the student such as confusion between two operations like intersection and difference then the confused operations' array elements are decreased by one each.

For example, let A = {cat, lion, bee, tiger}, B = {lion, hawk, bee}, and C = {eagle, bee}. The student is asked to determine the value of $((A \cap B) \cup C)$.

The correct value is {lion, bee, eagle, horse}. Let us suppose the student entered the set {cat, tiger, eagle, horse}. In this case, the program assumes that the student has confused the operations $\cap$ and \, since the set entered by the student is actually the value of the expression $((A \setminus B) \cup C)$. Therefore, it decrements the scores of $\cap$ and \ by one. If the program cannot determine any possible misconception, then the operations in the question are assumed to be misunderstood. This technique is to some extend similar to the bug catalogue used in the SEDAF project [2].

This tracking procedure is repeated for each question and at the end of each the final exam the topics that require re-teaching are selected starting from the most negative array element up to the last element below threshold level. This approach allows the system to determine the misconceptions of the student to a certain extent.

4.2.3. THE TEACHING MODULE

The *teaching module* is a system component that decides what to do next and how to do it during a session. Thus, the control strategy of the intelligent tutoring system is embedded into this module. The *control strategy* or the *flow trend* of an ITS is a very important issue that is mainly made up of two effective factors. These factors are the system and the user. In certain systems the control belongs entirely

to either the user or the system. *Logo* is a good example of user controlled system. The advantage of such systems is that the user is *free* to explore the subject on his own and thus learns the topic well. But in many cases the user can get stuck or lost and spend a lot of time losing his/her motivation. On the other hand, if the system possesses the total control of the flow of the session the tutorial is constrained from exploration and behaves more like an *electronic book* missing the intelligent approach a human tutor would take.

Probably the best control strategy is a combination of two distinct approaches inheriting the advantages of both. The sharing of the control power in these systems will form an environment that is both open to self exploration and can provide some assistance when needed. The drawback of this hybrid strategy that it requires more knowledge to be input to the system.

After this brief discussion about control strategies in ITSs a more detailed analysis of the topic in the implemented system will be performed in the following section.

There are four elements effecting the control of the implemented system determining the next step and its contents;

1. Teacher.
2. Student.
3. Student model.
4. Nondeterminism.

The effect of each element on the flow of the system will be discussed first and their combinations will be considered later.

