View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Bilkent University Institutional Repository

# DESIGN AND IMPLEMENTATION OF AN OBJECT - ORIENTED EXPERT SYSTEM SHELL

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND
INFORMATION SCIENCES
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Ismail Hakki Toroslu

June 1989

QA
79.9
.E96
T634
1989

# DESIGN AND IMPLEMENTATION OF AN OBJECT-ORIENTED EXPERT SYSTEM SHELL

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND

INFORMATION SCIENCES

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

İsmail HakkıToroslu

June 1989

İsmail Hakkı Toroslu

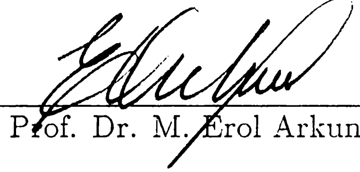
tarafından bağışlanmıştır.

QA
79.9
.E96
T684
1989
B 1859

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.

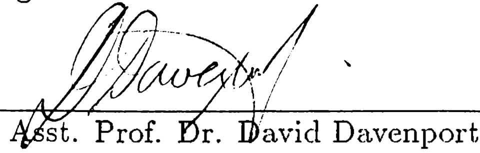
Asst. Prof. Dr. Halil Altay GÜVENİR (Principal Advisor)

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.



Prof. Dr. M. Erol Arkun

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.



Asst. Prof. Dr. David Davenport

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Science

# ABSTRACT


## DESIGN AND IMPLEMENTATION OF AN OBJECT-ORIENTED EXPERT SYSTEM SHELL



İsmail HakkıToroslu

M.S. in Computer Engineering and
Information Sciences

Supervisor: Asst. Prof. Dr. Halil Altay GÜVENİR

June 1989



Expert systems represent a new opportunity in computing. An expert system is a computing system capable of representing and reasoning about some knowledge-rich domain with a view to solving problems and giving advice. Expert system shells are developed to create expert systems in an easy way. In recent years the object-oriented paradigm has been developed. The object-oriented approach has many advantages such as data abstraction, program modularity, and structural data representation. Therefore, we are developing an expert system shell which stores knowledge and data in object-oriented style. Also, an object-oriented DBMS part of our shell satisfy the needs of several expert systems requiring large base of facts. Such shells can be used to build expert systems by only adding the domain-specific knowledge.



Keywords: Expert system, expert system shell, knowledge representation, object-oriented approach.

# ÖZET

## NESNESEL UZMAN SİSTEM KABUĞU TASARIM VE GERÇEKLEŞTİRİMİ

İsmail HakkıToroslu
Bilgisayar Mühendisliği ve Enformatik Bilimleri Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. Halil Altay GÜVENİR
Haziran 1989

Uzman sistemler bilgisayar bilimlerinde yeni bir konu olarak ortaya çıkmıştır. Uzman sistemler bilgi yoğun sistemlerin geliştirilmesinde kullanılan yazılım sistemleri olarak tanımlanabilir. Uzman sistem kabukları da uzman sistemlerin kolayca geliştirilebilmeleri için ortaya çıkmıştır. Son yıllarda yeni bir yazılım tekniği olarak ortaya çıkan nesnesel yaklaşım birçok üstünlüklere sahiptir. Bunlardan en önemlisi nesnesel yaklaşımın insanın düşünme tarzına yakın olmasıdır. Bu nedenle, bizim de geliştirmekte olduğumuz uzman sistem kabuğunda bilgilerin saklanması için nesnesel yaklaşım teknikleri kullanılmıştır. Sistemimizin bir parçası olan nesnesel veri tabanı büyük bilgi tabanını gerektiren uzman sistemlerin geliştirilmesine de olanak sağlamaktadır.

Anahtar Kelimeler: Uzman sistem, uzman sistem kabuğu, bilgi gösterimi, nesnesel yaklaşım.

# ACKNOWLEDGEMENT

I would like to thank Asst. Prof. Dr. H. Altay Güvenir, who is the supervisor of this thesis, because of his valuable help, remarks and advising this interesting thesis subject.

# TABLE OF CONTENTS

1 INTRODUCTION 1

2 EXPERT SYSTEMS 4

2.1 Types of Expert Systems . . . . . . . . . . . . . . . . . . . . . 5

2.2 Components of Expert Systems . . . . . . . . . . . . . . . . . . 6

2.3 Construction of Expert Systems . . . . . . . . . . . . . . . . . . 8

3 EXPERT SYSTEM SHELLS 9

3.1 Basic Properties of Expert System Tools . . . . . . . . . . . . 9

3.2 Some Examples of Expert System Tools . . . . . . . . . . . . 10

4 KNOWLEDGE REPRESENTATION MODELS 14

4.1 Production Rules . . . . . . . . . . . . . . . . . . . . . . . . . 14

4.2 Frames . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

5 OBJECT-ORIENTED APPROACH 20

5.1 Basic Concepts . . . . . . . . . . . . . . . . . . . . . . . . . . 20

5.2 Object Identity . . . . . . . . . . . . . . . . . . . . . . . . . . 22

6 THE OES SYSTEM 24

6.1 Unification of Expert and Database Systems Using Object-Oriented Approach . . . . . . . . . . . . . . . . . . . . . . . . 24

6.2 DBMS Part of the OES . . . . . . . . . . . . . . . . . . . . . . 26

6.3 Inference Engine Part of OES . . . . . . . . . . . . . . . . . . 38

6.3.1 Knowledge Representation Language of OES . . . . . 38

6.3.2 Control Structure of OES . . . . . . . . . . . . . . . . . 41

6.4 Implementation of OES . . . . . . . . . . . . . . . . . . . . . . 45

**7 CONCLUSION** **48**

**APPENDICES** **54**

**A SYNTAX OF THE KNOWLEDGE REPRESENTATION LANGUAGE OF OES** **54**

**B SYNTAX OF THE METHOD DEVELOPMENT LANGUAGE OF OES** **56**

**C AN EXAMPLE EXPERT SYSTEM IMPLEMENTED IN OES** **59**

C.1 Knowledge Base of ADVICE . . . . . . . . . . . . . . . . . . 60

C.2 Some Part of the Database of ADVICE . . . . . . . . . . . . 63

C.3 The Outputs of ADVICE for Some Executions . . . . . . . . 68

# LIST OF FIGURES

2.1 Structure of an ideal expert system. . . . . . . . . . . . . . . . 6

6.1 The structure of OES. . . . . . . . . . . . . . . . . . . . . . . . 25

6.2 The main menu of OES. . . . . . . . . . . . . . . . . . . . . . . 26

6.3 The menu of the DBMS part of OES. . . . . . . . . . . . . . . 26

6.4 The class hierarchy for a simple expert system. . . . . . . . . . 27

6.5 The menu of the DDL part. . . . . . . . . . . . . . . . . . . . . 27

6.6 The menu used for defining the properties of classes. . . . . . 28

6.7 Definition of the instance variable "TAKES" of class "COURSE." 29

6.8 Definition of the instance variable "PREREQUISITES"' of class "COURSE." . . . . . . . . . . . . . . . . . . . . . . . . 30

6.9 Definition of the message "COUNT" of class "COURSE." . . 32

6.10 The menu of the DML part. . . . . . . . . . . . . . . . . . . . . 33

6.11 Creation of an instance of class "STUDENT." . . . . . . . . . . 34

6.12 Deletion of an instance of class "COURSE." . . . . . . . . . . . 35

6.13 Main control algorithm of OES. . . . . . . . . . . . . . . . . . 42

6.14 Definitions of the classes of an example expert system. 43

6.15 Selecting a specific instance of class STUDENT. . . . . . . . 44

6.16 Obtaining the results of the execution of the expert system. . 45

# 1. INTRODUCTION

An expert system is a special kind of computer program that embodies the expertise of human experts in some specific domain. The behavior of an expert system is intended to be similar to that of a human expert in the specific field. This is the major difference of expert systems from conventional computer programs [24,15,16,33].

Another difference of expert systems from conventional computer programs is that their tasks usually have no algorithmic solutions or they must make conclusions based on incomplete information [13].

There are many tools for constructing expert systems ranging from general-purpose programming languages (e.g. LISP) to highly specialized tools (e.g. EMYCIN) [16]. Early expert systems are implemented with general-purpose programming languages. Earliest expert system tools such as EMYCIN are based on the past expert systems such as MYCIN [33]. Therefore, they were not general purpose tools. Today, there are several general-purpose expert system development tools such as OPS5 and ROSIE. Expert system shells can be viewed as new programming languages. Instead of the data objects in conventional languages such as integers and arrays, expert system shells maintain facts and rules. The control structure of expert system shells are also different from conventional languages. Conventional languages' control structures consist of sequential execution, conditional statements and loops. However, in expert system shells rule-chaining and pattern matching are the basic control structures [33].

Many expert system applications require both the problem solving capability and the management of large database of facts. The organization of new tools suitable for these types of applications is one of the most important issues in Computer Science [24].

Object-oriented knowledge representation model is used in several expert

systems, but there is no commercial expert system tool that has an object-oriented knowledge representation language [17,25,27]. Many expert systems and expert system shells provide database support in addition to their inference mechanisms [7,10,12,19,22,34]. However, these expert system tools are not suitable for all kind of expert system applications.

Each expert system application requires different properties. Some of them require very powerful database support, but for some expert system applications a simple database system or a static knowledge representation model may be enough, but they require numerical calculations or formulas. Therefore, the expert system shell must have all these properties to be a general purpose tool, and much research is being done on this subject [1,7,8,11,20,29,30,34].

Another important issue in expert system shells is the need for a high-level representation language for expressing the knowledge of human experts. The language should be readable, manageable and experts should be able to read and understand the rules easily. Among the present tools only a few has this property (e.g. ROSIE)[16]. Therefore, we designed a high-level representation language for this implementation.

The goal of this research is to design a general purpose expert system shell which is able to maintain large databases. An object-oriented approach is used for this reason. This implementation consists of two parts; object-oriented database management part is used to store large number of facts, and expert system shell part is used to invoke the application of the rules. Because of its object-oriented style, we called our system **Object-Oriented Expert System Shell (OES)**.

In the second chapter of this thesis, information about the expert systems is given. The basic properties, structures, type and the construction of the expert systems are explained in this chapter.

In the third chapter, expert system shells are defined and some well-known commercial expert system tools are introduced.

The fourth chapter defines the basic knowledge representation models used in expert systems. These models "production rules" and "frames" are defined and their advantages and disadvantages are discussed.

The fifth chapter gives the basic information about the object-oriented approach. Basic concepts of object-oriented approach and object identity

concepts are discussed in this chapter.

The sixth chapter is about the OES system. Its structure, knowledge representation model and DBMS part are explained in detail. The user interface of OES is also explained in this chapter by examples.

# 2. EXPERT SYSTEMS

The term "knowledge-based systems" and "expert systems" are used to describe the programs such as much knowledge embedded in the program and expert human methods are utilized to achieve expert-level performance [33].

Expert systems differ from conventional computer programs because their tasks have no algorithmic solutions and because they must often make conclusions based on incomplete or uncertain information [13]. The structures of expert systems are modular. Facts and other knowledge of the domain can be separated from the inference engine which applies the general knowledge to the particular problem. With this separation, the program can be changed by simple modification of the knowledge base.

Basic reasons of why expert systems are used instead of conventional systems are as follows [15]:

- Many difficult and interesting problems do not have tractable algorithmic solutions.
- Human experts achieve outstanding performance in their domain of expertise, therefore if computer programs embody human experts' knowledge, then, they should attain high performance also.
- Knowledge is a scarce resource. Extracting knowledge from humans and putting it in computable forms reduce the cost of knowledge reproduction. This reproduction is done by making the knowledge available for public in expert systems.

