CONSTRUCTION OF AN INTELLIGENT DATABASE APPLICATION DEVELOPMENT ENVIRONMENT USING OBJECT-ORIENTED CONCEPT

SuHoun Vandy Liu

Certificate of Approval:

[illegible]
William N. Ledbetter
Associate Professor
Management

W. J. Kenny Jih
W. J. Kenny Jih, Chairman
Assistant Professor
Management

[illegible]
F. Nelson Ford
Assistant Professor
Management

Norman J. Doorenbos
Norman J. Doorenbos, Dean
Dean
Graduate School

CONSTRUCTION OF AN INTELLIGENT DATABASE APPLICATION DEVELOPMENT ENVIRONMENT USING OBJECT-ORIENTED CONCEPT

SuHoun Vandy Liu

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

requirements for the

Degree of

Master of Science

Auburn, Alabama

December 9, 1988

CONSTRUCTION OF AN INTELLIGENT DATABASE

APPLICATION DEVELOPMENT ENVIRONMENT

USING OBJECT-ORIENTED CONCEPT

SuHoun Vandy Liu

Permission is granted to Auburn University to make copies of this thesis at its discretion, upon the request of individuals or institutions and at their expense. The author reserves all publication rights.

Vandy Liu
Signature of Author

Dec. 9. 1988
Date

Copy sent to:

Name	Date

VITA

SuHoun Vandy Liu, son of Jih-Hung Liu and Kwei-Yun Li Liu, was born June 14, 1962, in Miao-Li, Taiwan, Republic of China. He attended Chang-Kwang Primary School and graduated from the Pan-Chou High School in 1980. In September, 1980, he entered Tung-Hai University and received the degree of Bachelor of Business Administration in June, 1984. In September, 1986, he enterd Auburn University for graduate studies in management. Mr. Liu has held the position of graduate teaching assistant in the Department of Management at Auburn University.

THESIS ABSTRACT

CONSTRUCTION OF AN INTELLIGENT DATABASE APPLICATION DEVELOPMENT ENVIRONMENT USING OBJECT-ORIENTED CONCEPT

SuHoun Vandy Liu

Master of Science, December 9, 1988
(B.B.A., Tung-Hai University, 1984)

178 Typed Pages

Directed by Dr. Kenny W. Jih

Throughout the world, managers are beginning to recognize the potential of computers and information processing technology to help them cope with today's complex business world. When the demand for new or expanded computer-based information systems grows rapidly, it far exceeds the capability of present DP organizations to meet this demand. One feasible solution is to allow end-users to function as their own developers. However, to accomplish this transfer of application development from DP professional to DP user, support must be provided to these end-user developers.

One promising technical approach toward this end is to provide them an intelligent development aid. Such a development aid should contain general knowledge about system development to provide assistance to it's users, and it must be easy to use. Two techniques have been recognized to have great potential for the development of such an intelligent development aid. They are the knowledge-based technology and the object-oriented concept.

The purpose of this thesis was to explore the feasibility of developing an intelligent development aid for end-users by using the object-oriented concept and knowledge-based technology. This study investigates this feasibility by developing a conceptual model for such a development aid. A prototype was implemented to test the validity of the conceptual model. The prototype demostrates that the conceptual model can be implemented and that the resulting system does exhibit the kind of interaction process desired.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. BACKGROUND AND INTRODUCTION

Introduction

The basic purpose of information system installations is to effectively support needed information resources for the organization. Recent research has indicated that information systems are fast becoming indispensable allies in marketing, customer services, management, strategic planning, and many other tasks (Bryce, 1987; Gerstein and Reisman, 1982). The need for information systems grows substantially as more complex systems are required for users' more sophisticated problems.

As the demand grows, the MIS department is under great pressure from their users. As users become more sophisticated and systems become more essential to the success of the organizations, the need for reliable, high-quality software escalates. This need has created a dilemma of unprecedented proportions for the MIS department. The major problems for MIS departments in most organizations are discussed as follows.

1. Limited human resources. A study conducted by the Department of Defense estimated that by 1990 there could be a deficit of as many as 1 million software developers

(Case, 1986). Qualified system development personnel (including analysts, programmers, project managers) are rare and very expensive resources. Since a massive expansion of MIS professionals is impossible, many MIS departments and organizations will fail to meet the booming demands for their services.

2. Cost overrun. While the hardware has decreased in price and increased in performance, the software has steadily increased in cost. Unlike hardware development, today software development is largely composed of manual tasks performed by system developers. Unstandardized procedures are employed and no mass production is used to reduce the cost of development. Therefore, costs of systems can be unaffordably high.

3. Growing backlog. A critical aspect of the software crisis is the large backlog of application development requests. There are two primary reasons for this backlog. On the supply side, the productivity of system developers has not increased significantly in the past. On the demand side, as more users realize the benefits of computerized solutions to their problems, more requests are made. The backlog grows as the supply continues to fall behind demand. In addition to the documented backlogs, there are invisible backlogs resulting from the requests which are never submitted. Realizing that the MIS department is having difficulty meeting the demand, many users are now opting for small computers and are attemping to computerize

their applications by themselves. Many problems associated with inefficient resource allocation as well as lack of control have occurred with regard to this situation.

4. User dissatisfaction and system failure. User dissatisfaction occurs when the system delivered to the user fails to meet their needs and expectations. These system development failures are usually caused by miscommunication between the developer and the user. A frequently quoted statistic is that 50-80% of the budget for a system development project is spent in maintaining the system after it has been delivered to the user (The Report of Alvey Committee HMSO, 1982; Ramamoorthy, 1984).

Because of the concern for these problems, improving information systems development productivity has become a critical issue in MIS research. A 1984 survey revealed that MIS professionals identified productivity as one of the ten most urgent issues facing organizations (Dickson, 1984). During the past decade, business, government agencies, and academic institutions have extensively investigated various issues of system development productivity. As a result, two approaches have been identified as being of great potential. One approach is to make the MIS professionals more productive. The other approach is to allow end-users to function as developers of their own applications.

The first approach aims to increase the productivity of MIS personnel by providing more useful tools and more practical development methodologies. The efforts in this direction have met with a fair measure of success. A wide variety of technologies has been developed for the MIS personnel to be able to communicate better, to test the prototype easier, and as a result, to develop systems faster and cheaper. Examples of these productivity enhancement technologies are codes generators, structured analysis, and fourth-generation languages, etc.

The second approach attacks the problem from a different angle: end-users are allowed to develop their own systems. Empirical evidence supports the proposition that sufficient user involvement leads to more effective system development (Kasper, 1985). During the past ten years, capabilities of technologies have been improved to a degree that users may function as system developers. Since end-users know their business better than computer professionals, they do not have to spend so much time in analyzing and specifing their information requirements. The need for the programmer "middle man" is thus reduced.

Not only the creation but also the modification of the system may be performed by end-users. This helps relieve the MIS departments' applications development workload, and it helps distribute the maintenance burden (Ronald, 1987). However, in order to transfer part of the application development responsibility from the MIS department to application users, both hardware and software tools must be provided.

Today, the implementation of desktop computers in the business community contributes a positive hardware factor for end-user application development. Desktop computers provide more local control over and easier access to much of the desirable computing power for a relatively low cost (Gibson, 1988). However, developing an application system is a complicated task. One problem end-user developers have is that they do not posses the knowledge required for development of the system. Fortunately, recent research has created a variety of concepts and techniques capable of making system development tasks easier and more understandable to end-users. Among these, two have been recognized to have great potential. They are the knowledge-based technology and the object-oriented concept (Schmucker, 1986; Simon, 1986).

It is widely recognized that knowledge-based systems can play an important role in commerical DP systems development (Barder et. al., 1987). Because of the unique system architecture, a knowledge-based system can store the specialized development knowledge that the end-users do not have. These knowledge bases then can be made available to end-user developers, resulting in an interactive system development environment. A number of knowledge-based systems had been developed for this purpose (Bader, Hannaford, Cochran & Edwards, 1987; Choobineh, Mannino, Nunamaker & Konsynski, 1988; Hull, 1987; Karimi, 1988; Madhavji, 1988). However, these systems were designed primarily for MIS professionals. A considerable amount of training is required for these systems to be used by novices. In a survey conducted by Rivard and Huff (1985), the training requirement was found to be one of the main problems in making "everybody ... his own programmer" (Sammet, 1969).