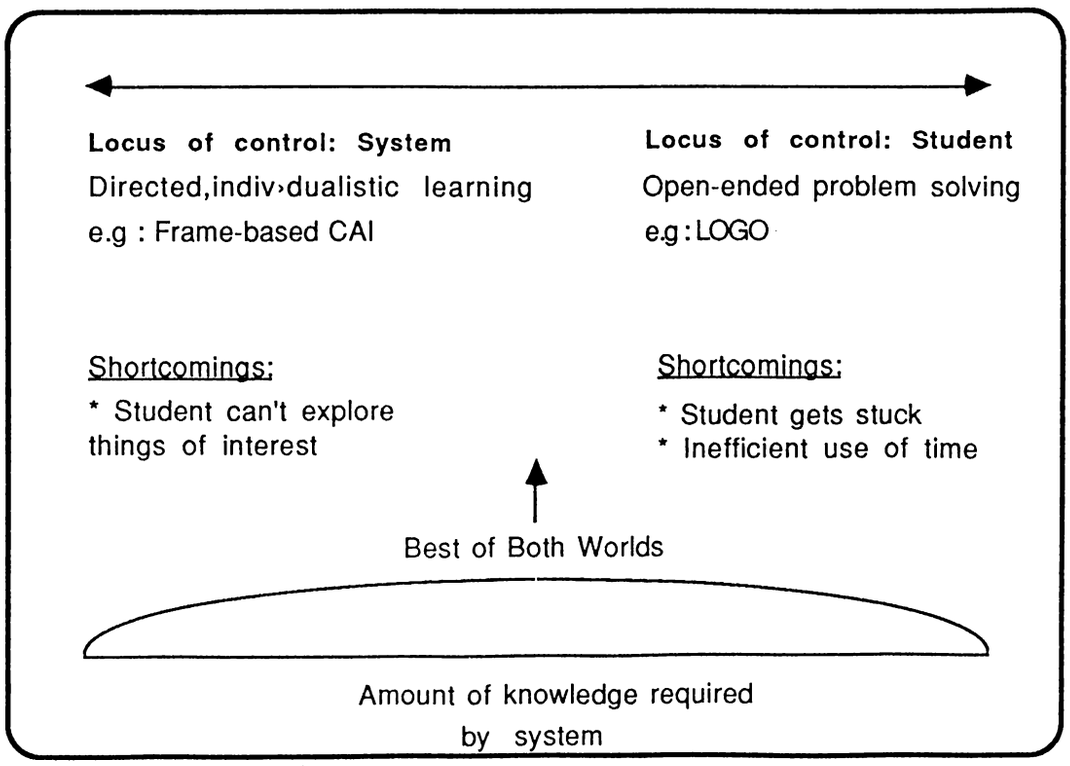

Fig.38 . Control Strategy in ITS [5]

If the *student* possesses the right to control the flow of the program, he will be able to skip without understanding the subject fully. Besides the program will not be be able monitor the extent of the student's comprehension. Another possibility is the *teacher* having the ability which makes the system no longer an ITS system. It would behave more like an electronic book or traditional CAI system. The third factor which is the *student model* is also not equipped with enough data to conduct an effective control of the session. It is only able to measure the comprehension level of the user at a single subject and lacks the mechanisms to continue to other subjects when one is understood. The fourth factor, *nondeterminism* will lead to a total chaos as it can be understood by its definition.

It is clear that the factors discussed above are individually insufficient. So another approach could be combination of three

elements ignoring the remaining one completely. The first scenario is the *teacher, student model and nondeterminism* combination which will undoubtfully lack an important characteristic of ITS systems that is interaction with the student. The second possibility is the *student, student model and nondeterminism* combination. This scenario will still be insufficient from an organizational point of view since its components will be devoid of the ability to manage the tutorial session. The third alternative is the *teacher, student, non determinism* mixture which will eventually be more like an CAI system that is not able to judge about the user's knowledge level. Finally the last trio is the teacher, student, student model which will have similar shortcomings to the previous scenario in the sense that it would have to generate the same questions and examples each time the program is invoked.

In view of the above discussion, a hybrid system must integrate all four factors (student model, student, teacher, nondeterminism) although in different degrees. Depending on the initial design objective it is necessary to adjust the effectiveness of each component. The implemented system is designed such that each factor is able to effect the flow of the session up to a certain degree. The following section will analyze each factor in detail.

•**Student:** The student is always allowed to initiate a dialogue with the system which effects the flow of the session immediately towards the intention of the student. For example, the student can interrupt a session and ask a question or request an example on a topic. The question menu is in the following form:

PLEASE SELECT THE OPERATION

1) MEMBER

2) EQUAL

3) SUBSET

4) UNION

5) INTERSECTION

6) DIFFERENCE

7) COMPLEMENT

8) QUIT

Performing the selection of operation the student is asked to select the universe to be used. After this stage the student formulates the elements or sets as he wishes and the system replies immediately.

The student can also request an example from any universe at any stage of the program except during the final examination. Another approach that is possible is to have these facilities available in a context-sensitive environment. But this alternative is not implemented since it is believed to limit the capability of the user to compare various operations.

•**Teacher:** The system is designed to assist the teacher and decrease the work load on him. In the implemented system there are two different factors effecting the control of the session from the teacher's point of view. These are the hard coded teaching strategy of the program and the inputs required from the teacher prior to the beginning of a session.The following section discusses the hard coded teaching strategy employed within the implemented system.

Since objects generating examples and questions on each topic of Set Theory have been defined, new object can be formed to teach the topics where first text related to it is shown, then examples are given and finally questions are asked. The teach method of this object will be as follows:

(1) Show related text
(2) Give examples
(3) Ask questions
(4) Evaluate user performance
(5) Depending on the level of comprehension either restart or skip to next topic.