Expert systems have some differences from conventional systems. The first basic difference of expert systems from the conventional systems is that expert systems perform difficult tasks at expert level of performance. Another

difference is that they include domain-specific problem solving strategies instead of general but weak methods. The third basic difference of expert systems is that they provide explanations or justifications on their conclusions and on their inference processes. Because of those important differences of expert systems from the conventional systems, expert systems are defined as a new area in Computer Science.

## 2.1 Types of Expert Systems

Basically, expert systems can be classified into ten different types [15,32]. However, some expert system applications do not fit directly into one of those types, and they can have the properties of more than one of those types.

- *Interpretation* systems (e.g. speech understanding system) analyze data to determine its meaning.
- *Prediction* systems (e.g. weather forecasting system) infer consequences from given situations.
- *Diagnosis* systems (e.g. medical diagnosis system) infer system malfunctions from observable data.
- *Design* systems (e.g. circuit board design system) develop configurations of objects that satisfy the constraint of the design problem.
- *Planning* systems (e.g. experiment planning system) design actions that can be carried out to achieve goals.
- *Monitoring* systems (e.g. monitoring system for air traffic) compare observations of features that seem crucial to successful plan outcomes.
- *Debugging* systems (e.g. computer-aided debugging system for computer programs) prescribe remedies for malfunctions and include recommendations for correcting diagnosed problem.
- *Repair* systems (e.g. computer maintenance system) develop and execute plans to administer a remedy for some diagnosed problem.
- *Instruction* systems (e.g. education system) diagnose and debug student behavior.
- *Control* systems (e.g. business management system) include interpreting, predicting, repairing and monitoring system behaviors.

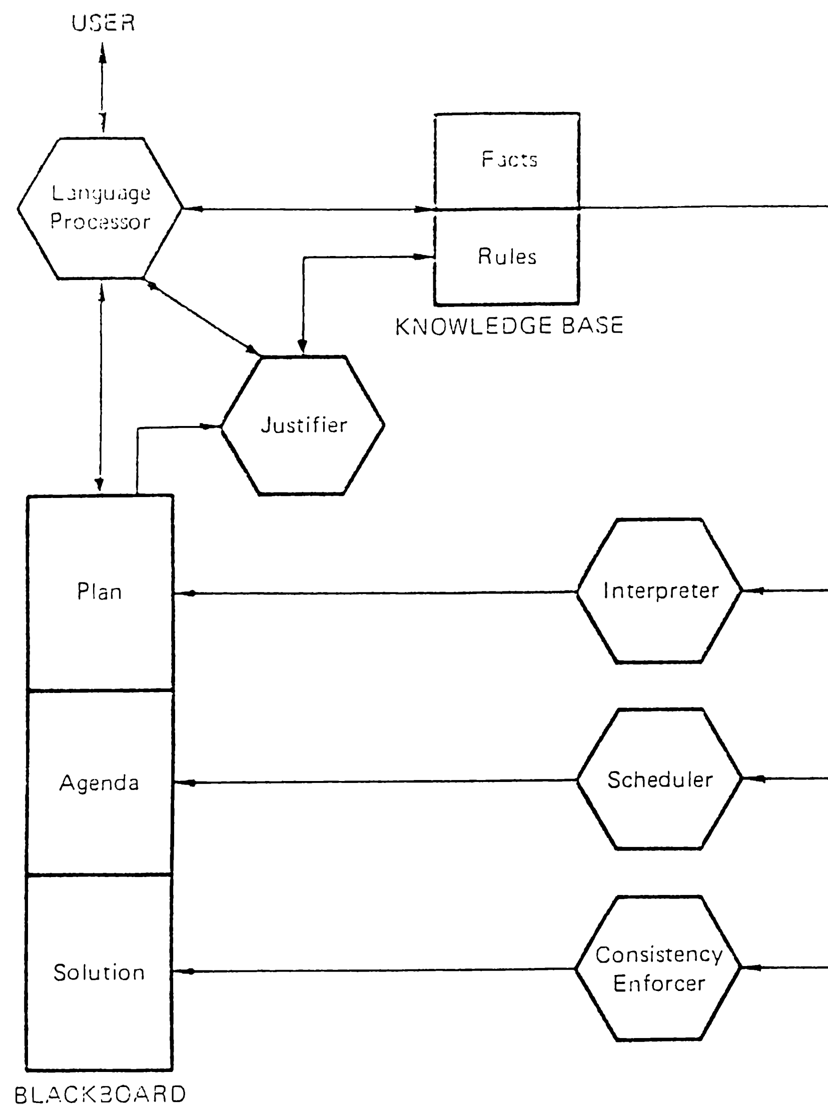



Figure 2.1: Structure of an ideal expert system.

## 2.2 Components of Expert Systems

Most expert systems have similar structures. None of expert systems contains all the components described here, but usually they have most of these components. An ideal expert system contains the following components [15] (Fig. 2.1):

- A language processor.
- A blackboard.

- A scheduler.
- An interpreter.
- A knowledge base.
- A consistency enforcer.
- A justifier.

A *language processor* is used for communication between the user and the expert system. This language is usually a restricted English-like language or other user interface facilities such as menus.

A *blackboard* is used to hold intermediate results. There are basically three types of decisions recorded on blackboard. The *plan* elements describe the overall steps of the current problem, including current plans, goals and problem states. The *agenda* elements include the actions waiting for the execution, that is the rules in the knowledge base which seem relevant to decision placed on the blackboard. The *solution* elements represent the candidate hypotheses and decisions generated by the system.

A *scheduler* is used to control the order of rule processing. The scheduler keeps the control of the agenda and determines the next action that will be executed.

An *interpreter* is used to execute the chosen agenda item by applying the corresponding knowledge base rule.

A *knowledge base* consist of facts and rules of the application domain.

A *consistency enforcer* is used to adjust previous conclusions when new data or knowledge alter their bases. The consistency enforcer tries to keep the knowledge base and database in an consistent form.

A *justifier* is used to explain the system behavior. In general, the justifier answers questions about why some conclusions was reached or how this conclusion was obtained by the system.

## 2.3 Construction of Expert Systems

There are five major stages in the evolution of an expert system [15]. They can be explained shortly as follows:

1. *Identification* is the process of determining problem characteristics. During identification, the knowledge engineer and expert work together to identify the problem area and define its scope.

2. *Conceptualization* is the process of finding concepts to represent knowledge. During conceptualization, the expert and the knowledge engineer find the key concepts, relations and information flow characteristics of the problem.

3. *Formalization* is the process of designing structures to organize knowledge. Formalization involves finding the formal representations corresponding to the key concepts and relations for some expert system tool.

4. *Implementation* is the process of formulating rules that embody knowledge. During implementation, the knowledge engineer combines and reorganizes the formalized knowledge to make it suitable to information flow characteristics of the problem.

5. *Testing* is the process of validating rules that embody knowledge.

# 3. EXPERT SYSTEM SHELLS

Many tools are used in the construction of expert systems. Those tools range from general purpose languages such as LISP to highly specialized expert system tools such as EMYCIN. Today, several general purpose tools are developed for the construction of expert systems. Those tools are called **expert system shells (tools)**. Expert system shells can be viewed as new programming languages. Instead of the data objects in conventional languages such as integers and arrays, expert system shells provide facts and rules. The control structures of expert system shells are also different from that of conventional languages. The control structure of conventional languages consist of sequential execution, conditional statements and loops. However, in expert system shells, rule-chaining and pattern matching are the basic control structures.

## 3.1 Basic Properties of Expert System Tools

The features needed in expert system tools depend on three things [16]:

- The characteristic of the domain, which includes the form of data and the structure of the problem.
- The characteristic of the approach to solving the problem, which includes the type of search, representation of knowledge and the form of the control.
- The desired characteristics of the expert system to be built, which includes the type of users and the method of extending the system.

The design of a tool for building an expert system involves many considerations including generality, completeness, language features, database structures and control methods.

In the design of an expert system tool one of the most important considerations is the need for a high-level representation language for expressing procedural knowledge [16]. If there is no such language, it becomes very difficult for the users to develop an expert system application or to extend an existing expert system. The language should be readable and manageable. With only a little training, application domain experts should be able to read and understand the rules written in this language.

Another issue that must be considered in the design of the expert system tool is the structure of database. Capability of the tool for representing static knowledge is extremely important, because if the system is too restrictive, even very simple problems cannot be implemented with this tool.

## 3.2 Some Examples of Expert System Tools

This section describes eight different well-known commercial expert system tools: EMYCIN, KAS, EXPERT, OPS5, ROSIE, RLL, HEARSAY-III, and AGE [16,3]. They are defined shortly and basic advantages and disadvantages of those tools are explained. The aim of this section is to give a general idea about the commercial expert system tools and their structures.

### EMYCIN

EMYCIN is a domain-independent version of MYCIN. Since the MYCIN was developed for diagnosis, EMYCIN is most suitable for deductive problems including diagnosis. EMYCIN uses backward-chaining control strategy. Knowledge is represented as production rules in EMYCIN. EMYCIN's greatest strength is its very convenient environment and user interface for building an expert diagnostic system. Major limitation of EMYCIN is its constrained control structure.

### KAS

KAS is based on the PROSPECTOR, which is a consultation program developed for diagnosis problems in mineral exploration. In KAS data is represented in semantic-network form and procedural knowledge is represented

as probabilistic inference rules. There are both backward and forward chaining control strategies in KAS. Binding of antecedent variables to be used in consequent actions is not permitted in KAS, therefore it cannot be suitable for some problems.

## EXPERT

EXPERT is a programming system developed for building consultation models based on classification problems. EXPERT evaluates its rules in an ordered manner. When more than one rule is applicable, the rule with the highest confidence is used. The major strength of EXPERT is that it is easy to use and permits the rapid development of a prototype model. However, it is not possible to develop some systems, especially those requiring extensive search.

## OPS5

OPS5 is a rule-based programming language. Rules in OPS5 are data-driven and operate on a single global database. Control in OPS5 is done by the recognize-act cycle, which is a simple loop in which rules with satisfied antecedents are found, one is selected and its action is performed. Unlike EMYCIN, KAS and EXPERT models, the OPS5 model does not have sophisticated user interface and explanation facilities. However, OPS5 is a general purpose tool, and it can be suitable for different types of applications.

## ROSIE

ROSIE is a general purpose rule-based programming system suitable for a broad range of knowledge engineering applications. ROSIE has an English-like syntax which has the capability of the creation and manipulation of its database. ROSIE has 3-types of inference mechanisms: ***state driven,*** where the state of the system directly causes a rule to fire; ***goal driven,*** where backward-chaining is used to find rules that will verify predictions in the rule conditions; *change driven,* where a database change causes a rule to fire. The main strength of ROSIE is its English-like syntax. This permits the user to write entirely readable code, even to those people unfamiliar

with programming. The English-like syntax speeds the task development process. A major weakness of ROSIE is its lack of accessibility to its own rules and control structure, and lack of facilities for database structure and construction.

## RLL

RLL is a structured collection of tools to help the knowledge engineer construct, use and modify expert systems. Its strength is its competence model of programming and its generality of data structures and algorithms. Its lack of a user friendly front end is the major limitation in developing the RLL model. All code had to be entered in a LISP-like format.

## HEARSAY-III

HEARSAY-III is a domain independent programming facility for developing prototype expert systems. It was designed to help the user in developing methods for representing and applying knowledge to a chosen problem area. In HEARSAY-III, the blackboard is used by the prototype expert system to store and coordinate information about the domain, partial solutions, and current activities. Execution usually results in the modification of blackboard information. HEARSAY-III model lacks sophisticated facilities for I/O, database construction, and explanation. The major strength of HEARSAY-III is its general-purpose control structure, which supports interaction among numerous sources of knowledge. A major weakness of HEARSAY-III is its lack of a high-level representation language.