The object-oriented concept relates to both the object-oriented design paradigm and the object-oriented implementation technique. Object-oriented (O-O) design is viewed as a software decomposition technique (Meyer, 1987). Unlike classical (functional) design, it bases the modular decomposition of a software system on the classes of objects the system manipulates. O-O design decomposes a system in a way that closely matches the model of reality which the users already know. Thus O-O design can be understood by nonprofessional users easier than the traditional functional decomposition approach (Feuche, 1988). Furthermore, O-O design leads to a new approach to system implementation -the O-O implementation. Booch (1985) proposed that, given a rich set of reusable object components (objects or object classes), implementation may proceed via composition of these object components.

When the knowledge base and the O-O concepts are applied to the development of an end-user application development aid, many benefits can be achieved. Literature on these topics abounds. In Chapter Two some of the most significant research will be reviewed.

Problem Statement

A general problem we are facing today is the need to significantly increase our productivity in producing quality systems to meet the tremendous demand. Both management and technical measures must be explored in this endeavor. One promising technical approach is to provide an intelligent environment within which end-users may develop their own applications. Most existing tools developed to improve system development productivity deal primarily with the implementation stage in the development life cycle. Since most problems require some degree of design, such types of tools are limited in their functions. An intelligent system development aid not only will automate the implementation task, but also is capable of automating much of the design function.

Three issues must be considered in developing an intelligent aid for user application development. First, the system must contain general knowledge of system development. For instance, in a relational database environment, normalization of relations is a critical task in the design stage. An intelligent system development aid should be capable of performing database normalization with only a minimal amount of user interference. Second, the problem domain must be well bounded in the sense that most activities must be identifiable and representable. The more the system knows about the activities in the domain, the more powerful the system could be in providing intelligent assistance. The third issue is user interface. Any system aimed at end-users must be easy to use. This is even more important for a system dealing with both design and implementation. An easy-to-use system provides more than just friendly interaction prompts and messages. The views of real world contructs used by the system must also have a natural correspondence with those of the users. For example, an input screen is often regarded as an inseparable object by the user. If a screen format is also viewed by the system as an object, rather than as a collection of relatively independent entities, there would be a natural communication channel between the system and the user.

Objective of This Research

The main goal of this research is to explore the possibility of applying the O-O concept in developing a knowledge-based development aid. The system's main function is to provide an environment within which novice users can design their own applications by specifying objects that correspond to the real world constructs of their problem domains. Once the design phase is finished the user can proceed into the usage phase. With this system's assistance the user acts both as a system developer and as a system user.

The system developed in this study represents a fundamentally different approach to applications development. Figure 1 depicts the model of the application development environment in a traditional approach. In this environment, system developers translate users' problem descriptions into application systems in a specific computer language. The programs are then compiled or interpreted into machine code and executed by computers. The model proposed in this research is shown in Figure 2. In the End-User Application Development Environment, users' specifications may be directly translated into executable code by the knowledge-based application development aid.

In order to construct the End-User Application Development Environment, this study first develops an architecture for an automatic application development environment. The environment consists of a knowledge-based development assistance component which was developed using the O-O concept. The architecture will be discussed in the third chapter. According to McCracken (1980), an end-user application development aid should have a natural syntax, should be interfaced with a database management system, should provide a query language plus a report generator, and should have the ability to insure data integrity and validity. In order to test the proposed architecture, a prototype of an end-user application development aid has been developed using this architecture.

Research Hypothesis and Research Methodology

The hypothesis for this research was that it is feasible to develop an object-oriented, knowledge-based

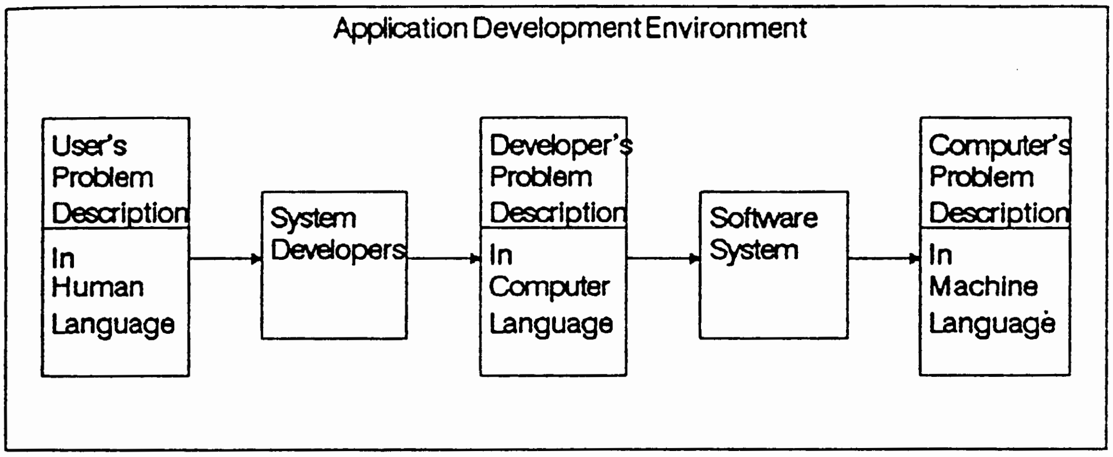

Application Development Environment
User's Problem Description
In Human Language
System Developers
Developer's Problem Description
In Computer Language
Software System
Computer's Problem Description
In Machine Language

Figure 1. Traditional Application Development Environment

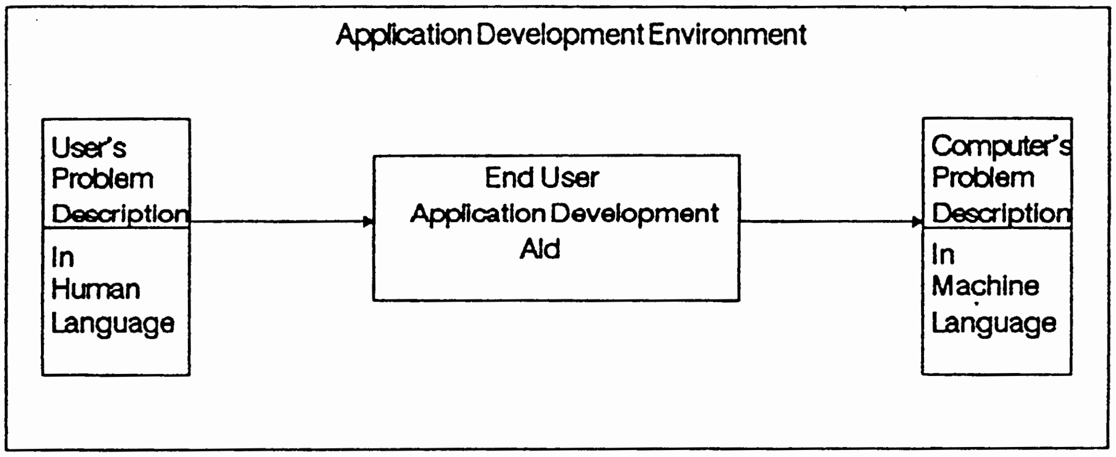

Application Development Environment
User's Problem Description
In Human Language
End User Application Development Aid
Computer's Problem Description
In Machine Language

Figure 2. End-User Application Development Environment

system for end-users to use in designing and implementing their own applications. This hypothesis was tested by developing a conceptual model and implementing a prototype system based on that model to demonstrate the function of the system. The focus was placed upon the feasibility rather than the efficiency and effectiveness aspect of the issue. Statistical testing for efficiency and effectiveness evaluation for this system was not feasible in this case. First of all, if statistical testing were to be used, it would be necessary to duplicate the development many times so that enough sample points would be available for statistical testing. Also, because of the different nature of this system, the intelligent system development environment proposed here supports a new development approach. That is, system development in this environment has a different meaning when compared to system development in a traditional development environment. In this intelligent system development environment, system development is a process of object specification while in the traditional development environment it is a process of processing description.