The definition of object "topic" to teach a topic of set theory is as follows (Fig. 39):

```
Topic = object
     Filename : fnamestr;
     numberofexamples : integer;
     numberofquestions :  integer;
     studentModel : integer;
     satisfactory : boolean;


     constructor initialize(dataFile:
     fnamestr;n,m : integer) ;
     procedure Teach ; virtual;
     procedure ShowText ; virtual;
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
     procedure UpdateStudentModel ;
     procedure CheckifSatisfactory;
end ; {topic}
```

Fig.39 Definition of "topic"

The objects "Settopic", "Membertopic", "NotMembertopic", "equaltopic", "notequaltopic", "subsettopic", "notsubsettopic" are all defined upon "topic" object. Their "giveexamples" and "askquestions" methods differ so an object is defined for each topic. The objects "NotSubsetTopic", "NotMemberTopic", "NotEqualTopic" have redefined teach method as they only give examples (Fig.40).

```
SetTopic = object (Topic)
     constructor initialize(dataFile:fnamestr;n,m:integer) ;
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
end ; {MemberTopic}
```

```
MemberTopic = object (Topic)
     constructor initialize(dataFile:fnamestr;n,m:integer);
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
end ; {MemberTopic}

NotMemberTopic = object (Topic)
     constructor initialize(dataFile:fnamestr;n,m:integer) ;
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
     procedure Teach ; virtual;
end ; {NotMemberTopic}

EqualTopic = object (Topic)
     constructor initialize(dataFile:fnamestr;n,m:integer) ;
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
end ; {EqualTopic}

NotEqualTopic = object (Topic)
     constructor initialize(dataFile:fnamestr;n,m:integer) ;
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
     procedure Teach ; virtual;
end ; {NotEqualTopic}

SubsetTopic = object (Topic)
     constructor initialize(dataFile:fnamestr;n,m:integer) ;
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
end ; {SubsetTopic}
```

```
NotSubsetTopic = object (Topic)
     constructor initialize(dataFile:fnamestr;n,m:integer) ;
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
     procedure Teach ; virtual;
end ; {NotSubsetTopic}
```

Fig.40 Some question generating objects

The object "RelationTopic" is defined to teach all relations where it has instances of objects mentioned above (Fig.41), and it invokes these objects one by one by saying teach yourself. Teach method of "relationtopic" is as follows.

Member.teach;
NotMember.teach;
Equal.teach;
NotEqual.teach;
Subset.teach;
NotSubset.teach;

```
RelationTopic = object (Topic)
     Member : MemberTopic;
     NotMember : NotMemberTopic;
     Equal : EqualTopic;
     NotEqual : NotEqualTopic;
     Subset : SubsetTopic;
     NotSubset : NotSubsetTopic;
     constructor initialize (dataFile : fnamestr;
          n, m : integer;
          otherDataFiles : filenameArray; otherns : arrayN;
          otherms : arrayM) ;
     procedure Teach; virtual;
     procedure GiveExamples; virtual;
     procedure AskQuestions; virtual;
end ; {RelationTopic}
```

Fig.41 Definition of "Relationtopic"

Assume that SetRelations is an instance of RelationTopic. So it is enough to write.

```
Setrelations.initialize(parameter list);
Setrelations.teach;
```

Fig. 42. A source code segment for teaching relations

Once the teach method is invoked, it will invoke the teach methods of member, notmember,equal,notequal,subset and notsubset one by one and each will teach itself. Also a mechanism is established for teaching operations. We have objects "unionTopic", "intersectionTopic", "differenceTopic" and "complementTopic" on "topic" object. And again "operationTopic" object has instances of these objects and teach method of "operationTopic" invokes each object teach method one by one (Fig.43). At the end, a final exam is given.

```
OperationTopic.teach:
Union.teach;
intersection.teach;
difference.teach;
complement.teach;
finalexam;
```

Fig.43 Teach method of "OperationTopic"

```
UnionTopic = object (Topic)
     constructor initialize(dataFile:fnamestr;n,m:integer) ;
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
end ; {UnionTopic}
```

```
IntersectionTopic = object (Topic)
     constructor initialize(dataFile:fnamestr;n,m:integer) ;
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
end ; {IntersectionTopic}

DifferenceTopic = object(Topic)
     constructor initialize(dataFile:fnamestr;n,m:integer) ;
     procedure GiveExamples; virtual;
     procedure AskQuestions; virtual;
end ; {DifferenceTopic}

ComplementTopic = object(Topic)
     constructor initialize(dataFile:fnamestr;n,m:integer);
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
end ; {ComplementTopic}

OperationTopic = object(Topic)
     UnionX : UnionTopic;
     IntersectionX : IntersectionTopic;
     DifferenceX : DifferenceTopic;
     ComplementX : ComplementTopic;

     constructor initialize(dataFile:fnamestr;n,m:integer;
          otherDataFiles: filenameArray;otherns: arrayN;
          otherms: arrayM) ;
     procedure Teach ; virtual;
     procedure GiveExamples ; virtual;
     procedure AskQuestions; virtual;
     procedure Finalexam;
end ; {OperationTopic}
```