## AGE

The AGE system is a tool for helping knowledge engineers design, build, and test different frameworks for expert systems. It provides an environment in which different representational and control techniques can be developed. The AGE model lacks useful facilities for I/O, database construction and explanation. A major strength of AGE is its flexibility of representation and its easy way for the user to apply the general control frameworks supplied by the system.

Five of the tools (EMYCIN, KAS, EXPERT, OPS5, ROSIE) are variations of conventional rule-based systems and thus provide the IF-THEN rule as a basic building block. OPS5 and ROSIE are general-purpose tools that provide greater flexibility of control and representation than do EMYCIN, KAS or EXPERT. The other tools (HEARSAY-III, AGE and RLL) are general-purpose systems for experimenting with expert system architecture. HEARSAY-III provides blackboard framework with cooperating knowledge sources as the basic design paradigm. AGE provides both a blackboard and backward-chaining paradigms. RLL provides a unit representation for both rules and facts.

Because of the restricted specialized structures in EMYCIN, KAS, and EXPERT, it is difficult to represent static knowledge(database) including objects having complex relationships between each other. Therefore, it is not possible to do search on those complex objects in these system. In RLL and AGE, static knowledge is represented using LISP code, therefore more complex than the one used in OPS5 and ROSIE. Other than the static knowledge, basically there are two ways of representing the knowledge. They are as follows:

- In declarative knowledge (such as predicate logic) most of the knowledge is represented as a static collection of facts accompanied by a small set of general procedures for manipulating them.
- In procedural knowledge the bulk of the knowledge is represented as procedures and it shows how to do things.

The declarative knowledge can be represented in EXPERT, OPS5, ROSIE and HEARSAY-III as procedural forms, and in RLL and AGE as static forms. EMYCIN does not have a direct way of representing declarative knowledge. In KAS, declarative knowledge can be represented as definitions. The procedural knowledge is represented in IF-THEN rules in all systems easily.

The ROSIE language seems to capture the intended meaning of the given knowledge most easily. This is because of its English-like syntax. Also, ROSIE code is easy to read and understand than the code of the other systems.

# 4. KNOWLEDGE REPRESENTATION MODELS

There are several knowledge representation models. Basic knowledge representation models are [35]:

- Semantic networks,
- Frames,
- Production rules,
- Predicate calculus, and
- Hybrid of those models.

However, in expert systems usually only two of those models are used for knowledge representation, which are *frames* and *production rules.* This chapter discusses those two most commonly used knowledge representation models.

## 4.1 Production Rules

Systems using production rules as a knowledge representation model are called **rule-based** systems. In a rule-based system, a rule base composed of a set of production rules and an inference engine controls the activity of the system [18]. Rules provide a modular and uniform approach to knowledge representation. Tools that support rules as their only representation paradigm are relatively simple to learn and use [23].

Two fundamental components of rule-based systems [5]:

- *Inference engine* which is a mechanism that uses the knowledge in the knowledge base, applying it to the problem.

- *Knowledge base* which contains rules of the very simple if-then decision form and facts.

Experts usually express their knowledge in term of a situation-action rules. Therefore, usually rule-based systems are used in expert systems, which require codifying the problem solving know-how of human experts. Most rule based systems share the following properties [14]:

- They incorporate human knowledge in if-then rules.
- Their skill increases as their knowledge base enlarges.
- They can solve a wide range of problems by selecting related rules and combining the results in appropriate ways.
- They determine the best way to execute the rules.
- They explain their conclusions.

Rule-based systems permit the representation of knowledge in a highly uniform and modular way. A knowledge engineer has to codify human knowledge using only if-then rules. Therefore rule-based systems are uniform.

The modularization of the program can be defined as the degree of separation of its functional units into isolatable parts. A program is modular if any functional unit can be changed (added, deleted or modified) with no unanticipated change to other functional units. Rule-based systems are highly modular, because the next rule to be invoked is determined only by the contents of the database and no rule is called directly [6]. Thus, the change of a rule does not require the modification of any other rule.

The most popular and effective representational form for declarative knowledge is pattern-action rules, which are called production rules in knowledge systems. Production rules are indeed a subset of the predicate calculus [9]. How the information in the rules is to be used during reasoning is added in rule-based systems. Production rules can be easily understood by the domain experts and have sufficient expressive power to represent a useful range of domain-independent inference rules.

PROLOG was the first general-purpose logic-based (using predicate calculus) programming language. PROLOG is a rule-based system that uses stored facts and rules to deduce solutions to goal patterns [14]. The predicate calculus has very general expressive power and well-defined semantics. However, it has some major disadvantages: firstly, it is difficult to define complex objects using predicate calculus, and secondly, domain experts have difficulty using the predicate calculus or understanding knowledge expressed in it [9]. Therefore, production rules are used instead of predicate calculus as a knowledge representation model.

In rule-based systems rules perform a variety of functions [14]:

- They define a parallel decomposition of state transition behavior. Every result can thus be traced to its antecedent data and intermediate rule-based inferences.
- They simulate deduction and reasoning by expressing logical relationships.
- They can simulate subjective decision making by using conditional rules to express heuristics.

Facts, the other kind of data in knowledge base, express assertions about properties, relations and propositions. In contrast to rules, which the rule-based systems interpret as imperatives, facts are usually static and inactive. In addition to its static memory for facts and rules, a rule-based system uses a working memory to store temporary assertions. These assertions record earlier rule-based inferences.

There are a number of shortcomings in conventional programming technology. They are [14]:

1. The nonspecifiability of programs.
2. The rapid changes in principles of operation that can arise during development.
3. The lack of user/expert participation in operations specification.
4. The lack of experimental development for computer based competence.
5. The lack of expertise in exploiting computer capabilities.

Rule-based systems have solutions to those shortcomings. The features of rule-based systems are [14]:

1. Modular know-how.
2. Knowledge bases for storing rules and facts that directly determine decisions.
3. The capacity for incremental development with steady performance improvements.
4. Explanations of results, lines of reasoning, and questions asked.
5. Intelligibly encoded beliefs and problem-solving techniques.
6. Inference chains assembled dynamically by built-in control procedures that can often perform efficient searches.

In contrast to conventional programming, rule-based programming requires a programmer to think more analytically than procedurally. Many experts represent their knowledge in rule-based manner, because rules seem like a natural way to express the situation-action heuristics of problem-solving protocols of experts and experts are able to develop learning procedures capable of inferring rules from experience.

As it explained in this section rule-based systems have three basic advantages. Rule-based systems are modular, uniform and natural (that is they are structured similarly to the way people think about solving a problem). In addition to their advantages, rule-based systems also have some disadvantages. Uniformity of rule-based systems can introduce a rigid structure that makes it difficult to follow the flow of the control in problem solving. Another disadvantege of rule-based systems is that, every execution must go through the match-action cycle in context data structure, making it difficult to efficiently execute predetermined situational sequences.

## 4.2 Frames

Sometimes "what to do and when" is not the only focus of interest. Sometimes a detailed representation of the physical or conceptual objects in the domain is important [18]. A frame language provides the knowledge base

builder with an easy means of describing the types of the domain objects that the system must model [9]. The knowledge base for a frame-based system consists of a large number of frames, each capturing a single prototypical description [5].

A frame provides a structured representation of an object or class of objects. In a frame language frames are represented in a hierarchical manner. The major disadvantages of production rules can be overcome by frames. The information stored in frames has often been treated as the database of the knowledge system. The control of the system is done by other parts of the system.

Frame languages are based on semantic networks. In semantic networks collection of objects (nodes) linked together by relationships (arcs) in an unrestricted graph structure. Nodes represent objects, concepts and situations. Arcs represent the relationships between nodes.

Some important advantages of semantic networks as a knowledge representation model are:

- It is easy to add, modify, delete nodes and relationships.
- Inheritance property, where one node can inherit relationships of other nodes without having direct links to them. By using this property objects form classes and a member of a class inherits all of the attributes of that class. Classes also form a hierarchical structure.

Semantic network is a very powerful knowledge representation model. However, it has some major disadvantages which prevent it to be used in real-life knowledge system applications. The major disadvantage is that there is no formal representation structure for semantic networks. Another basic disadvantage of semantic network is that because same kind of links are used in inheritance, it is not easy to distinguish between an individual inheritance and a class of inheritances. Therefore, the implementation of semantic networks is very difficult.

Because of those reasons, frames are developed as a knowledge representation model. In a frame-based representation a frame consists of a collection of slots that contain attributes to describe:

- An object,

- a class of objects,
- a situation,
- an action, and
- an event.

Therefore, each individual or class is represented by a frame. Frames can be organized to represent relationships between frames. Member links and subclass links are used to create those relationships. ***Member links*** are used to show the class of individual frames and ***subclass links*** are used to create a hierarchical structure between frame classes.

Frames have sets of attribute descriptions called ***slots*** [28]. A frame used for representing a class contains prototype descriptions of members of the class and descriptions of the class as a whole. In the KEE system, prototype descriptions are distinguished from other descriptive information by the use of two kinds of slots, which are own slots and member slots. ***Own slots*** can occur in any frame and are used to describe attributes of the object or class represented by the frame. ***Member slots*** can occur in frames that represent classes and are used to describe attributes of each member of the class itself.

Although frame languages provide no specific facilities to declaratively describe behavior, they provide various ways of attaching procedural information expressed in some other language (e.g. LISP) to frames [28]. This *procedural attachment* capability enables behavioral models of objects and expertise in an application domain to be built.

Knowledge systems have proved to be particularly effective for performing diagnostic tasks in a variety of domains. Such tasks involve determining a description of a given situation in terms of the types of situations the system knows about. Frame languages have several representational features that are particularly useful for designing and directing the reasoning processes that are involved in diagnostic tasks.

The primary advantage of frame representation is that the more concise and compact the knowledge base, the shorter the amount of time required for searching for specific information. Frames allow for layers of abstraction to separate out low-level details from high-level abstracts. However, frame representation also have some disadvantages, such as their inabilities to express procedural knowledge of the human experts.

# 5. OBJECT-ORIENTED APPROACH

Object-oriented paradigm has been developed in recent years. Object-oriented approach has many advantages such as data abstraction, program modularity, and structural data representation. There are three basic concepts in object-oriented computation [28]. They are as follows:

- **Object**,
- **message**, and
- **class**.

The major application areas of the object-oriented approach are programming languages, database management systems, knowledge representation, CAD/CAM systems and office automation systems [36]. Object-oriented systems have many advantages in the production of the software. Code sharing, portability, flexibility are some of its advantages. Also, in object-oriented approach, the problem can be easily decomposed into subproblems. Because of these advantages, object-oriented systems are being used in many areas of Computer Science.

## 5.1 Basic Concepts

In an object-oriented system, all conceptual entities are modeled as objects [31,26]. An object encapsulate a private data set and it can only be accessed or modified by sending messages to it. Objects having the same properties are grouped into classes and each object is an instance of a class. Another important mechanism of object-oriented paradigm is ***inheritance*** property. Classes can be arranged in a hierarchical structure. When a new class is

created, it must be defined as a subclass of an existing class, and this new class inherits all properties of its superclass. This hierarchical representation and inheritance property is a natural way of representation of entities.

The state of an object is represented using a collection of instance variables. Also, the value of each instance variable is an object. The description of object's instance variables, methods, and messages defines a class, and an object created using this description is called an instance of this class. The class provides necessary information to create and use instances of that class. An instance has a single class, but a class may have any number of instances. The class concept reduces storage and duplication, because class definition is shared by all instances of the class.