Significance

The development of a successful application system is usally complicated by an unfortunate fact: The paradigm (model of realty) of the end-user and the internal model of the software system are substantially different (Meyer, 1987). Meyer (1987) and Booch (1986) believe that end-users view their world in terms of objects they encounter in their work environment. They employ abstraction and tend to develop models for their work environment by identifying the objects and operations that exist at each level of interaction (Booch, 1986). For example, a student views a world as consisting of courses, faculty, and so forth. Software systems traditionally hold the world view in terms of functions or processes in the problem domains. For example, a system probably views the student world in terms of "adding_a_course", "dropping_a_course", and so forth.

The main contribution of this research is that the proposed development model frees the end-user developers from following the footsteps of the professional developers. Capturing the abstract knowledge of the environment in the object-oriented paradigm makes the assumptions, policies, and rules of the work environment formal and explicit, instead of embedding them in procedures, as is the case in traditional function/process-view models (Karimi, 1987). By applying the O-O approach, the end-user's models of reality for their work environment can be implemented into reusable object classes in the development environment. Thus, the end-user developers can design and develop their own application systems based on their own problem descriptions. For example, if student, faculty and course are the object types in a student's model of reality, these can be captured and implemented into three object classes in the development environment. When this student makes a problem description, student, faculty and course will be used and specified in his description (e.g. who is the faculty that teaches the course COBOL). Then a process can be triggered by the development aid to search for a specific object in the object class "faculty" that has a certain relationship (teach) with an object (COBOL) in the object class "course".

The second contribution of this research is that it proposes a way to design a high level user interface in an automatic development system. Applying O-O development can simplify the normally complicated task of designing a user-friendly interface. In an object-oriented system, the interface directly matches the internal content of the system. This could significantly reduce the need for user training as well as the fear of computer usage. Also, since the system is object-oriented, a graghic-based interface can be implemented more easily. For example, student, faculty and course can be represented by graphic symbols in the interface. If functional decomposition were employed, the system would have to incorporate functions such as "teaching_a_course" and "taking_a_course". Any user specification in this interface will have to be translated into "teaching_a_course" and "taking_a_course" functions before they could be incorporated into the system.

The third contribution of this study is that it proposes a way for organizations to manage their software assets. In the End-User Application Development Environment object classes will facilitate the sharing and reuse of program code. Application systems developed by different users are implemented by sharing the object classes. Also, the O-O implementation protects the independence between the design phase and implementation phase as the system evolves. Thus, end-users and MIS professionals can share the responsibility of maintenance. The independence allows implementational changes (changes on the object class library) to be made by MIS professionals while the design changes (changes on the specifications for the application systems) are made by users.

Organization

The remainder of this reseach is organized into four chapters. Chapter II is a review of the literature relating to the construction of an end-user development aid. Chapter III presents the conceptual model which is proposed by this study. In Chapter IV, the conceptual model is applied to a database application to develop a prototype system. Finally, Chapter V summarizes the conclusions of the study and outlines recommendations for further research.

II. LITERATURE REVIEW

This research investigates the feasibility of developing an end-user application development aid using knowledge base techniques and object-oriented concepts. The areas involved include end-user computing, intelligent system development aid, and the object-oriented development paradigm. The following sections present an overview of these subjects. The first section provided a discussion of the end-user developers. The next section reviews studies concerning intelligent development aid. The last section of this chapter presents an overview of object-oriented concepts used in the system development.

End-User Computing

In an effort to design more effective systems, information systems managers are constantly searching for new management ideas and new technologies. Although the development of new assistance tools for system development can benefit the productivity in system development, such tools are currently seen by management in many organizations as only a part of the overall picture. The basic philosophy about system development is changing in those organizations to allow individuals with little or no formal DP training to develop and use their own computer-based applications (Hughes, 1988). McLean described this approach as follows:

> Another approach is to allow end-users to function as their own developers. In this way, the programmer "middle man" is eliminated and users can create and modify their own applications as need arises (McLean, 1979:37).

Discussions on end-user computing abound in the academic literature and popular press. For example, back in 1969, Sammet (1969) argued in favor of making everyone his own programmer. Boehm (1973) foresaw a trend toward the day when most problems could be programmed by a user "in less than an hour... with one day of specialized trainning". Gibson and Hughes (1987) discussed the role of design and development environment in the overall end-user work environment. Dolotta et. al. (1976) suggested that this approach appeared to be the only way the growth of the data processing industry could be sustained. Leitheiser and Wetherbe (1986) stated the reasons that end-users would choose to do their own computing:

1. Lead times on development requests are shorter.
2. End-users have more control over system development and use.
3. Services are not available from the MIS department.
4. MIS department procedures are not appropriate for small applications.
5. The MIS department is not perceived as being concerned about users' needs.

6. End-users want to learn about computing.
7. End-users gain more flexibility.
8. The information systems developed better meet users' needs.
9. Development costs are lower.

Besides the advantages for users, Leitheiser and Wetherbe (1986) also identified the benefits for the MIS department. They were

1. The shortage of systems development personnel can be relieved.
2. End-user computing allows users to avoid going through the time consumming and error-prone process of communicating requirements to an outside developer.
3. The responsibility of the MIS department for successful system implementation is removed.

The report of a survey in December of 1985 (Journal of Systems Management Report, 1985) indicates that after adding the development function into a group of end-users' work environment, those users believe that their work environment now allows them to "complete more work in a given amount of time, freeing them for other activities." Furthermore, another study (Benson, 1983) shows that the increased productivity due to the use of these development functions indicates that the benefits largely outweigh the cost.

Other studies show that end-users are capable of becoming good system developers. For example, Kozar and Mahlum

(1987) demonstrated that system design tasks could be done successfully by end-users. In their study a project was conducted in which users, with tutoring and guidance, perform as system analysts. In the study, the end-users not only accomplished the task, but also did it very efficiently.

To help end-users function as system developers, proper development support is needed. In a 1979 article, McLean discussed the key variables that must be understood in the transfer of application development from DP professional to DP user. His work provided excellent background knowledge about the relavant factors in supporting end-users to function as developers.

An empirical study conducted by Rivard and Huff (1985) in ten large Canadian firms showed that proper training is one of the most important types of support for end-user developers. In their study, nearly 75% of the survey respondents had at least some exposure to a programming language. But most users interviewed indicated that they required relatively longer learning periods and considerable practice to become proficient with the software tools. A study conducted by Kozar and Mahlum (1987) suggested that more time should be spent teaching user groups how to define problems or to describe the environment of the problem than in applying technology to the problem.

Assistance during the system development process is another important support for end-user developers. McLean recognized the value of applying system prototyping in end-user developers' development process. Gibson and Corman (1987) identified the possible causes of errors in the end-user programmers' development tasks. Their recommendations for supporting end-users in dealing with these problems are, first, to provide sufficient aids such as information center, computer-assisted instruction or other tutorials, seminars, and continuing education; and second, to use the technological advances in the areas of natural languages and artificial intelligence to provide the ability to reduce the risk of errors.

Intelligent System Development Aid

System development is a knowledge-intensive activity. It has been suggested that knowledge base technologies are a promising approach to significantly improving the efficiency of this activity (Goldberg, 1986). The need for an intelligent system development environment was recognized in the mid-1970s as a result of continual problems with software development (Karimi & Konsynski, 1988). For example, Goldberg (1986) suggested that a knowledge-based development aid would increase productivity of the system developers by (1) providing a rapid prototyping capability, (2) reducing the cost of development and maintenance, and (3) increasing reliability. Simon (1986) explained the necessity of automating system development function and the potential of artificial intelligence in this area by stating that:

> It is not a question of whether we want to automate more of this (system development) process, and since part of the (development) system we want now to automate is a highly unstructured part of the process, it is not really a question of whether we want to use artificial intelligence methods in software engineering: it is a question of whether artificial intelligence is powerful enough, whether we yet know enough about artificial intelligence, or whether it is advanced enough to really help us (Simon, 1986; 726).