Fig.44 Some objects teaching operations

Finally, the object "SetTheoryTopic" is defined which has 3 variables; basicset an instance of "settopic", Relation an instance of "relationTopic" and operation as an instance of "operationTopic" (Fig.45).

```
SetTheoryTopic = object (Topic)
    BasicSet : SetTopic;
    Relation : RelationTopic;
    Operation : OperationTopic;

    constructor initialize;
    procedure Teach ; virtual;
    procedure GiveExamples ; virtual;
    procedure AskQuestions; virtual;
end ; {SetTheoryTopic}
```

Fig.45 Definition of "SetTheoryTopic"

The teach method of "SetTheoryTopic" is given in Fig.46.

```
BasicSet.teach;
Relation.teach;
Operation.teach;
```

Fig.46 Teach method of "SetTheoryTopic"

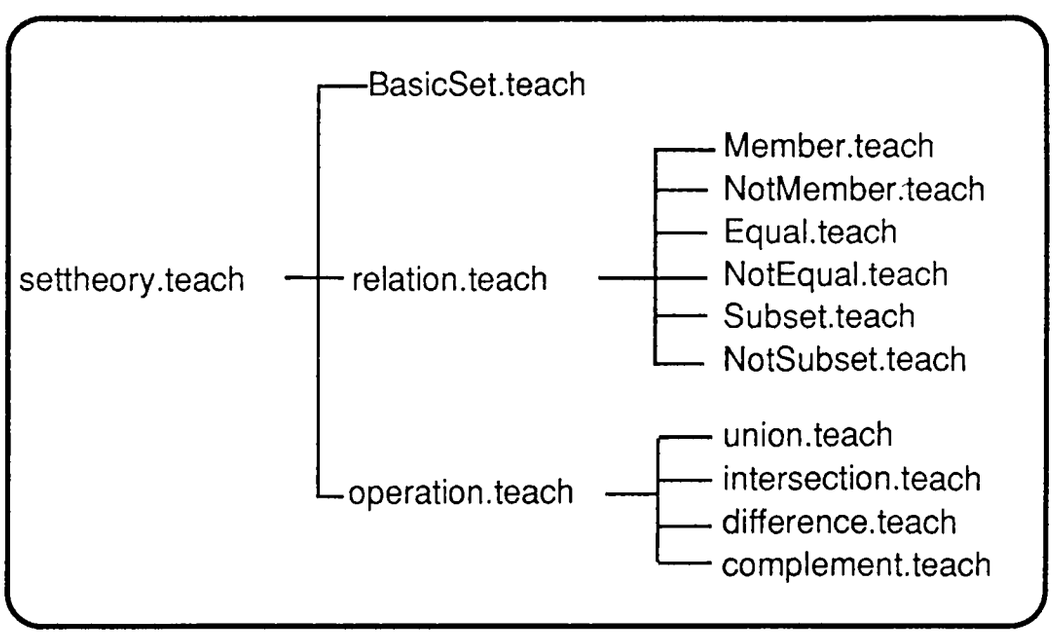

Fig. 47 Mechanism for teaching set theory

Here a streamlined mechanism is established as follows with the help of OOP approach (Fig.47). So in the main program an instance of the object SetTheoryTopic is defined, initialize method is invoked with parameters and teach method is invoked. The last method will teach the entire Set Theory to student (Fig.48).

```
MAIN PROGRAM

program SetTheoryITS;
var
     SetTheory : SetTheoryTopic;
begin
     SetTheory.initailize(parameter list);
     SetTheory.teach;
end;
```

Fig.48 The main program

Object-oriented Programming brought modularity,easy coding in designing such an ITS system as seen above. The only action required from the teacher's side is to input certain parameters that serves as a basis of the session. These are:

-Universes to be used
-Min default value for cardinality
-Max default value for cardinality
-Universe of examples
-Text files for topic
-Number of examples
-Number of questions
-Threshold success level
-Number of questions in final exam
-Threshold score for repetition after final exam