Messages of object-oriented systems are analogues to procedure calls of conventional programming systems. In an object-oriented system messages are used to access or modify the objects. In most object-oriented systems message sending process is done as follows:

<object-name> <message-name> <arguments>

Arguments of a message (if exist) are objects and this message also returns another object. When a message is sent to an object, corresponding method is activated. Methods describe how to perform some operations and messages specify which method will be executed. Therefore, methods behave like the procedure bodies of conventional systems. Implementation of methods are not visible from outside the objects. As instance variables and messages, methods are also defined in the class definition.

The class concept provides modularization and conceptual simplicity as well as reducing duplication [2]. All messages, methods and instance variables are defined only once in the class definition and they are shared by all instances of the class. Another tool in object-oriented approach which reduces storage and duplication, is inheritance. Inheritance means a class can be defined as a subclass of another class and inherits all the descriptions of its superclass. Therefore, all classes in the system form a class hierarchy. In this hierarchy, a parent node is called as a superclass and a child node is called as a subclass. Inheritance enables programmers to create new classes by specifying only the differences between a new class and an existing class. In this way, a large number of code is shared and can be reused between classes. There are two types of inheritance; simple inheritance and multiple inheritance. In the *simple inheritance*, a class can have only one superclass

and the class hierarchy forms a tree, whereas in the ***multiple inheritance*** a class can have more than one superclasses and therefore, the class hierarchy forms a lattice structure.

## 5.2 Object Identity

Every language must have some way to identify one object from other objects. *Identity* is a property of an object which distinguishes it from all others [21]. Each object must keep a separate identity regardless of its location in the memory or how it is accessed and content or how it is modeled with descriptive data.

Most programming languages and file systems provide user-defined names, that is variables in languages and file names in file systems, to represent identity. This approach mixes addressability and identity concepts [21]. Addressability is external, but identity is internal to an object. Addressability provides a way to access to an object. However, identity provides a way to represent an object independently of how it is accessed. One problem of this representation is that a single object may be accessed in different ways and bound to different variables without a way of finding out whether they refer to the same object or not.

In database languages, which are designed to support large number of objects, every real-world object is an individual. In other words, there is something unique for everything. When a real-world object is modeled, some subset of its description is used in the identity. An identifier key is some subset of the attributes of an object which is unique for all objects in the relation. Therefore, this approach mixes data value and identity concepts, and this creates some problems [21]. One problem is that identifier keys cannot be changed, even though they are user-defined descriptive data. Another problem is that identifier keys cannot provide identity for every object in the relational model, because each attribute or meaningful subset of attributes cannot have identity. The third problem is that the choice of which attribute to use for an identifier key may need to change. The last problem is that the use of identification keys cause joins to be used in retrievals instead of path expressions.

An object-oriented system, including large number of objects, is ***consistent*** if no two distinct objects have the same identifiers and for each identifier

present in the system there is an object with this identifier [21]. An object can belong to multiple objects through set membership or attribute assignment without being replicated and without being owned by any objects. Object-oriented systems provide several special operators to compare or manipulate objects having strong identity. Identities of objects are not related with their contents or locations in object-oriented systems.

# 6. THE OES SYSTEM

## 6.1 Unification of Expert and Database Systems Using Object-Oriented Approach

Data Base Management System (DBMS) and expert system technologies are completely different. DBMSs deal with simple facts in well-organized structures, whereas expert systems deal with complex objects (the knowledge)[24]. In this study an object-oriented expert system tool, called **Object-Oriented Expert System Shell (OES)**, is developed. It unifies these two different technologies in one system to satisfy the needs of many expert system applications requiring both problem solving capability and management of large base of facts.

There are basically two ways in combining expert systems and DBMS technologies, namely homogeneous and heterogeneous approaches [24]. In *homogeneous approach*, both rules and facts are represented in the same programming system. PROLOG is an example of this type of approach. In PROLOG rules and facts are represented and can be manipulated in the same way. PROLOG seems to be suitable for small applications, but it has major limitations. For the program to be executed, all rules and facts must be in the main memory, which limits the size of the application. Another disadvantage of this approach is its limited data structure in representing both rules and facts.

*Heterogeneous approach* is the second way of combining expert system and DBMS. This approach can be implemented in two ways. In expert system-DBMS *loosely coupled* system, DBMS can act as a server to the expert system and supplies data when needed by the expert system. Separating data retrieval system from the expert system inference mechanism is the major

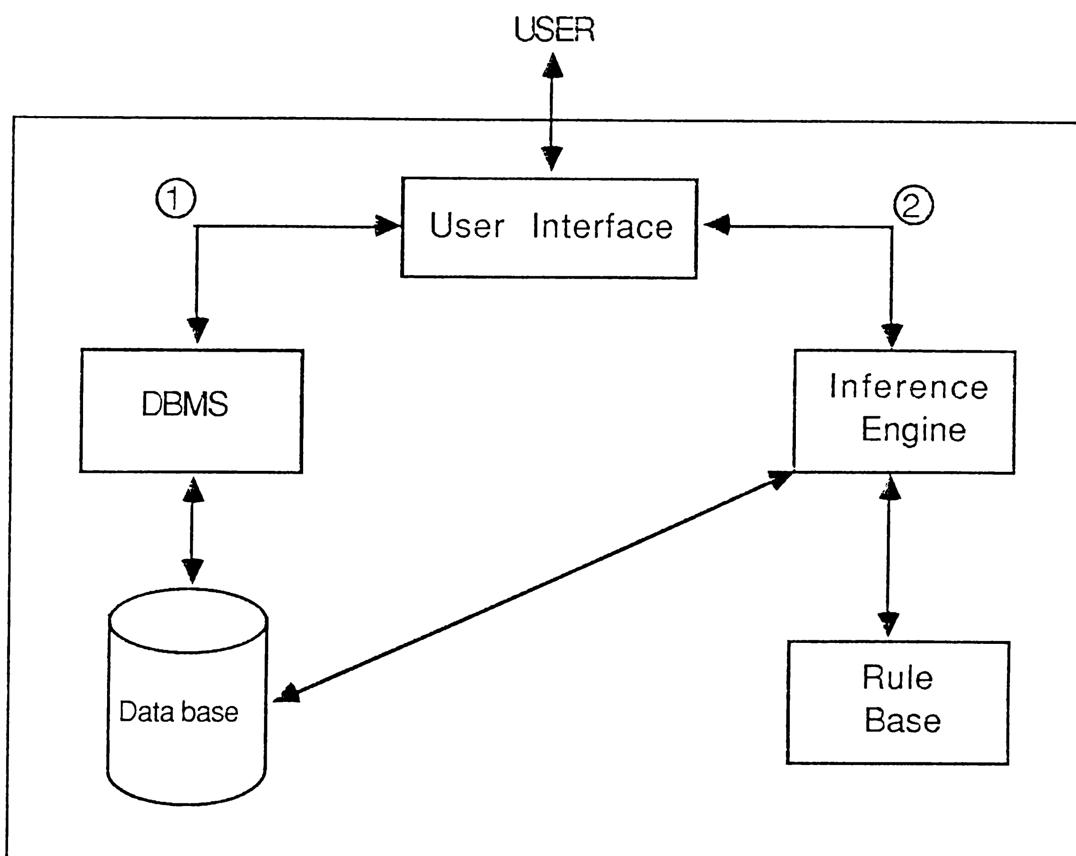



Figure 6.1: The structure of OES.

disadvantage of this approach. However, in expert system-DBMS *tightly coupled* system interaction between expert system and database can take place directly. Database seems as the extension of Knowledge Base. Expert system decides when and how to use the database.

The aim of our implementation is to create an expert system-DBMS tightly coupled system. Object-oriented approach seemed suitable to implement such a system. Basic concepts of object-oriented approach are used both in DBMS and inference engine parts of OES. Therefore, an impedance mismatch problem which occurs between the programming languages and data manipulation languages does not exist in OES.

OES has two main parts, which are the DBMS part and the Inference Engine part (Fig. 6.1, 6.2). DBMS part is used to create and manipulate the contents of the database of the applications. Inference Engine executes the rules using the objects created by DBMS.

To develop a new expert system application in OES, first, a user must create the necessary database using the DBMS of the system. An existing database can be very large, including many classes, and therefore it can be used in several expert systems. Even the different expert systems may use the same parts of the database. Thus, if some part of the required data already

```
      MAIN MENU

  1- Object-Oriented DBMS
  2- Inference Engine
  3- Quit

     >>
```

Figure 6.2: The main menu of OES.

```
      OBJECT-ORIENTED DBMS MENU

  1- Modify Class Structure
  2- Create / Delete Instances of Class
  3- Quit

     >>
```

Figure 6.3: The menu of the DBMS part of OES.

exists in the database of the system, it is not necessary to create it again. Therefore, a large amount of code can be shared.

## 6.2 DBMS Part of the OES

Object-oriented DBMS part of OES is a menu-driven system. It has a Data Definition Language (DDL) part which is used to modify the class structure by adding new classes to the system, or by deleting existing classes of the system, and a Data Manipulation Language (DML) part which is used to create and delete instances of the classes of the system.

The first menu of the DBMS part of OES, which is used to select one of those DDL and DML parts is "OBJECT-ORIENTED DBMS MENU"(Fig. 6.3). When the user selects the first item of this menu, the menu of the DDL part appears on the screen, and when the user selects the second item of this menu, the menu of the DML part of the system appears on the screen.

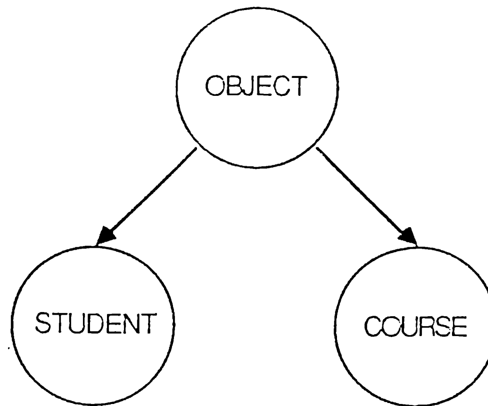



Figure 6.4: The class hierarchy for a simple expert system.

```
CLASS STRUCTURE SYSTEM PRIMITIVES

1- Add New Class
2- Delete Class
3- Quit

    >>
```

Figure 6.5: The menu of the DDL part.

The menu used in the DDL part of the system is "CLASS STRUCTURE SYSTEM PRIMITIVES" (Fig. 6.5). It has only "Add new class" and "Delete class" options. If "Add new class" choice is selected by the user, the system first asks the name of this class. If the user enters a valid class name (i.e., it does not exist in the existing class structure, and it is a string of alphanumeric characters, starting with an alphabetic character) the superclass of this class is asked to the user. To do this, existing class names are listed on the screen, and the user is asked to select one of them as the superclass of this newly defined class. The class hierarchy is in the form of the tree in OES, and the name of the root class is OBJECT. All the user defined classes must either be defined as the subclass of the class OBJECT or any other previously defined user defined class. The class hierarchy of the simple expert system application used to advice courses to the students is shown in the figure 6.4. When the superclass of the class is defined, all properties of this superclass (its instance variables and messages) are inherited by this new class. Then, another menu is displayed on the screen to define other instance variables and messages of this class (Fig. 6.6). This menu is called "CLASS DEFINITION MENU" and it has four items. The user can add new instance variables and messages and can delete existing instance variables and messages of the class by using this menu.

```
CLASS DEFINITION MENU

1- Add Variable
2- Delete Variable
3- Add Message
4- Delete Message
5- Quit

   >>
```

Figure 6.6: The menu used for defining the properties of classes.