An important issue that must be considered when assessing the potential of knowledge-based systems in a particular domain is whether the application area is mature enough to be the basis for a knowledge-based system (Bader, Hannaford, Cochran & Edwards, 1987). The area of software engineering, when compared with more mature areas such as the legal, medical or geological domain, has had a short evolutionary span. However, a number of sophisticated systems with training, engineering, and heuristic knowledge have been constructed. For example, Vitalari (1985) presented a summary of the knowledge for the system analysis, and Goldberg (1986) presented a framework for the organization of the knowledge for program development. For the development of a knowledge-based system development aid, the traditional system development life cycle (SDLC) model provides useful guidelines. Different types of systems may be developed to address different portions of the life cycle. Some development aids that provide assistance in different stages of the SDLC are reviewed below.

A number of knowledge-based tools are now available for the system analysis and requirement specification. For example, IS-DOS (Teichroew & Hershey, 1977), SREM (Alford, 1978), and SPECIF (SPECIF, 1984) are three automated tools for software requirement specification. With SPANNER (Aggarwal, Babra & Meth, 1988), a software design environment, the user can formally produce a specification of a distributed computing problem, and then verify its "correctness" through reachability analysis and simulation. Henry (1979) defined and validated a set of software metrics which were based on the measurement of information flow between system components. The metrics were demonstrated to be useful in design and development of the UNIX operation system. Giddins and Colburn (1984) developed rules to quantify notions of good design based on the connectivity and the complexity of the components. Karimi and Konsynski (1987; 1988) applied metrics to the design phase and built an automated software design aid which would derive a structured modular design from the logical model.

In the area of database design, a forms-oriented model was proposed to be of great value for construction of an intelligent aid. Properties of forms were used as the inputs to generate requirements specification for the system. Sue, Wong, and Lum (1982) reported a forms-oriented approach to database design. However, no computer-assisted database design aid in support of their standardized form was described. Batini, Demo, and Leva (1984) proposed a methodology for deriving a conceptual schema from a collection of forms. Their approach was to derive the global schema in a bottom-up fashion starting from subparts of forms. For each form, the designer compiled a data glossary of the form fields. These form fields then were grouped into entity types and were used to generate the entity type for each form.

Holsapple, Shen, and Whinston (1982) implemented the form-oriented approach for database design. Their expert system designed database schema from business reports. Each report was formalized into a report schema and consequently transformed into a database schema. Knowledge about the design was represented as production rules which could be used as a consultation system by the database designer. Choobineh, Mannino, Nunamaker and Konsynski (1988) developed an expert system for logical database design. The system created an entity-relationship dialog by analyzing a collection of forms. The conceptual schema was created by incrementally integrating related collections of forms. It applied a collection of rules against a form definition database to interactively guide a designer in constructing an Entity Relationship Diagram for the database.

Various Computer-Assisted Software Engineering (CASE) systems such as report generator, screen generator and some very high level computer language have been proposed to help automate the system implementation stage in the SDLC.

Sample works include Luo and Yaos' work (1981) in developing a language for office information processing based on examples. There also have been a number of automated tools for database application implementation. For example, the form-oriented information management model developed by Tsichritzis in 1982.

Automatic programming is a well-investigated field in improving the system development productivity. Madhavji (1988) proposed a basis for automatic programming environments. Users in this programming environment may develop software programs based on objects called fragtypes. Fragtypes range from a simple expression type to a complete subsystem type. MUPE-2, which is currently under development at McGill University, is a programming environment that uses fragtypes as the building blocks. Programmers of the MUPE-2 environment may assemble their programs from fragtypes (Madhavji, 1988; 1986). Meyer suggested that the object-oriented design could be used as a decomposition technique to form reusable objects. Henderson developed "The Trillium User Interface Design Environment" which provided a means of interactively designing user interfaces through the exploitation of some key abstractions from the language of interface design and used the notion of functioning frames as the basic representation style (Easterby, 1987).

Because of the significant difference between design and implementation, few studies have been conducted on systems with capabilities of supporting both design and implementation tasks. The development of Intellipse is one of them. Intellipse is a knowledge-based tool for supporting both the design and implementation of commercial data processing systems (Bader, Hannafold, Cochran and Edwards, 1987). Also, some of the intelligent design aid systems discussed previously, such as the one reported by Hull and Metcalfe (1987), exhibited the possibility of integrating with some high level languages for system implementation.

Studies have demonstrated that knowledge-based systems can provide effective support for the operation and maintenance stages. Some intelligent systems were developed to produce system documentation. For example, Munro (1983) constructed a system, SADIST, which functioned as an interactive editor in a computer-aided documentation system. Hull and Metcalfe (1987) outlined the design and development of a software tool for the generation of SADT (Structured Analysis and Design Technique) diagrams implemented on a Fortune microcomputer. Studies show that the maintenance of computer systems could benefit greatly if the process knowledge could be captured and used in order to reason about the consequences of changing conditions or requirements (Dyer, 1984; Hurst, Frewin & Hamer, 1985; Leung, 1985). Dunning (1985) developed an expert system that supported rapid prototyping of conventional software. Dhar and Jarke (1988) proposed a formalism called REMAP which accumulated design process knowledge to manage system evolution. REMAP acquired and maintained dependencies within the design decision made during a prototyping process, and was able to learn general domain-specific design rules on which such dependencies are based.

Object-Oriented Development Paradigm

Description of Object-Oriented Concept

This section introduces basic concepts of programming with objects and reviews some of the major works related to the object-oriented development paradigm. Simply stated, the object-oriented development paradigm is an approach to system development in which the decomposition of the system is based upon the concept of objects. Many of the ideas behind object-oriented programming have roots going back to SIMULA (Stefik & Bobrow, 1984). The first substantial interactive, display-based implementation for object-oriented paradigm was SMALLTALK (SMALLTALK, 1984). Rentsch (1982; 51) predicted that "object-oriented programming will be in the 1980's what structured programming was in the 1970's".

Object-oriented development is fundamentally different from traditional functional methods. Using the object-oriented concept, the primary criteria for system decomposition is that each module in the system represents a component in the overall problem domain instead of a function the system performs (Meyer, 1987). All information in an object-oriented system is represented as objects. Each object responds to certain messages that are sent to it. Associated with each message is a method in the object that describes how the object should react when it receives the message. As an object responds to a message, the corresponding method is invoked and the associated action taken. The components (physical or logical) in the problem domain are simulated by means of objects and their methods in the object-oriented system. In developing an object-oriented system, one may identify the objects by looking at the nouns within the requirement specification. Methods within each object can be identified by the verbs associated with the object in the problem description. Stefik and Bobrow (1986) provide a good comment on the themes and variations in object-oriented programming.

There are a number of object-oriented programming languages in use today. Examples are SMALLTALK, LOOP, and C++. Object-oriented programming features can be added to nearly any conventional programming language by grafting a small number of new syntactic features alongside the existing capabilities of the language. The new language retains the efficiency and compatibility of the base language, but it provides the reusability and productivity of an object-oriented programming language (Cox, 1986). Some AI languages such as LISP and PROLOG appear to be appropriate for the object-oriented approach. Object-oriented programming has already been adopted extensively in some LISP-based systems (Cannon, 1982). The inherent rule-based processing capability of PROLOG also makes it a good vehicle for the use of the object-oriented approach (Stabler, 1986).

Object-Oriented Paradigm For System Development

The object-oriented approach provides system developers with a new system decomposition technique and a new system building approach. The concepts of object and object-oriented programming represent a promising unifying paradigm for the design of knowledge-based systems, database systems, and programming language (Carlo et. al., 1986). Booch (1986) detailed the major steps for the development process in an object-oriented development environment:

1. Identify the objects and their attributes.
2. Identify the operations suffered by and required of each object.
3. Establish the visibility of each object in relation to other objects.
4. Establish the interface of each object
5. Implement each object.

Traditionally, system development has decomposed the system by functions, routines, or procedures. Stein and Maier (1986) argue that the traditional decomposition is imperative in nature and concentrates on the major processes of a system and ignores the objects that perform or suffer these actions or processes (Karimi, 1986). Booch compared the object-oriented and the traditional functional approaches and concluded that "the object-oriented decomposition closely matches our model of reality. On the other hand, the functional decomposition is only achieved through a transformation of the problem space" (Booch, 1986; 212).