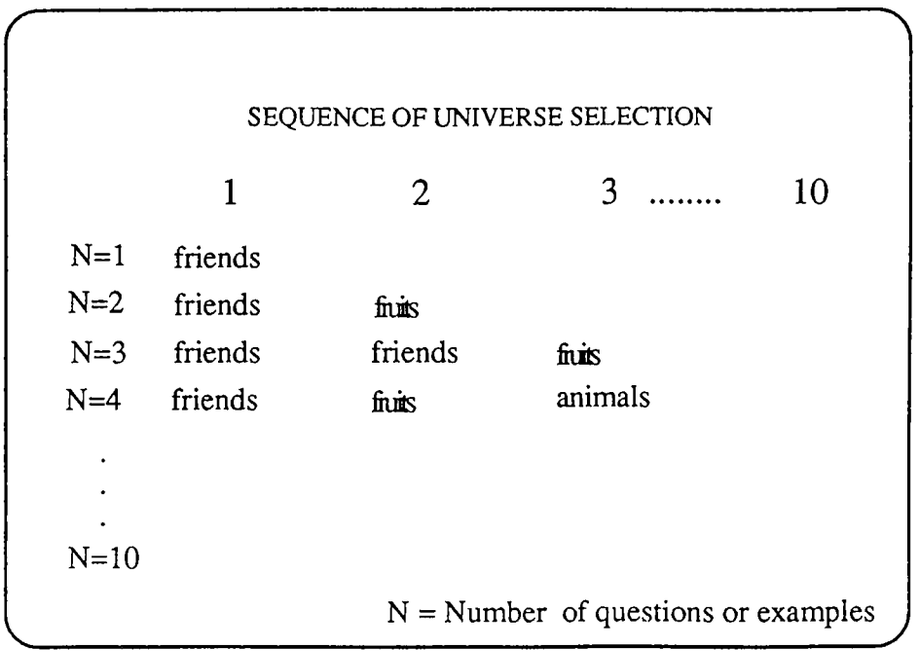

Fig.49 An example of universe selection

It is obvious that it is impossible to have infinite universes within a computer environment. Therefore the teacher is asked to input all universes in set notation into a proper text file which will be later used to conduct the session.

Another issue is how to select universes for question and answer generation. If these universes are randomly selected the student could get confused. In almost all text books, the examples and questions in set theory section employ simple universes with a maximum cardinality of 5. The idea behind this is to instruct the student using easy to grasp examples that will allow them to understand the topic faster. In the implemented system, the teacher is given the chance to select the strategy employed when generating questions and examples. The teacher might prefer to give every example using simple universes or abstract universes or a combination of them. Therefore, the teacher is asked to input an array that shows how many examples will be generated, which universes should be used and in what sequence. A simple example is provided that shows the number of questions and answers in the columns and the number of universes (N) used in the rows (Fig.49).