The first item of this menu is used to add new instance variables to the class. When this item is selected, the user is asked for the name of an instance variable. After the name of the instance variable is entered, the properties of this variable is defined as follows (Fig. 6.7, 6.8):

- First, the user is asked whether this instance variable is a "primitive" variable or a "collection" variable.
- Secondly, the domain of the variable is defined. The domain of the instance variable can be another user defined class as well as the primitive data types such as "integer", "float", "boolean", and "string." If the user wants to define the domain of the variable as another user defined class, all existing class names are listed on the screen and the user selects one of them as the domain of this instance variable. This way, very complex class structures even including the variables having the domain of the class itself may be created. For example, the instance variable "prerequisites" of the class "course" must have the type "course", because, it contains the list of the courses that must be taken before taking this course. If the course "CS202" can be taken after taking the courses "CS102" and "CS201", in the definition of this course instance, prerequisites of this course must be defined as those two courses. Therefore, in some cases, an instance variable of a class also has the domain of the same class.
- In the third step, the type of an instance variable is defined as "temporary" or "persistent" The contents of "persistent" variables are stored with the instances in the database of the system, but "temporary" variables are used only in the execution, and their values are not saved in

```
Enter instance variable name: TAKES

   Enter (1-Primitive  2-Collection): 2

   Enter (1-Integer  2-Float  3-Boolean  4-String
          5-User defined class): 5

   List of User Defined Classes:
   =============================
      1-COURSE
      2-STUDENT

   Enter class number: 1

   Enter (1-Temporary  2-Persistent): 1

   Enter (0-Set  1-Bag): 0
```

Figure 6.7: Definition of the instance variable "TAKES" of class "COURSE."

the database after the execution.

- The last step is used only for "temporary" variables. For some primitive variables, the user does not want to change the value of it anymore once it is instantiated. Therefore, the user is asked whether this variable is "changeable" or not. Also, the user can define collection variables such that they may include the same element in their lists more than once (i.e., bag) or the same element cannot be repeated in their lists (i.e., set). The user selects one of the "bag" and "set" options to define this property of the "collection" variables.

Using those four steps (only the first three steps are used for "persistent" variables), all instance variables of the class are defined.

The second item of the "CLASS DEFINITION MENU" is used to delete instance variables of the classes. If this item is selected, the system asks the name of an instance variable that will be deleted, and if it already exists, it is deleted from the definition of the class.

The third item of "CLASS DEFINITION MENU" is used to add new messages to the classes. To define a new message, first its name is entered

```
Enter instance variable name: PREREQUISITES

   Enter (1-Primitive  2-Collection): 2

   Enter (1-Integer  2-Float  3-Boolean  4-String
          5-User defined class): 5

   List of User Defined Classes:
   ==============================
      1-COURSE
      2-STUDENT

   Enter class number: 1

   Enter (1-Temporary  2-Persistent): 2
```

Figure 6.8: Definition of the instance variable "PREREQUISITES"' of class "COURSE."

and the properties of the object returned by the execution of this message is defined. After that, the number of arguments of the message and the number of variables used in the method definition corresponding to this message are specified. Then, domains of those arguments and variables are defined (Fig. 6.9).

Definition of the object, returned by the execution of the message, includes three steps:

- First, it is asked whether it is a "primitive", or a "collection" object.
- Then, its domain is asked to the user as in the definition of the instance variables.
- For some messages, the user may want to execute it only once, store the result of that execution and return that value whenever this message is invoked again. Therefore, it is asked to the user whether this message will be executed only once or it can be executed many times.

After obtaining the number of arguments of the message and the number of variables of the method, each of them is defined as follows:

- First, the name of an argument or a variable is asked.
- Next, it is asked whether it is a "primitive" or a "collection" variable.
- Then, its domain is asked to the user as in the definition of the instance variables.

The last item of "CLASS DEFINITION MENU" is used to delete existing messages of the classes. When this item is selected, the name of the message is asked to the user, and the message is deleted from the definition of the class.

The definition of the class is completed by exiting from "CLASS DEFINITION MENU" and the definition of this class is added to the existing class structure.

The second choice of "CLASS STRUCTURE SYSTEM PRIMITIVES" is "Delete class" option. When this item is selected, the system lists the names of existing user defined classes and asks the user to choose one of them, and deletes that class from the class structure. Whenever a class definition is dropped, all its instances are deleted automatically, since instances cannot exist outside of a class. However, subclasses of this class are not dropped, but they will gain its superclass as thier immediate superclass. Further, when a class is dropped, its subclasses will not lose the instance variables (and methods) that they had previously inherited from that class in OES.

The second item of "OBJECT-ORIENTED DBMS" menu is used to select the DML part of the system. The menu displayed when this item is selected is "OBJECT UPDATE MENU"(Fig. 6.10). This menu contains only two items. They are "Create New Object" and "Delete Object." Only two basic functions of the DML are implemented in the OES system. Those two functions can be used to create new objects and delete existing objects of the classes. Object update and query processing functions are not implemented in OES. If the user wants to create a new instance of any existing class, he/she must select the first choice of "OBJECT UPDATE MENU" and to delete an existing instance of any class, he must select the second choice of this menu.

In the creation of the new instance, the system first asks the class of the object and then displays all "persistent" instance variables of this class to the user to enter a value for each of them (Fig. 6.11). If an instance variable has a primitive domain, the user can enter any valid value of that domain for that instance variable. However, if an instance variable has any user defined

```
Enter message name: COUNT

    Enter (1-Primitive  2-Collection): 2

    Enter (1-Integer  2-Float  3-Boolean  4-String
           5-User defined class): 1

    Enter (0-Invariant  1-Changeable): 1

    Enter the number of the arguments: 1

    Enter the number of the variables of the method: 1

Enter argument name: C_LIST

    Enter (1-Primitive  2-Collection): 2

    Enter (1-Integer  2-Float  3-Boolean  4-String
        5-User defined class): 5

    List of User Defined Classes:
    =============================
       1-COURSE
       2-STUDENT

    Enter class number: 1

Enter variable name: I

    Enter (1-Primitive  2-Collection): 1

    Enter (1-Integer  2-Float  3-Boolean  4-String
           5-User defined class): 1
```

Figure 6.9: Definition of the message "COUNT" of class "COURSE."

```
OBJECT UPDATE MENU

1- Create New Object
2- Delete Object
3- Quit

   >>
```

Figure 6.10: The menu of the DML part.

class as its domain, the system displays all instances of that class and asks the user to select any of them(or some of them for "collection" variables) as the value of this instance variable. This way seems too much time consuming because all instances of one class is searched sequentially. However, there is no way to prevent this time consuming search other than the querying the instances of the class. In object-oriented systems, each object must have an identity independently from its content and its location in the storage. This property of object-oriented systems prevent them from creating pointers to the objects using some part of their contents or locations in the storage, as identifiers of these objects. Because of the lack of the query processing facilities in OES, there is no way of identifying a specific instance for the user other than looking at the content of that object. Therefore, this sequential search is required.

The deletion process is also similar to the object creation process. First, the class name of the object is asked to the user, and then all instances of this class are listed to the user one by one and the user is asked to select one instance to delete from that class (Fig. 6.12). However, the deletion process does not end by deleting an object from the class. To preserve the integrity constraints, some additional process must be done after deleting an instance.

There is a similar integrity problem in object-oriented DBMSs with referential integrity constraint of relational model. Referential integrity rule can be defined as follows; every value in a foreign key (primary key of any other relation) in relation $R$ must either be equal to the primary key value of a tuple in relation $S$ or be wholly null [4]. This integrity constraint can be explained with an example as follows:

Let us assume that in Supplier-Parts database there are three relations;

```
   Variable NAME String and Primitive
      Enter a string: Ismail_Hakki_Toroslu

   Variable YEAR Integer and Primitive
      Enter an integer: 1

   Variable SEMESTER String and Primitive
      Enter a string: spring

   Variable STANDING String and Primitive
      Enter a string: probation

   Variable COURSES_TAKEN has Domain of Class COURSE and
Collection

      A COURSE instance:
      ==================
         CNO: math101
         REQUIRED_YEAR: 1
         REQUIRED_SEMESTER: fall
         GIVEN_SEMESTER: fall
         PREREQUISITES:
         TYPE: must_course

      Is this COURSE instance that you want?
      Enter (1-Yes  0-No): 1
   .
   .
```

Figure 6.11: Creation of an instance of class "STUDENT."

```
A COURSE instance:
==================
   CNO: math101
   REQUIRED_YEAR: 1
   REQUIRED_SEMESTER: fall
   GIVEN_SEMESTER: fall
   PREREQUISITES:
   TYPE: must_course

Is this COURSE instance that you want to delete?
Enter (1-Yes  0-No): 0

A COURSE instance:
==================
   CNO: phys101
   REQUIRED_YEAR: 1
   REQUIRED_SEMESTER: fall
   GIVEN_SEMESTER: fall
   PREREQUISITES:
   TYPE: must_course

Is this COURSE instance that you want to delete?
Enter (1-Yes  0-No): 0

A COURSE instance:
==================
   CNO: cs101
   REQUIRED_YEAR: 1
   REQUIRED_SEMESTER: fall
   GIVEN_SEMESTER: fall spring
   PREREQUISITES:
   TYPE: must_course

Is this COURSE instance that you want to delete?
Enter (1-Yes  0-No): 1
```

Figure 6.12: Deletion of an instance of class "COURSE."

Supplier (S), Part (P), and Supplier_Part (SP). The relation S has four attributes; supplier number (S#), which is a primary key of this relation, supplier name (SNAME), status (STATUS), and city (CITY). The relation P also has four attributes; part number (P#), which is a primary key of this relation, part name (PNAME), color (COLOR), and weight (WEIGHT). The last relation SP has three attributes; S# and P#, which are together primary key of this relation, and quantity (QTY). In relation SP, S# and P# are foreign keys, because they are the primary keys of relations S and P.

If Supplier-Parts database includes the tuples S(S1, Smith, 10, Paris), SP(S1, P1, 100) and SP(S1, P2, 200), and if the S(S1, Smith, 10, Paris) tuple is deleted from the relation S, the tuples SP(S1, P1, 100) and SP(S1, P2, 200) must also be deleted from the relation SP to preserve referential integrity. If they are not deleted from the relation SP, these two tuples become meaningless, because supplier S1 does not exist in supplier relation but it supplies part P1 with quantity 100 and part P2 with quantity 200. Therefore, those two tuples must be deleted from the SP relation.

A similar problem occurs in object-oriented DBMSs. This problem can be explained with an example as follows:

Assume that our class structure includes two classes; COURSE and STUDENT. The domain of the instance variables PREREQUISITES, which is an instance variable of the class COURSE, and COURSES_TAKEN, which is an instance variable of the class STUDENT, are the user defined class COURSE. We can create the following instance of those classes (The Object-ids used in these examples are not the same in OES. They are used to make the examples easily understandable):

```
OBJECT-ID   CLASS NAME   INSTANCE VARIABLE    VALUE
=========   ==========   =================    =====

  C001        COURSE       CNO                math101
                            .
                            .
                            .
                          PREREQUISITES          -

  C002       COURSE        CNO                math102
```

```
                         .
                    PREREQUISITES        C001

  S001     STUDENT  NAME                 Ismail_Toroslu
                         .
                         .
                         .
                    COURSES_TAKEN        C001 C002
```

If an instance C001 is deleted from class COURSE, to preserve the integrity constraint some update operations must be done on other instances of classes COURSE and STUDENT. Because an instance C001 is deleted, the prerequisites of no course can be the course having an object-id C001, and no student can take this non-existing course. Therefore, after deleting this object, other instances must be updated as follows:

```
OBJECT-ID   CLASS NAME   INSTANCE VARIABLE   VALUE
=========   ==========   =================   =====

  C002        COURSE       CNO               math102
                           .
                           .
                           .
                          PREREQUISITES        -

  S001       STUDENT      NAME                Ismail_Toroslu
                           .
                           .
                           .
                          COURSES_TAKEN       C002
```

This update operation is done in OES by changing the corresponding pointers in the memory and in the files where those classes are stored.