In constrast with the functional approach, the object-oriented approach captures the abstract knowledge of the environment in the object-oriented specification. Borgida (1985) believes that the object-oriented approach makes the assumptions, policies, and rules of the application environment formal and explicit. This leads to systems that better conform to the real requirements. Bohem-Davis and Ross (1984) reported that the object-oriented approach seemed to produce a better specification for systems with natural concurrency and real-time processing. Kroenke (1987) suggested that since the object-oriented design decomposed the system directly from the user's view, it is a powerful tool for database design.

In addition, object-oriented decomposition leads to better maintentability and reusability. Booch found that there is a basic relationship between reusable software components and object-oriented development: "reusable software components tend to be objects or class of objects" (Booch, 1986; 220) and are implemented as modules. Meyer explained this as follows:

> The top-down functional approach is probably adequate if the program you are writing solves a fixed problem once and for all. But the picture changes when you take a long-term view, for what

> the system will do in its first release is probably going to be a little different from what you think it will do at requirements time, and very different from what it will do five years later, if it survives that long.
> However, the categories of objects on which the system acts will probably be more or less the same. An operating system will always work on devices, memories, processing units, communication channels, and so on; ...
> Thus it is wiser in the long term to rely on categories of objects as a basis for decomposition (Meyer, 1987; 53).

The object-oriented paradigm is also viewed as a system-building tool. Meyer (1987) noticed that object-oriented development blurs the distinction between design and implementation. Booch (1986) proposed that, given a rich set of reusable software components, implementation should proceed via composition of these parts, rather than by further decomposition. He concluded that object-oriented development is amenable to automated support. Cox (1986) used a concept which he labeled Software-IC to specify the nature of such support. The name Software-IC is normally used to emphasize the parallel with the way hardware engineers build circuits from a stockroom of generic, reusable silicon chips. Each Software-IC implements a class of objects, such as a class of Envelopes or a class of FileFolders. It is a package of programming effort that is independent of the specific job at hand and highly reusable in future jobs.

Chapter Summary

This chapter has reviewed some of the research on end-user computing, intelligent system development aid, and the object-oriented development paradigm. The literature review reveals that there are a variety of ways to aid users to function as system developers. Two conclusions result from this review. First, with proper assistance, end-users are capable of developing their own application systems. Secondly, an intelligent development aid and the object-oriented concept are capable of supporting such assistance for end-users.

The purpose of this study is to investigate the feasibility of developing an intelligent application development aid based on the object-oriented concept. There is an apparent need for conceptual and implementation research on such an object-oriented intelligent application development aid. In the next chapter, a conceptual model is presented for the development of an end-user application development aid, and a prototype system which implements this conceptual model is described in Chapter IV.

III. DEVELOPMENT OF THE CONCEPTUAL MODEL

Introduction

The literature review presented in the preceding chapter identified two limitations in making end-users as developers of their own application systems. These limitations are discussed as follows.

1. Traditional system development tools support functional decomposition which requires transformation of the problem space into a collection of functions. This transformation demands a sufficient degree of professional expertise which end-users normally do not have. This requirement makes these tools usable only by specially trained system developers. Even with tools that are intended for end-users, a considerable amount of time for training and learning is required (Rivard & Huff, 1985). An example is the ASSIST interface in dBase III Plus. It is assummed to be easier to use than the command mode (dBase III Plus, 1986). However, when the interface is needed as an end-user database development tool, the problem described above remains. For users of any database system (manual or computer-based), the model of reality is composed by objects such as files or folders that they

encounter at work. When dBASE is used to develop their applications, users first have to learn to view their database in terms of dBASE functions and commands before they can actually develop any database application.

2. Under the traditional system development approach the program codes are constructed by functions, procedures, or routines. The limitation with the functions, procedures and routines is that they do not provide much flexibility because they force programmers to decide on too much detail too early in the system development process (Meyer, 1987). As a result these program codes tend to have low reusability. This leads to the waste of valuable resources because most end-user applications usually have high similarity (Wartik & Penedo, 1986; Hall, 1987; Kamel, 1987; Prieto-Diaz & Freeman, 1987; Biggerstaff & Richter, 1987).

This chapter describes a conceptual model for the end-user application development aid developed in this study. The model was formulated in a way that the limitations outlined above may be alleviated, if not eliminated. Specifically, the model specifies an architecture for an end-user application development aid called Intelligent Database Application Development Aid (IDADA). The way that IDADA alleviates these limitations is explained below.

IDADA represents an integration of three distinct technologies. First, the object-oriented concept was used for the decomposition and implementation of the application systems. Second, the knowledge-based concept was employed to provide intelligent design assistance to the end-user. That is, the knowledge base component in the IDADA provides users with portable design knowledge. The third approach is the use of a dictionary to coordinate all the activities in developing the application systems. Most of the existing development aids generate program code for the application systems. The dictionary approach, in contrast, has two advantages. First of all, the dictionary is nonprocedural, a key feature for rapid prototyping. Secondly, the dictionary approach allows multiple application systems to share generic programs in object classes. The program generation approach generates programs which can only be used by certain application systems.

Description of the Intelligent Database Application Development Aid

Architecture of the Intelligent Database Application Development Aid

Figure 3 portrays the structure of the End-User Application Development Environment with IDADA. The environment includes two subenvironments - the knowledge-based application development subenvironment and the interactive application operation subenvironment. These two subenvironments interface with each other through the Object Specification Dictionary.

The knowledge-based Application Development Environment differs from the traditional application development