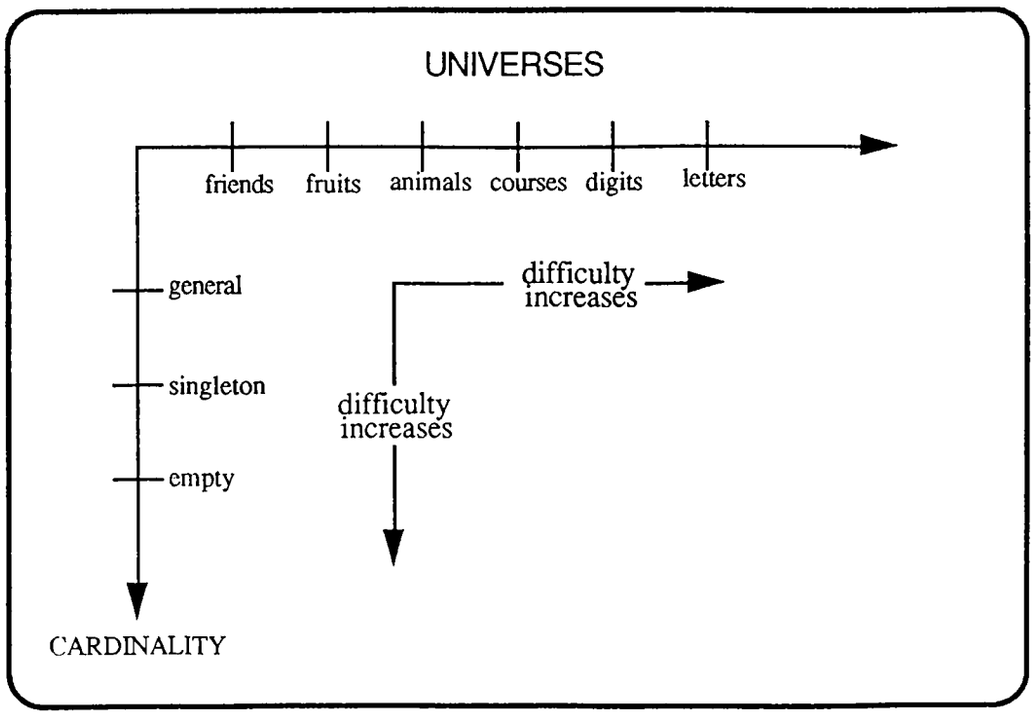

Fig.50. Distribution of difficulty level in set theory

If only one example or question is to be generated, the selected universe will be the set of fruits. However, if three examples or questions are required the universes will be selected in the sequence of are the set of friends, friends and fruits respectively.

This approach allows the system to be flexible in the difficulty level of questions and examples. This is a desirable feature that allows the teacher to tailor the system according to his/her needs. This adjustment can be performed by combining the difficulty level of sets and their minimum, maximum cardinalities. A simple set with a high cardinality would yield a relatively simple example whereas an abstract set with a low cardinality would form a difficult example (Fig.50).

Another dominant input is the threshold success level of a tutorial session. Actually two different threshold values are used by the system: one for normal sessions and one for the final exam. This level, as mentioned previously, controls flow of the program by acting as a decision parameter determining whether to continue or to repeat.

The last inputs required from the teacher are related with the final exam. These are:

- The number of questions for each question type.

  1) $A \oplus_1 (B \oplus_2 C)$
  2) $(A \oplus_1 B) \oplus_2 C$
  3) $(A \oplus_1 B) \oplus_2 (C \oplus_3 D)$

- Threshold success level of final exam

•**STUDENT MODEL:** The student model contains the current state of knowledge of the student and thus directly affects the flow of the session, since the knowledge contained in this module is a primary input to the teaching module which decides whether to continue or to repeat. The last question and answer section within each topic and the final exam is used to determine the users comprehension level and to decide what to do next.

•**NONDETERMINISM:** All aspects considered up to now are related with the decision of what the next step will be. Nondeterminism, on the other hand, is related with how each session will take place. This is done by including *randomness* in question and example generation techniques. Since the control strategy concept involves determination of the next step and the presentation method of each session nondeterminism is an effective factor of this strategy.

All the components of the implemented system, the strategies employed reflect the design selection performed and constitute a different approach to ITS applications.

5. CONCLUSIONS AND FURTHER RESEARCH

The Intelligent Set Theory Tutor is a good example of applying suitable programming techniques to subjects that possess a special internal structure. Set theory, due to its hierarchical structure, is easily applicable to an ITS system especially using OOP technique. Therefore, the contents of this work provides a well established system in the field of Intelligent Tutoring, employing OOP. OOP technique has been used for various applications but it is a new approach in the field of ITS.

The main objectives of this implementation were:

1) To design a system that is able to generate the instructional content like questions and examples on its own in a nondeterministic manner.

2) To distribute the control flow of the tutorial session to the student, the teacher, the student model and even to add some degree of nondeterminism.

3) To design a modular system that could be easily understood, enhanced and modified.

4) To demonstrate the applicability of OOP to certain topics within the academic curricula that exhibit hierarchical structure and inheritive properties.

It is believed that the objectives listed above have been met by the implemented system although there are certain possible future extensions and weak points of the system. But again all of these handicaps can be overcome with the advantages of OOP such as modular structure and easy extension. Future enhancements that could be performed upon this study can be listed as follows:

1) Improvement within the expert module by incorporating the bug catalogue.

2) Improving the student model simultaneously with the expert module.

3) Improving the capabilities of the user interface by including features like pull-down menus, graphical representations, attractive screen designs.

The attributes of set theory discussed in the introduction section such as well defined rules and operations, suitability to easy question and example generation and the fact that set concept forms a basis of the entire theory are proved to assist the design and implementation phases of the ITS. This argument can easily be validated by comparing the figures 30, 33 and 47 to the attributes of OOP such as inheritance, polymorphism and encapsulation.

The implemented system has not been experimented in a real classroom environment since it is only developed to test how OOP techniques can be applied to the field of ITS. Only after performing the above listed enhancements, would the system be sufficient enough to test on an actual environment.