## 6.3 Inference Engine Part of OES

When the second item of "MAIN MENU" (Inference Engine Part) is selected, the system asks the name of the expert system application to the user and then executes it. Rules of the application must have been written using the knowledge representation language of OES, and stored in a file before the execution.

Expert system shells should have a high-level representation language for expressing the knowledge of human experts. This language should be readable and manageable.

### 6.3.1 Knowledge Representation Language of OES

Knowledge representation language of OES is very flexible. The user can write the same rule in several different ways. Therefore, the user can select the best way to write the rules. The syntax of the knowledge representation language of OES in BNF notation is given in Appendix A.

In OES, a newly developed knowledge representation model is used to express knowledge of human experts. This model is a hybrid model which combines production rules with the object-oriented approach. Therefore, it has the advantages of both the production rules and frame-based knowledge representation models (because of its object-oriented style). In OES all rules are written using IF-THEN constructs. In addition to those IF-THEN rules, there is an object-oriented DBMS of OES and all the entities used in those IF-THEN rules correspond to the objects, instance variables of the objects, class names and messages of those classes created in this DBMS part.

A method, which is an executable code corresponding to a message, is an other way of expressing procedural knowledge. In some expert system applications, numerical calculations, input/output processing and some set operations may be needed. OES supports methods for these types of applications. The user can write methods in a specific language of OES, and the messages execute those methods and return their results when they are invoked. The messages can be used in the rules of an expert system. The syntax of the method development language in BNF notation is given in Appendix B.

The rules are written using the keywords, such as "if", "then", "and" , and

the entities (which are concepts of an object-oriented approach) developed in DBMS. Those entities are instance variables of classes, class names which are pointers to instance objects of that class, and messages. For example, the rule for "If a *student* had received 'F' from a *course* and the *course* is a 'must-course' and the *course* is given in this semester and the number of courses assigned to the *student* is less than 5 then the *student* must take this *course*." can be written in OES as follows:

```
if type of course is equal_to  'must_course'
and course is member_of f_courses_list of student
and semester of student is member_of given_semester of course
and student count(takes) is less_than 5
then student takes course;
```

In this rule `student` and `course` are class names, therefore they are pointers to the instances of those classes. `Type` and `given_semester` are instance variables of the class `course`. The variable `given_semester` is a collection variable and can take one or more values. Its meaning in this rule is that, the course may be given in "fall" or "spring" semesters or in both semesters. `F_courses_list`, `semester`, and `takes` are instance variables of the class `student`. The variable `f_courses_list` is a collection variable with type pointer to the class `course`. Therefore, its value is a list of courses, that is the list of the courses that the student had received "F." The variable `takes` has the same type as the variable `f_courses_list`, but this variable is a temporary variable and it is not saved in database. In this example `takes` contains the lists of courses already assigned to the student. When this rule is fired and its condition part is satisfied for some course and student instances, in the action part, the current course instance is added to the collection instance variable `takes` of the current student instance. At the end of the session, the user can check the courses assigned to the student by looking at the content of this variable. `Count` is a message with one attribute, and it is sent to current instance of the class `student` to find the number of courses in the variable `takes`.

In OES, methods are simple codes which are executed when the corresponding messages are invoked. This simple method definition only involve sequential executions, which include arithmetic calculations, input/output processing statements, and set operation functions. An example message which is used to find the averages of the union of two sets (sets of integers) has the following method definition:

```
union(S1,S2,L);
Avg = sum(L) / count(L);
return(Avg);
```

In this method definition, the first line is used to find the union of the sets S1 and S2, which are the parameters of this method. The union of those two sets is put into the set L. The second line is used to find the averages of the numbers in the set L. The function sum finds the sum of the numbers in this set and the function count finds the number of the elements in the set L. The last line returns this average when this method is executed by the corresponding message, and the result is returned to the main system.

In this method, Avg and L are the variables declared with the message definition to use in the implementation of the method. The variables S1 and S2 are the parameters of the method. Therefore, the message that executes this method can be called from the main system as follows:

<class_name> <message_name> ( < Set_1 > , < Set_2 >)

The <class_name> is indeed an instance of this class, and two variables < Set_1 > and < Set_2 > are bound to the variables S1 and S2 in the method. < Message_name > and the name of the file which is used to store method definition must be the same in OES.

Another method that includes input/output operations which is used to find the Value Added Tax (VAT) of the price of a good can be implemented as follows:

```
write(Good);
prompt('Enter the tax rate: ');
read(Rate);
VAT = Price * Rate / 100;
return(VAT);
```

When this method is executed, the user is asked to enter the tax rate of the Good, and *VAT* is calculated using this tax rate and the price of the good and the result is returned.

### 6.3.2 Control Structure of OES

OES has a data driven control structure. In most data driven systems, data are stored as facts, but in OES, because of its object-oriented style, data are stored as objects.

Before the execution of the expert system application is started, the user must define which classes will be used for this application and how they will be used. To do this, for each user defined class, the system asks the user whether a specific instance of this class will be used, or all instances of the class is required in the execution, or this class is not needed in this application (Fig. 6.14). In figure 6.14, the classes of the expert system which is developed for giving advice to students are defined. All instances of the class `COURSE` is used in this expert system application, and a specific `STUDENT` instance is needed for the execution of the system. When a specific instance of a class is needed in the execution, the system lists the instances of that class one by one and asks the user whether he wants that instance or not, until one of them is selected by the user. (Fig. 6.15). The purpose of this expert system execution is to advice courses to this specific student. Therefore, only two classes `COURSE` and `STUDENT`, are used in this expert system application.

When the system starts execution all class names which are the pointers to instances are initialized to the first instances of the classes, if those classes are defined such that all instances of them will be used in the execution. Although there is no order in the instances of the classes, they are searched sequentially in the order they are created. However, some class names may be pointed to the specific instances by the user before the execution as explained before. These pointers do not change during the execution.

During execution, the system tries to fire the rules with the current instances of the classes. Whenever a rule is fired, all other rules are tried again. This loop continues until no rule can fire with the current instance objects. Then, one of those instances, whose class is defined such that all instances of it to be used in the execution, is replaced with the next one from the same class, and the loop starts again. This process continues until the action "stop" is executed or all instances of classes are exhausted. The main control algorithm of OES is shown in the figure 6.13.

When the execution of the system terminates, the results of the session can be obtained by the user by checking relevant instance variables. The purpose of OES is to find the values of the instance variables defined by the

```
 get the classes whose all instances will be used
 (C1_list)
   initialize these classes with the first instances
 get the classes whose specific instances will be used
 (C2_list)
   initialize these classes with those specific instances
initialize the fired_rules_list to nil
while (stop is not executed
       and all instances of C1_list are not exhausted)
begin
  initialize this_rule to first_rule
  initialize fire_flag to false
  while (not all rules are checked and fire_flag is false)
  begin
     if this_rule is not in fired_rules_list
     then try to fire it
       if it is fired
       then make fire_flag true
          and put this rule into fired_rules_list
     get next rule
   end
   if all rules are checked
     and fire_flag is false
     and stop is not executed
   then get new instance from one of the classes in C1_list
     and initialize fired_rules_list to nil
 end
```

Figure 6.13: Main control algorithm of OES.

```
DEFINITION OF THE CLASSES OF THE APPLICATION
============================================

Class Name : COURSE
   0 ) This class is not used in this application
   1 ) A specific instance of this class is used
       in the application
   2 ) All instances of this class is needed in
       this application
Enter Your Choice : 2

Class Name : INSTRUCTOR
   0 ) This class is not used in this application
   1 ) A specific instance of this class is used
       in the application
   2 ) All instances of this class is needed in
       this application
Enter Your Choice : 0

Class Name : STUDENT
   0 ) This class is not used in this application
   1 ) A specific instance of this class is used
       in the application
   2 ) All instances of this class is needed in
       this application
Enter Your Choice : 1
```

Figure 6.14: Definitions of the classes of an example expert system.

```
A STUDENT instance:
===================
   NAME : Ugur_Gudukbay
   YEAR : 2
   SEMESTER : spring
   .
   .
   .
Is this STUDENT instance that you want?
  Enter (1-Yes  0-No) : 0

A STUDENT instance:
===================
   NAME : Faruk_Polat
   YEAR : 3
   SEMESTER : spring
   .
   .
   .
Is this STUDENT instance that you want?
  Enter (1-Yes  0-No) : 0

A STUDENT instance:
===================
   NAME : Ismail_Hakki_Toroslu
   YEAR : 1
   SEMESTER : spring
   .
   .
   .
Is this STUDENT instance that you want?
   Enter (1-Yes  0-No) : 1

This STUDENT is selected...
```

Figure 6.15: Selecting a specific instance of class STUDENT.

```
Execution of the system terminated...
=======================================
Instance variable MATH_COURSES

   Do you want to see the content of this variable?
   Enter (1-Yes  0-No): 0

Instance variable TAKES

   Do you want to see the content of this variable?
   Enter (1-Yes  0-No): 1
```

Figure 6.16: Obtaining the results of the execution of the expert system.

user as "temporary" variables. These instance variables are not saved with the instances in database. For example, if an expert system is written for designing the list of courses to take for a given semester for the student, in the action part of the rules, the variable used to store the list of courses may be assigned a new value or may be modified (The variable `takes` is used to store the advised courses). At the end of the session, the user can obtain the list of the courses by asking the value of this instance variable. When the execution terminates, the system displays all the "temporary" variables of the classes used in this expert system application, and asks the user whether he wants to see the contents of those variables or not (Fig. 6.16). If the result is "yes", the content of the variable is displayed, and by this way the result of the expert system application can be obtained. In the example shown in figure 6.16 all courses advised to the student are listed by displaying the content of the variable `takes`.

## 6.4 Implementation of OES

OES has been implemented on Sun system running under Unix environment, and the C programming language is used in the implementation of OES. No special features of Sun system is used in the implementation of OES, therefore it is possible to execute OES on every standard C and Unix environment.

The OES system consists of eleven modules. One of those modules is a header file used for the global data structure declarations (*main.h*). Other modules of the OES system and their functions are as follows:

- The module *main.c* is the shortest module and it controls the whole system. When the system is executed the control of the execution starts from this module and also the initializations of the main data structures are done in this module.

- The module *lexical.c* is used to find the tokens of the data files. The tokens of both the knowledge base and the method definitions are analyzed and required symbol tabels and token lists are created by this module.

- The module *syntax.c* is used for parsing the tokens of the knowledge base. This module also creates some error messages to the user whenever required. If the knowledge base is syntatically error free, necessary data structures are created corresponding to the rules of the knowledge base.

- The module *semantic.c* is used to find other the errors in the knowledge base which could not be found by the *syntax.c*. Those errors include type and argument mistmatches.

- The module *msyntax.c* is used for parsing the tokens of the method definitions. It is similar to the *syntax.c*.

- The module *mseman.c* is used to find the other errors of the method definitions. This module is similar to the *semantic.c*.

- The module *struct.c* is the DDL part of the OES system. The modifications on the class structures of the system is done by this module. Class definitions, deletion of the existing classes, maintaining of the class hierarchy are the basic functions of this module.

- The module *object.c* is the DML part of the OES system. Secondary storage management of the database system including the object create and delete operations is the basic function of this module.

- The module *mexecute.c* executes the methods and return their results. Numerical calculations, input/output processing and some set operations are the basic functions of this module.

- The module *forward.c* executes the rules of the expert system applications. This module is the inference engine of the OES system.