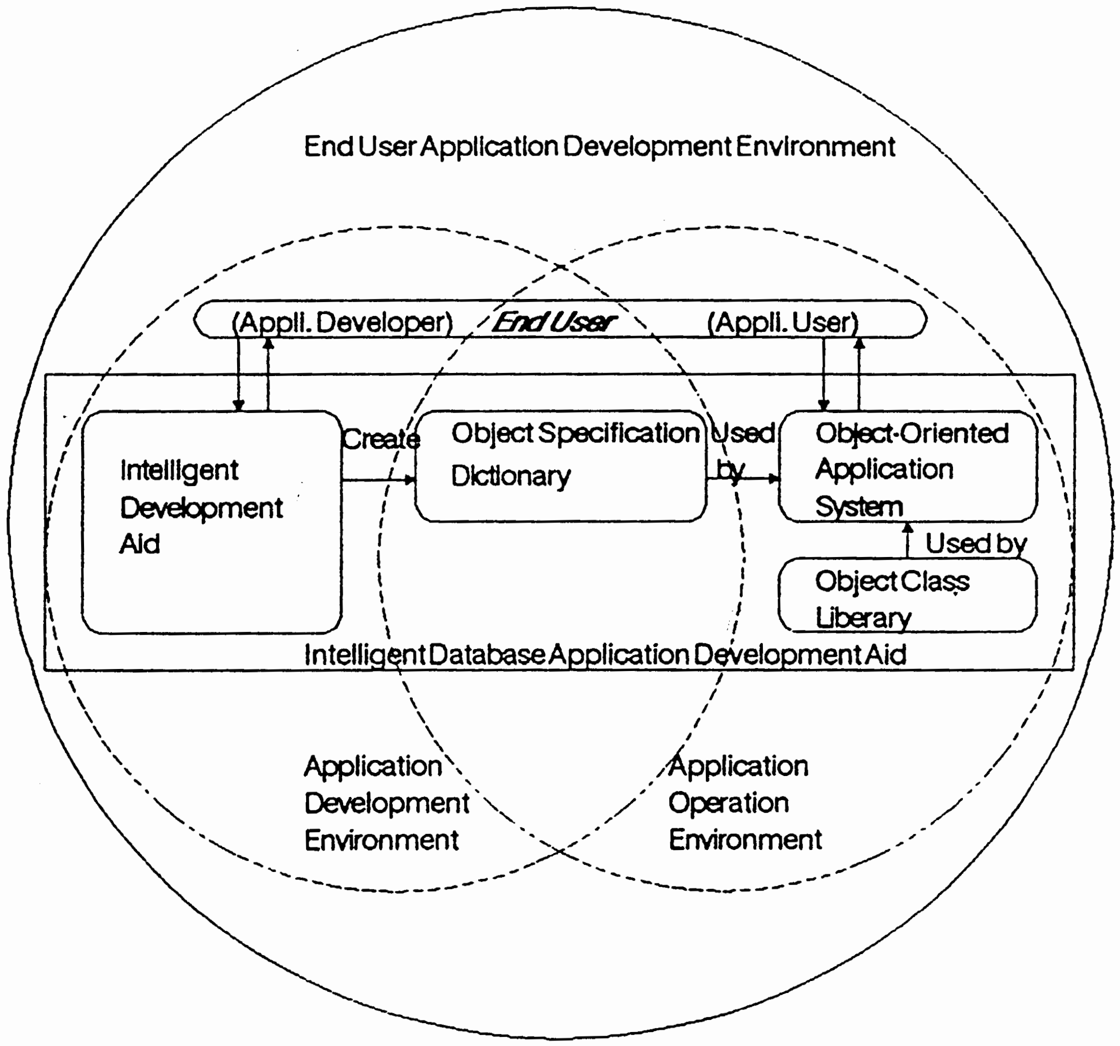

End User Application Development Environment
(Appli. Developer)
End User
(Appli. User)
Intelligent Development Aid
Create
Object Specification Dictionary
Used by
Object-Oriented Application System
Used by
Object Class Liberary
Intelligent Database Application Development Aid
Application Development Environment
Application Operation Environment

Figure 3. End-User Application Development Environment

environment in that the application systems are designed and developed by users with the assistance of a knowledge-based system. Users interact with the Intelligent Development Aid to specify objects in the problem domain. The specifications of these objects are stored in the Object Specification Dictionary. The input-processing-output relationship of this portion of the environment is shown in Figure 4.

In the Application Operating Environment, users interact with the Object-Oriented Application System to perform their normal daily work with the computer. The Object-Oriented Application System is constructed by consulting the Object Specification Dictionary which is created in the Application Development Environment and the Object Class Library of the IDADA. The input-processing-output relationship in the Application Operating Environment is shown in Figure 5.

Intelligent Development Aid

The major role of the user is generally considered to be using application systems to support their normal functioning in the business. This also applies to the users in End-User Application Development Environment. However, these same users expend time and effort, perform design and development roles, and produce results of design and development activities (Gibson & Hughes, 1988). The Intelligent

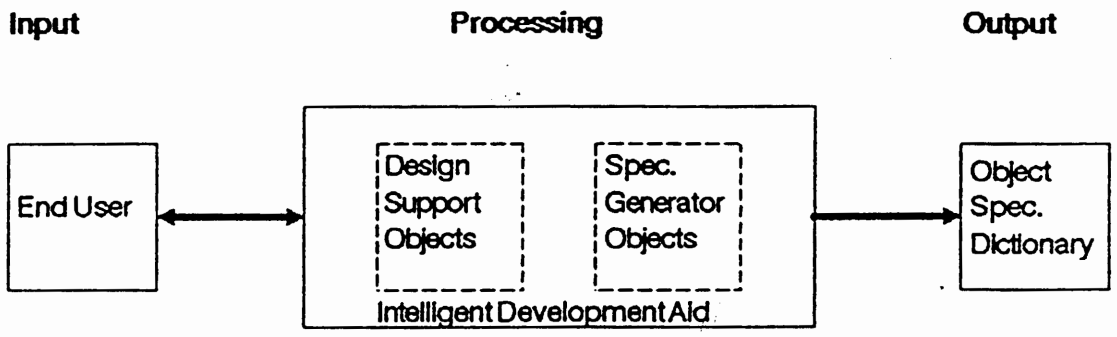


Figure 4. Application Development Process in Application Development Environment

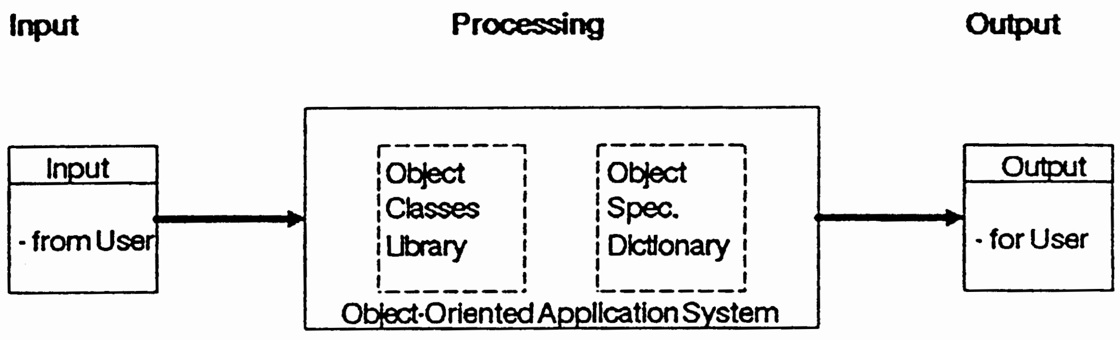


Figure 5. Application System Processing in Application Operating Environment

in Development Aid (IDA) is a knowledge-based development assistance system. It is comprised of two types of components: Design Support Objects and Specification Generator Objects. The IDA provides an object-oriented application development environment. Figure 6 depicts the structure of the IDA. The Design Support Objects contain design knowledge to assist user developers in formulating a proper design. They work interactively with the user developer to generate proper designs for the application system. The resulting design is then sent to the Specification Generator Objects, which contain the knowledge about the format of the object specifications, to generate the Object Specification Dictionary.

Object Specification Dictionary

The Object Specification Dictionary (OSD) is the ouput from the Intelligent Development Aid and the input to the Object-Oriented Application Systems. Communications between the Application Development Environment and Application Operating Environment take place through the OSD. This interface between the two environments provides an independence between the design (specified by the OSD) and programming tasks of an application system. In this way, the generic programs employed by certain application systems can also be used by other application systems with different OSD. In the IDADA those generic programs are