6. REFERENCES

[1] Adams, D.M., and M. Hamm, *Artificial Intelligence and Instruction: Thinking Tools for Education* , T.H.E. Journal, August 1987, pp. 59-62.

[2] Aiello,l., M. Carioso, and A. Micarelli, *The Design of an Intelligent Tutoring System in Mathematics: The SEDAF Project,* Universita Degli Studi di Roma **La Sapienza** Technical Report No: TR88001, 1988.

[3] Chase, W.G.,and H.A. Simon, *The mind's eye in Chess*, Visual Information Processing, pp. 215-281.

[4] Chi, M.T.H., P.Feltovich,and R.Glaser, *Categorization and Representation of Physics Problems by experts and Novices*, Cognitive Science, vol.5, pp.121-152.

[5] Clancey, W.J., **Intelligent Tutoring Systems : A Tutorial Survey in Current Issues in Expert Systems ,** eds A. van Lamsweerde, P. Dufour, Academic Press, London, 1987.

[6] Dede, C., *A review and Synthesis of Recent Research in Intelligent Computer-assisted Instruction*, International Journal of Man-Machine Studies, vol.24, 1986, pp. 329-353.

[7] Duchastel, P., *ICAI Systems: Issues in Computer Tutoring*, Computers in Education, vol.13, no.1, 1989, pp. 95-100.

[8] Fischetti, E.,and A. Gisolfi, *From Computer-Aided Instruction to Intelligent Tutoring Systems*, Educational Technology, August 1990, pp. 7-17.

[9] Gevarter, W.B., **Intelligent Machines: An Introductory Perspective of Artificial Intelligence and Robotics**, Prentice-Hall, New Jersey,1985.

[10] Hennessy, S., T. O'Shea, R. Evertsz and A. Floyd, *An Intelligent Tutoring System Approach to Teaching Primary Mathematics*, Educational Studies in Mathematics, vol.20, 1989, pp. 273-292.

[11] Johnson, B.L., R.D. Bergeron and P. Malcolm, *Modeling the Teaching Consultant*, Computers in Education, vol.14, no.2, 1990, pp. 125-136.

[12] Kearsley, G., **Artificial Intelligence and Instruction: Applications and Methods**, Addison-Wesley Publishing Company, Massachusetts, 1987.

[13] Kinzer, C.K., R.D. Sherwood, and J.D. Bransford, **Computer strategies for Education Foundations and Content-area Applications**, Merrial Publishing Company, Columbus, 1986.

[14] LeClaire, B., *Object-Oriented Programming : An overview of key O-O concepts*, OR/MS Today, vol.18, no.1, February 1991, pp. 20-24.

[15] Nierstrasz, O.M., *What is the 'Object' in Object-Oriented Programming?*, **Objects and Things**, ed. D.Tsichritzis, Centre Universitaire D'Informatique, Universite de Geneve, March 1987, pp.1-13.

[16] Nierstrasz, O.M., *A Survey of Object-Oriented Concepts,* **Active Object Environments,** ed. D.Tsichritzis, Centre Universitaire D'Informatique, Universite de Geneve, July 1988.

[17] Nierstrasz, *Active Objects in Hybrid*, OOPSLA '87 proceedings.

[18] Nwana, S.N., *Intelligent Tutoring Systems: an Overview*, Artificial Intelligence Review, vol.4.,1990, pp.251-277.

[19] O'Shea, T., and J. Self, **Learning and Teaching with Computers : Artificial Intelligence in Education**, The Harvester Press Limited, 1988.

[20] Pascoe, G.A., *Elements of Object-Oriented Programming*, Byte, August 1986, pp.139-144.

[21] Self, J., *IKBS in Education*, Educational Review, vol.39, no.2, 1987, pp.147-154.

[22] Stefik, M., and D.G. Bobrow, *Object-Oriented Programming: Themes and Variations*, AI Magazine, January 1986, pp.40-62.

[23] **Turbo Pascal 5.5 : Object Oriented Programming Guide.**

[24] Wah, B., and Li, G.J., **Computers for Artificial Intelligence Applications**, IEEE Computer Society Press, Washington, D.C., 1986.

[25] Wenger, E., **Artificial Intelligence and Tutoring Systems**, Morgan Kaufmann Publishers, 1987.

[26] Zaniolo C. et.al., *Object-Oriented Database Systems and Knowledge Systems*, 1st International Workshop on Expert Database Systems, 1985, pp.1-17.