A *Makefile* is used to complie and link those modules of the system. After the compilation and the link operations, an executable code is created with

the name **oes**. The whole system consists of around nine thousand lines of the C codes and it requires around 250 KB of the disk memory. In the execution of the system some additional data files are created to store the class structure of the expert system application. Those files are *class.i*, *message.i* and *variable.i*. Also, the system creates files for each user defined classes and keeps the instances of those classes in these files.

The aim of this study is to experiment how an expert and database systems can be unified as a tightly-coupled system using object-oriented approach. Instead of implementing the whole object-oriented DBMS, only a portion of it, is implemented. The database side of OES includes only the DDL part and object create and delete facilities as the DML part. Therefore, it is not a real DBMS, but it is a database system and the basic object-oriented concepts are used in it. To make the implementation simple, class hierarchy is represented in the form of a tree. The knowledge representation language involves production rules as the basic knowledge representation model. Usually, experts express their knowledge in IF-THEN rules, and they can easliy understand the knowledge expressed in production rules. Therefore, production rules are used as a basic knowledege representation model. In addition to the production rules, methods are used to represent procedural knowledege. Both in the knowledege representation part and database part object-oriented concepts are used. Thus, impedence mismatch problem does not exist in OES.

# 7. CONCLUSION

The aim of this study is to unify expert and database systems as a tightly-coupled system using object-oriented approach. Therefore, only a portion of the database side of the system is implemented to maintain the objects that will be used in the expert system side of the system. The system developed in this study is called Object-Oriented Expert System Shell (OES).

OES is a general-purpose expert system shell. It has a DBMS as a part of the whole system. A new software technique, object-oriented approach, is used in OES. Object-oriented approach provides a natural way of representing knowledge and solves the impedance mismatch problem between DBMS and expert system language. Also, high-level language of OES makes it easy for the users to express their knowledge.

The OES system has been designed and implemented, and it is a prototype system. OES is tested only with some small expert systems. An example expert system application is given in the Appendix C. The execution of such simple expert system applications that are developed with OES, does not takes more than a few seconds. The major limitation of OES is that only one instance of a class can be active at a time. Thus, it is not easy to make comparisons among the objects of the same class. For those objects, different classes may be created, and then the instances of those different classes can be compared by this way. Therefore, OES may not be suitable for some expert system applications. However, the idea used in OES can be developed to make it suitable for all expert system applications.

In our prototype system only one instance of a class can be active at a time, because each class name is a pointer to an instance of that class. This limitation of the system can be overcame by creating multiple pointers to the classes. To do this, for each class, several pointers must be defined before the execution of the system. Also, the lack of a query processing facilities of the database side of the system is another important limitation of it. However,

the query processing facility can easily be added to the whole system, and this increases both the speed and the usability of the system.

Although, OES is not a complete system (since query processing facilities of DBMS do not exist and there is only one pointer to each class), it is possible to develop many different expert system applications, especially requiring large databases, using OES. Therefore, we can claim that the unification of expert systems with object-oriented databases as tightly-coupled systems is a reasonably good approach.

# REFERENCES

[1] Akman, V., ten Hagen, P., Rogier, J., Veerkamp, P., "Knowledge Engineering in Design," Knowledge-Based Systems, Vol. 1, No. 2, pp. 67-77, March 1988.

[2] Banerjee, J., et. al., "Data Model Issues for Object-Oriented Applications," ACM Transactions on Office Information Systems,Vol. 5, No. 1, pp. 3-26, Jan. 1987.

[3] Barstow, D. R., "Languages and Tools for Knowledge Engineering," Building Expert System, ed. Hayes-Roth, F., Waterman, A., Lenat, D. B. , Addision-Wesley Publishing Company, Inc., pp. 283-345,1983.

[4] Date, C. J.,*An Introduction to Database Systems*, Addision-Wesley Publishing Company, Inc.,1981.

[5] Davis, R., "Knowledge-Based Systems: The View in 1986," AI in 1980s and Beyond, ed. Grimson, E. L., Patil, R. S., The MIT Press, pp. 43-74,1987.

[6] Davis, R., King, J. J., "The Origin of Rule-Based Systems in AI," Rule-Based Expert Systems, ed. Buchanan, B. G., Shotliffe, E. H., Addision-Wessley Publishing Company, pp. 20-52,1985.

[7] Delcambre, L. M. L., "RPL An Expert System Language with Query Power," IEEE Expert, pp. 51-61, Winter 1988.

[8] Dhar, V., Pople, H. E., "Rule-Based versus Structure-Based Models for Explaining and Generating Expert Behavior," Communications of the ACM, Vol. 30, No. 6, pp. 542-555, June 1987.

[9] Fikes, R., Kehler, T., "The Role of Frame-Based Representation in Reasoning," Communications of the ACM, Vol. 28, No. 9, pp. 904-920, Sept. 1985.

[10] Fisher, E.L., " An AI-Based Methodolgy for Factory Design," AI Magazine, Vol. 7, No. 4, pp. 72-85, Fall 1986.

[11] Francioni, J. M., Kandel, A., " A Software Engineering Tool for Expert System Design," IEEE Expert, pp. 33-41, Spring 1988.

[12] Frank, J. L., Duffield, C. A., Swearingen, C. A., "Mentor-I: An Expert Database System for Student Guidence," IEEE Expert, pp. 40-46, Summer 1988.

[13] Gevarter, W. B., "Expert Systems: Limited But Powerful," Application in Artificial Intelligence, ed. Andriole, S. J., Petrocelli Books, Inc., pp.125-139.

[14] Hayes-Roth, F., "Rule-Based Systems ," Communications of the ACM, Vol. 28, No. 9, pp. 921-932, Sept. 1985.

[15] Hayes-Roth, F., Waterman, A., Lenat, D. B., "An Overview of Expert Systems," Building Expert Systems, ed. Hayes-Roth, F., Waterman, A., Lenat, D. B. , Addision-Wesley Publishing Company, Inc., pp. 283-345,1983.

[16] Hayes-Roth, F., Waterman, A., "An Investigation of Tools for Building Expert Systems," Building Expert Systems, ed. Hayes-Roth, F., Waterman, A., Lenat, D. B. , Addision-Wesley Publishing Company, Inc., pp. 169-214,1983.

[17] Hever, S., Koch, U., Cryer, C., "INVEST: An Expert System for Financial Investment," IEEE Expert, pp. 60-68, Summer 1988.

[18] Jackson, P., "Review of Knowledge-Representation Tools and Techniques," IEE Proceedings, Vol. 134, No. 4, pp. 224-230, July 1987.

[19] Kaiser, G. E., et. al. , "Database Support for Knowledge-Based Engineering Environments," IEEE Expert, pp. 18-32, Summer 1988.

[20] Kaiser, G. E., Feiler, P. H., Popovich, S.S., "Intelligent Assistance for Software Development and Maintenance," IEEE Software, pp. 40-50, May 1988.

[21] Khoshafian, S. N., Copeland, G. P., "Object Identity," ACM OOPSLA'86 Proceedings, pp. 406-416, Sept. 1986.

[22] Marcus, S., Stout, J., Mc Dermott, J., "VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking," AI Magazine, Vol. 9, No. 1, pp. 95-112, Spring 1988.

[23] Mettrey, W., "An Assessment of Tools for Building Large Knowledge-Based Systems ," AI Magazine, pp. 81-89, Winter 1987.

[24] Missikoff, M., Wiederhold, G., "Towards A Unified Approach For Expert And Database Systems," Expert Database Systems: Proceedings from the First International Workshop, ed. Larry Kerscberg, The Benjamin / Cummings Publishing Company, Inc., pp. 383-397, 1986.

[25] Mohan, L., Kashyap, R. L., "An Object-Oriented Knowledge Representation for Spatial Information," IEEE Trans. on Software Engineering, Vol. 14, No. 5, pp. 675-681, May 1988.

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

[27] Phillip, C., Sheu, Y., "VLSI design with object-oriented knowledge bases," Computer Aided Design, Vol. 20, No. 5, pp. 272-280, June 1988.

[28] Ramamoorty, C. V., Sheu, P. C., "Object-Oriented Systems," IEEE Expert, pp. 9-15,Fall 1988.

[29] Scohen, E., Smith, R. G., Buchanan, B. G., "Design of Knowledge-Based Systems with a Knowledge-Based Assistant," IEEE Trans. of Software Engineering, Vol. 14, No. 12, pp. 1771-1790, Dec. 1988.

[30] Smith, D. R., Kotik, G. B., Westfold, S. J., "Research on Knowledge-Based Software Environments at Kestrel Institute," IEEE Transaction on Software Engineering, Vol. 11, No. 11, pp. 1278-1295, Nov. 1985.

[31] Stefik, M., Bobrow, D. G., "Object-Oriented Programming: Themes and Variations," AI Magazine, pp. 40-62, Jan. 1986.

[32] Stefik, M., et. al., "The Organization of Expert Systems, A Tutorial," Artificial Intelligence, Vol. 18, pp. 135-173, 1982.

[33] Szolovits, P., "Expert System Tools and Techniques: Past, Present and Future," AI in 1980s and Beyond, ed. Grimson, E. L., Patil, R. S., The MIT Press, pp. 43-74,1987.

[34] Tsur, S., "LDL- A Technology for the Realization of Tightly Coupled Expert Database Systems," IEEE Expert, pp. 41-51, Fall 1988.

[35] Wolfgram, D. D., Dear, J. J., Galbraith, C. S., *Expert Systems for the Technical Professional*, John Wiley Sons Inc, New York,1987.

[36] Zaniola, C., et. al., "Object-Oriented Database Systems and Knowledge Systems," 1st International Workshop on Expert Database Systems, pp. 1-17,1985.

# A. SYNTAX OF THE KNOWLEDGE REPRESENTATION LANGUAGE OF OES

- rule_base
  <rule_base> ::= <rule_list>
- rule_list
  <rule_list> ::= <if_statement> ; | <if_statement> ; <rule_list>
- if_statement
  <if_statement> ::= **if** <condition_part> **then** <action_part>
- condition_part
  <condition_part> ::= <condition_list>
- condition_list
  <condition_list> ::= <condition> | <condition> **and** <condition_list>

- condition
  <condition> ::= <class_name> <variable_name> <operand> |
  <operand> **is** <operator_and_operand> |
  <operand> **is** <simple_operator>
- operand
  <operand> ::= <class_name> <message_name> **(**<argument_list>**)** |
  <variable_name> **of** <class_name> |
  <constant>
- operator_and_operand
  <operator_and_operand> ::= [**not**] <condition_operator> <operand>
- condition_operator
  <condition_operator> ::= **equal_to** | **member_of** |
  **subset_of** | **greater_than** | **less_than** |
  **greater_than_or_equal_to** | **less_than_or_equal_to**

- simple_operator
  <simple_operator> ::= [not] **null** | [not] **true** | [not] **false**

- action_part
  <action_part> ::= <class_name> <variable_name> <operand> |
  <variable_name> **of** <class_name> **is** <operand> |
  **skip** <class_name> | **stop**

- argument_list
  <argument_list> ::= <argument> | <argument> **,** <argument_list>

- argument
  <argument> ::= <variable_name> | <constant>

- constant
  <constant> ::= <single_constant> | <multi-valued_constant> | <class_name>

- multi-valued_constant
  <multi-valued_constant> ::= [ <single_constant_list> ]

- single_constant_list
  <single_constant_list> ::= <single_constant> |
  <single_constant> **,** <single_constant_list>

- single_constant
  <single_constant> ::= <integer> | <float> | <string_constant>

- string_constant
  <string_constant> ::= '<string>'

# B. SYNTAX OF THE METHOD DEVELOPMENT LANGUAGE OF OES

- method
  <method> ::= <statement_list>

- statement_list
  <statement_list> ::= <statement> ; | <statement> ; <statement_list>

- sentence
  <statement> ::= <functions> | <set_functions> | <assignment_statement>