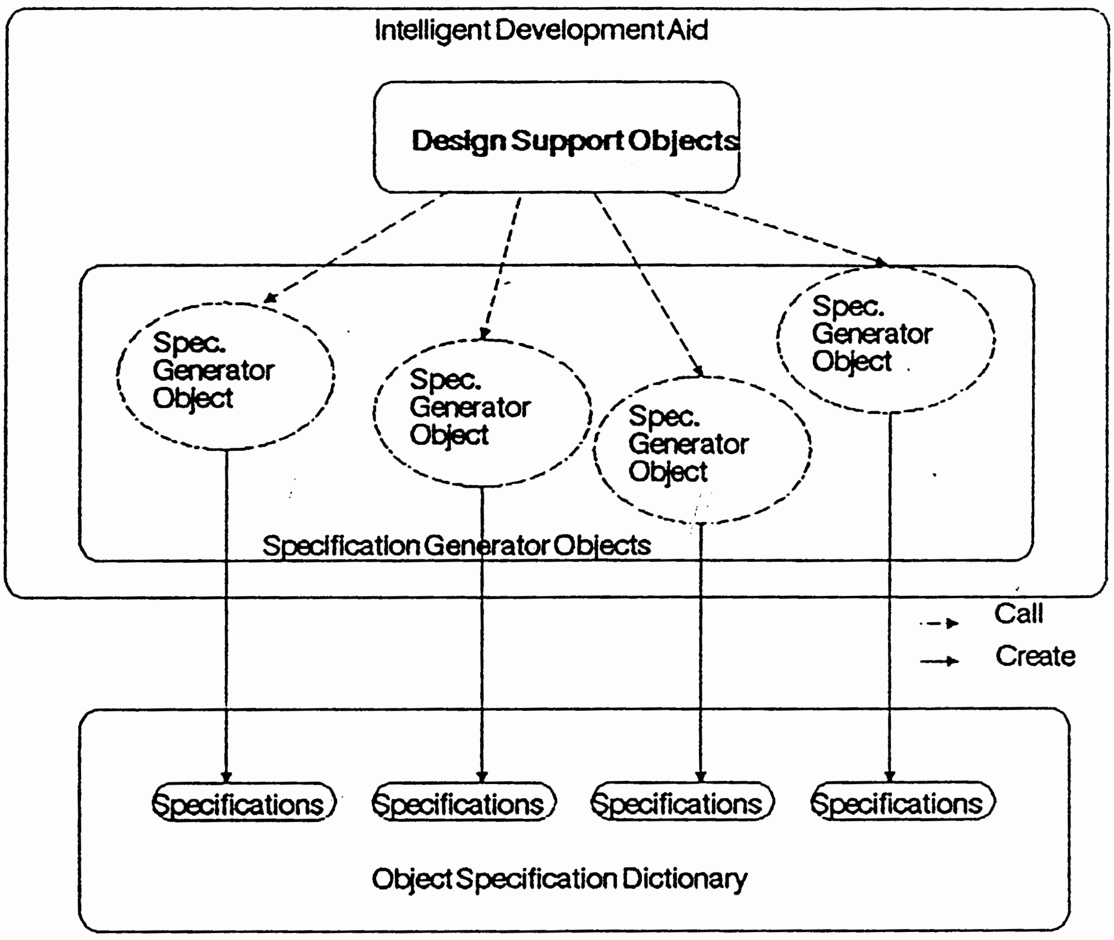


Figure 6. Structure of the Intelligent Development Aid

grouped under object classes in the Object Class Library. The advantage of this independence is that the maintenance responsibility of the user developer is separated from that of the MIS personnel. The development and maintenance of OSD are mainly the user's responsibility, while the responsibility of the development and maintenance of generic programs in the Object Class Library belongs to the MIS personnel. Thus, users can develop and maintain their application system by themselves if no major changes in generic programs are required.

In order to fully achieve this advantage, the OSD must have a standard and structured format. This is the reason that the output of the Intelligent Development Aid is a specification dictionary instead of a computer program. A dictionary has a standard structure. A computer program, in contrast, is based on processing procedures. The advantages of using a dictionary format is that certain elements can always be found in certain positions. Users can make changes to the application system by only changing the portion of the OSD that needs to be changed, which saves the work load of user developers in both system development and maintenance.

It must be noted that this study does not attempt to establish a generic format for the object specification in the OSD. There are two reasons. First, such a format would be highly dependent on the objects and object classes that are used to simulate the user's model of reality.

Unless some standardized object decomposition techniques are developed so that the concept of Software-IC (Cox, 1986) becomes reality, a generic format would not be available. Second, the development of a generic format for object specification is beyond the scope of the present research in that it is primarily concerned with the management of the object-oriented development. This research relates primarily to the feasibility of applying the object-oriented concept in a user developer environment.

The Object-Oriented Application System

The application systems are the systems that the user uses to accomplish his daily data processing works. An Object-Oriented Application System is similar to other application systems except its design and development is based on objects. An Object-Oriented Application System is made up of object classes. Figure 7 shows an example structure for an Object-Oriented Application System. The structure of the basic component, object class, is shown in Figure 8. Each object class has several objects. They are specified by the Object Specification Dictionary. Each object employs the methods under the object class to perform processing functions. Methods are generic programs that are written in a particular computer language. These programs have high flexibility and are generic to all the objects defined under the object class. Object classes and

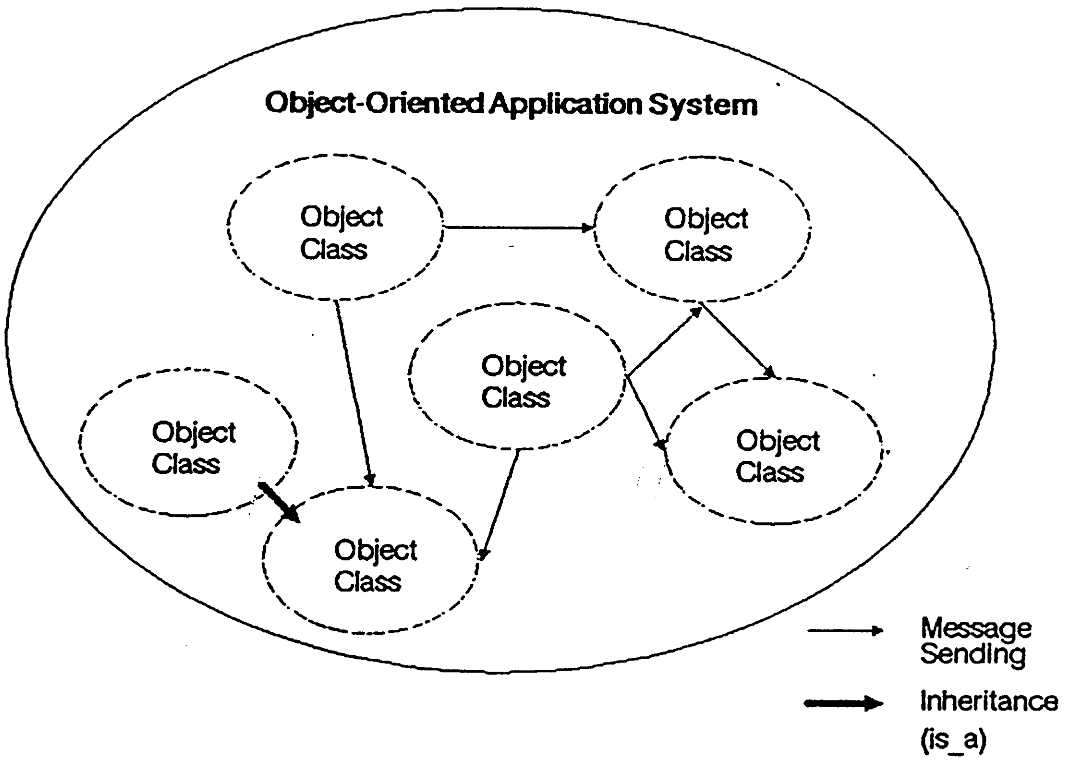


Figure 7. Structure of the Object-Oriented Application System

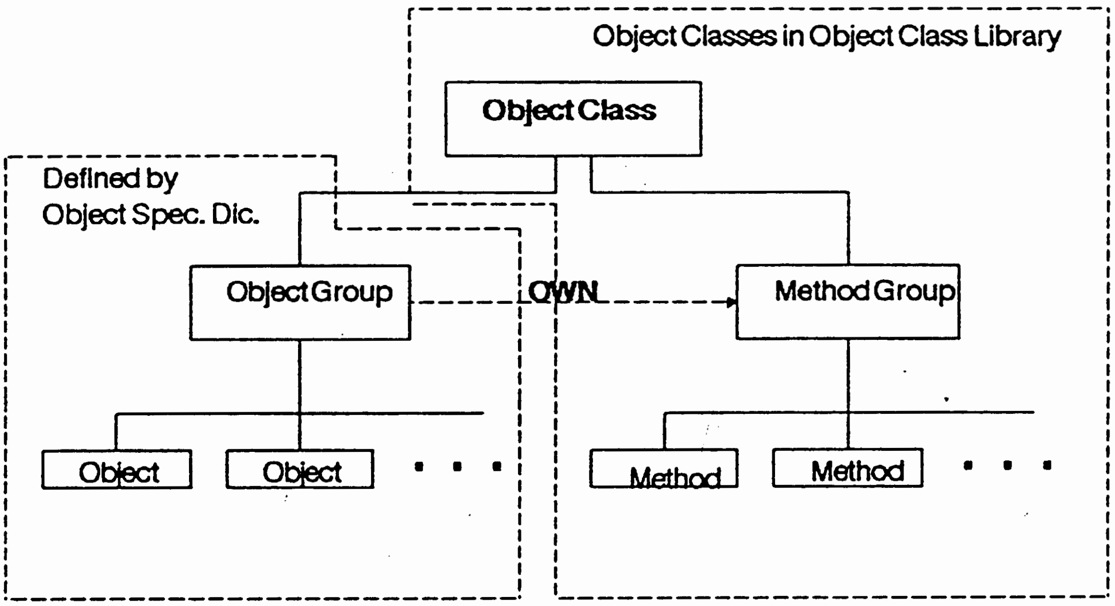


Figure 8. Structure of the Object Class

their methods are predefined and stored in the Object Class Library. There are two kinds of relationships between object classes in the library. First, object classes may interact with each other by message sending. Second, one object class inherits characteristics from the upper level object classes. Inheritance is represented by an is_a relationship between object classes. For example, a graduate student is a student. So the object class "graduate_student" can inherit characteristics from the object class "student". Any process and function (e.g. Add, Drop) that an object under the object class "student" can perform can also be performed by an object under the object class "graduate_student". The object class "graduate_student" can have its own methods which may be used to perform the processings and functions that are unique to a graduate student.