- functions
  <functions> ::= **read** ( <variable1> ) | **write** ( <variable> ) |
  **prompt** ( <string> ) | **return** ( <variable1> )

- set_functions
  <set_functions> ::= **set_copy** ( <variable> , <variable1> ) |
  **union** ( <variable> , <variable> , <variable1> ) |
  **intersection** ( <variable> , <variable> , <variable1> ) |
  **difference** ( <variable> , <variable> , <variable1> )

- assignment_statement
  <assignment_statement> ::= <variable1> = <expression>

- expression
  <expression> ::= <term> | <expression> + <term> |
  <expression> − <term>

- term
  <term> ::= <factor> | <term> * <factor> |
  <term> / <factor>

- factor
  <factor> ::= <variable> | ( <expression> ) | <constant> |
  **sum** ( <variable> ) | **product** ( <variable> ) |
  **count** ( <variable> ) | **count_objects** ( <class_name> ) |
  **sumf** ( <variable> ) | **productf** ( <variable> ) |
  **countf** ( <variable> ) | **icountf_objects** ( <class_name> )

- variable
  <variable> ::= <variable1> | <variable2> | <variable3>
  <variable1> : variable used in method definition
  <variable2> : parameter of method (argument of message)
  <variable3> : instance variable of class

Method definition language provides 16 different functions. They are described as follows:

- *Read* is used to input value to a single variable.
- *Write* displays the value of a single variable on the screen.
- *Prompt* is used to display a string on the screen.
- *Return* returns the value of its parameter as the value of the method.
- *Set_copy* is used to copy a set from one variable to the other. It copies the value of its first parameter into the second one.
- *Union* finds the union of the two sets (first two parameter) and puts the result into the new variable (third parameter).
- *Intersection* finds the intersection of the two sets (first two parameter) and puts the result into the new variable (third parameter).
- *Difference* finds the difference of one set from the other one (first two parameters) and puts the result into the new variable (third parameter).
- *Sum* finds the sum of the values in the set and returns its result as integer.
- *Product* finds the product of the values in the set and returns its result as integer.
- *Count* returns the number of the elements in the set.
- *Count_objects* returns the number of the instances of the class.

- *Sumf* finds the sum of the values in the set and returns its result as float.
- *Productf* finds the product of the values in the set and returns its result as float.
- *Countf* returns the number of the elements in the set. It returns its result in the float type.
- *Countf_objects* returns the number of the instances of the class. It returns its result in the float form.

# C. AN EXAMPLE EXPERT SYSTEM IMPLEMENTED IN OES

The name of this expert system is "ADVICE" and it is developed to advice courses to the students. Only two classes `course` and `student` are used in this expert system application. Instance variables of the class `course` are as follows:

- `Cno`: Course number (e.g. CS101).
- `Required_year`: If the value of this variable is 2 then second year students must take this course.
- `Required_semester`: The content of this variable shows the required semester of the course.
- `Given_semester`: The content of this variable shows the given semesters of the course. Some courses may be given in only one semester in a year, but some courses may be given in both semesters of a year.
- `Prerequisites`: Shows all he courses that must have been taken before taking this course.
- `Type`: Shows whether the course is a must course or an elective course.

Instance variables of the class `student` are as follows:

- `Name`: Name of the student.
- `Year`: Year of the student.
- `Semester`: Semester of the student.

- `Standing`: This variable shows whether the student is probation or satisfactory in the last semester.
- `Courses_taken`: List of the courses that the student had passed.
- `F_courses_list`: List of the courses that the student had failed.
- `Takes`: This temporary variable is used to store the courses that are adviced to the student by this expert system.
- `Math_course`: This is a temporary variable.
- `Elective`: This is a temporary variable.

Also the message **count** is defined for the class **student**. This message is used to find the number of the courses assigned to the student.

## C.1 Knowledge Base of ADVICE

```
if course is member_of courses_taken of student
then skip course;

if student count(takes) is equal_to 5
then stop;

if type of course is equal_to 'must_course'
and standing of student is not equal_to 'repeat'
and standing of student is not equal_to 'probation'
and course is not member_of courses_taken of student
and required_year of course is equal_to year of student
and required_semester of course is equal_to semester of student
and prerequisites of course is subset_of courses_taken of student
and student count(takes) is less_than 5
then student takes course;

if type of course is equal_to 'must_course'
and course is member_of f_courses_list of student
and semester of student is member_of given_semester of course
and student count(takes) is less_than 5
```

```
then student takes course;

if type of course is equal_to 'must_course'
and standing of student is equal_to 'probation'
and course is not member_of courses_taken of student
and required_year of course is equal_to year of student
and required_semester of course is equal_to semester of student
and prerequisites of course is subset_of courses_taken of student
and student count(takes) is less_than 5
then student takes course;

if type of course is equal_to 'must_course'
and course is not member_of courses_taken of student
and semester of student is member_of given_semester of course
and required_year of course is less_than year of student
and prerequisites of course is subset_of courses_taken of student
and student count(takes) is less_than 5
then student takes course;

if type of course is equal_to 'must_course'
and course is not member_of courses_taken of student
and semester of student is member_of given_semester of course
and required_year of course is equal_to year of student
and required_semester of course is equal_to 'fall'
and semester of student is equal_to 'spring'
and prerequisites of course is subset_of courses_taken of student
and student count(takes) is less_than 5
then student takes course;

if student year 3
and student semester 'fall'
and type of course is equal_to 'math_course'
and course is not member_of courses_taken of student
and student count(takes) is less_than 5
then student math_course course;

if student year 2
and type of course is equal_to 'nontechnical_elective'
and course is not member_of courses_taken of student
and student count(takes) is less_than 5
```

```
then student elective course;

if student year 3
and type of course is equal_to 'nontechnical_elective'
and course is not member_of courses_taken of student
and student count(takes) is less_than 5
then student elective course;

if math_course of student is not NULL
then student takes math_course of student;

if elective of student is not NULL
then student takes elective of student;

if student year 4
and type of course is equal_to 'restricted_elective'
and student wants_software_courses() is true
and course group 'software'
and course is not member_of courses_taken of student
and prerequisites of course is subset_of courses_taken of student
and student count(takes) is less_than 5
then student takes course;

if student year 4
and type of course is equal_to 'restricted_elective'
and student wants_hardware_courses() is true
and course group 'hardware'
and course is not member_of courses_taken of student
and prerequisites of course is subset_of courses_taken of student
and student count(takes) is less_than 5
then student takes course;
```

## C.2 Some Part of the Database of ADVICE

```
Instances of the class COURSE
=============================

Object-id: C001
---------------
   CNO: math101
   REQUIRED_YEAR: 1
   REQUIRED_SEMESTER: fall
   GIVEN_SEMESTER: fall
   PREREQUISITES:
   TYPE: must_course

Object-id: C002
---------------
   CNO: phys101
   REQUIRED_YEAR: 1
   REQUIRED_SEMESTER: fall
   GIVEN_SEMESTER: fall
   PREREQUISITES:
   TYPE: must_course

Object-id: C003
---------------
   CNO: cs101
   REQUIRED_YEAR: 1
   REQUIRED_SEMESTER: fall
   GIVEN_SEMESTER: fall
   PREREQUISITES:
   TYPE: must_course

Object-id: C004
---------------
   CNO: chem101
   REQUIRED_YEAR: 1
   REQUIRED_SEMESTER: fall
   GIVEN_SEMESTER: fall spring
   PREREQUISITES:
```

```
    TYPE: must_course

 Object-id: C005
 ---------------
    CNO: eng101
    REQUIRED_YEAR: 1
    REQUIRED_SEMESTER: fall
    GIVEN_SEMESTER: fall
    PREREQUISITES:
    TYPE: must_course

 Object-id: C006
 ---------------
    CNO: math102
    REQUIRED_YEAR: 1
    REQUIRED_SEMESTER: spring
    GIVEN_SEMESTER: spring
    PREREQUISITES: C001
    TYPE: must_course

 Object-id: C007
 ---------------
    CNO: math110
    REQUIRED_YEAR: 1
    REQUIRED_SEMESTER: spring
    GIVEN_SEMESTER: fall spring
    PREREQUISITES:
    TYPE: must_course

 Object-id: C008
 ---------------
    CNO: phys102
    REQUIRED_YEAR: 1
    REQUIRED_SEMESTER: spring
    GIVEN_SEMESTER: spring
    PREREQUISITES: C002
    TYPE: must_course
```

```
Object-id: C009
---------------
   CNO: cs102
   REQUIRED_YEAR: 1
   REQUIRED_SEMESTER: spring
   GIVEN_SEMESTER: spring
   PREREQUISITES: C003
   TYPE: must_course

Object-id: C010
---------------
   CNO:  eng102
   REQUIRED_YEAR: 1
   REQUIRED_SEMESTER: spring
   GIVEN_SEMESTER: spring
   PREREQUISITES: C005
   TYPE: must_course

Object-id: C011
---------------
   CNO:  cs221
   REQUIRED_YEAR: 2
   REQUIRED_SEMESTER: fall
   GIVEN_SEMESTER: fall
   PREREQUISITES:
   TYPE: must_course

.

Object-id: C014
---------------
   CNO:  ee281
   REQUIRED_YEAR: 2
   REQUIRED_SEMESTER: fall
   GIVEN_SEMESTER: fall
   PREREQUISITES:
   TYPE: must_course
```

```
Object-id: C015
---------------
   CNO:  cs242
   REQUIRED_YEAR: 2
   REQUIRED_SEMESTER: spring
   GIVEN_SEMESTER: spring
   PREREQUISITES: C011
   TYPE: must_course

Object-id: C016
---------------
   CNO:  cs232
   REQUIRED_YEAR: 2
   REQUIRED_SEMESTER:  spring
   GIVEN_SEMESTER: spring
   PREREQUISITES: C007 C009
   TYPE: must_course

Object-id: C017
---------------
   CNO:  ee212
   REQUIRED_YEAR: 2
   REQUIRED_SEMESTER: spring
   GIVEN_SEMESTER: spring
   PREREQUISITES: C014
   TYPE: must_course

.

Object-id: C032
---------------
   CNO:  el123
   REQUIRED_YEAR:
   REQUIRED_SEMESTER:
   GIVEN_SEMESTER: fall
   PREREQUISITES:
   TYPE: nontechnical_elective
```

```
.
.

Instances of the class STUDENT
================================

Object-id: S001
---------------
   NAME: Ali
   YEAR: 1
   SEMESTER: spring
   STANDING: probation
   COURSES_TAKEN: C001 C002 C005
   F_COURSES_LIST: C003 C004

Object-id: S002
---------------
   NAME: Veli
   YEAR: 1
   SEMESTER: spring
   STANDING: satisfactory
   COURSES_TAKEN: C001 C002 C005
   F_COURSES_LIST: C007


Object-id: S003
---------------
   NAME: Selami
   YEAR: 2
   SEMESTER: spring
   STANDING: satisfactory
   COURSES_TAKEN: C002 C004 C005 C010 C011 C014
   F_COURSES_LIST: C001 C003 C007 C008
```

## C.3 The Outputs of ADVICE for Some Executions

ADVICE is executed for the following applications and the following results are obtained:

- All instances of class COURSE and the first instance (S001) of the class STUDENT is used to find the courses advised to the student "Ali" and the system advised courses with id's C004, C006, C007, C008 and C010.
- All instances of class COURSE and the second instance (S002) of the class STUDENT is used to find the courses advised to the student "Veli" and the system advised courses with id's C004, C006, C007, C008 and C010.
- All instances of class COURSE and the third instance (S003) of the class STUDENT is used to find the courses advised to the student "Selami" and the system advised courses with id's C007, C008, C015, C017 and C032.