There are two components in the Object-Oriented Application System: (1) The Object Class Library, and (2) the Object Specification Dictionary. The following sections describe their functions within the application processing.

The Object Class Library

In order to perform his task in the work environment, every user first must develop mental models for his work environment. The user uses these models of reality to understand his work environment (Booch, 1985). These models of reality may be represented by object types that the user encountered in the work environment. The Object Class Library simulates the end-user's mental model(s) in a specific domain by object classes. For example, the following is an expression about a student work environment - school:

"student takes courses."

This description can be simulated in the Object Class Library by two object classes ("student" and "course"). The object class "student" should have a method call TAKE which sends out a message to the object class "course" to set up the relation between a student object and several course objects. The object classes "student" and "course" simulate the object types, and the method TAKE simulates the relationship of students taking courses in the model of reality.

Object Specification Dictionary

The Object Specification Dictionary provides required information about objects in the object classes. To specify an object, two kinds of information must be present in the OSD. The identification information identifies the object and the description information is needed for the object to perform methods. For example, the specification for an object in the object class "student" may have a format like this :

student(NAME, MAJOR, ADDRESS)

where NAME is the identification information, and MAJOR and ADDRESS are the description information. Suppose that student Tom Brown's major is Business. When one of the other objects sends out a message asking about Tom Brown's major, the object, student('Tom Brown'), associated with the object class "student" will use the information MAJOR to respond to the message.

It must be emphasized that the specifications in the Object Specification Dictionary must follow certain formats. When changes occur in the user's model, the formats may be changed to accommodate the changes in the object class library. Also, when new classes are added into the library, new formats need to be defined for the new classes.

Personnel Roles in the End-User Application Development Environment

The use of an Intelligent Database Application Development Aid involves the corporative efforts of end-users and MIS personnel. Both groups share the responsibilities for creation, operation and maintenace of the End-User Application Development Environment as illustrated in Figure 9. This section discusses the responsibilities of these two groups.

End-users

End-users are the developers and users of the application system. Their responsibility is to develop and to maintain the Object Specification Dictionaries. The role of end-users in the End-User Application Development

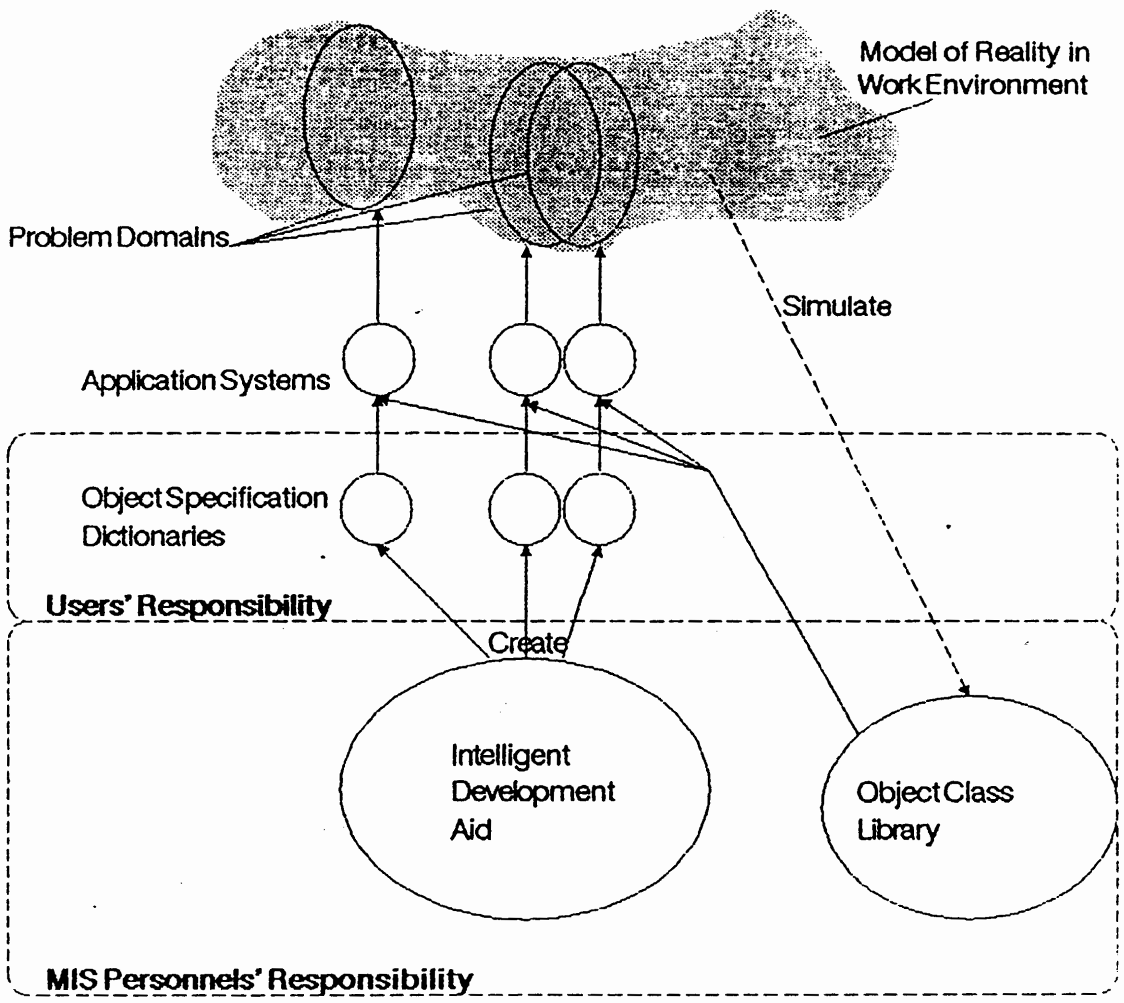

Model of Reality in Work Environment
Problem Domains
Simulate
Application Systems
Object Specification Dictionaries
Users' Responsibility
Create
Intelligent Development Aid
Object Class Library
MIS Personnels' Responsibility

Figure 9. Responsibility Sharing in End-User Application Development Environment

Environment involves three tasks:

1. They use existing application systems to perform their daily data processing activities at their work.
2. They develop the application systems with the assistance of the Intelligent Database Application Development Aid.
3. They maintain the application systems. End-users are responsible for the changes that can be made by changing the Object Specification Dictionaries.

MIS Personnel

MIS personnel are MIS professionals who construct the software systems such as IDADA. They are: (1) the technical supporters or developers of the Object Class Library; (2) the builders of the Intelligent Development Aid, who design and construct the development aid; and (3) the intermediaries to assist the users when necessary. The structure of the IDADA (Figure 3) suggests that the MIS personnel's responsibilities are as follows.

1. They develop the system such as the IDADA. Developing an intelligent development environment for end-users involves the following tasks:
   1) Identify the structure of the user's model(s) of their work environment.
   2) Implement the Object Class Library.
   3) Define the Object Specification Dictionary formats
   4) Implement an intelligent development aid

2. Maintain the development environment. MIS personnel are responsible for changes that can not be made by changing the Object Specification Dictionaries. These major changes require more sophisticated analysis and programming knowledge.

The System Development Procedure of IDADA

The system development procedure of IDADA is a four-step interactive process (Figure 10) derived from the approachs proposed by Abbott and Booch (198 1986). These steps are described below:

Step 1: Identify work environment model

1) Identify the object types and their attributes. The object types in the model of reality are identified at this step.

2) Identify the processing and functions required for each object type. This step characterizes the behavior of each object class. It establishes the static semantics of the object class by determining the operations that will be performed on the object or by the object.

3) Establish the abstraction of each object type in relation to other object types. Here, developers establish the interfaces and the visibility of each object. The purpose of this step is to capture the topology of objects from the model of reality.

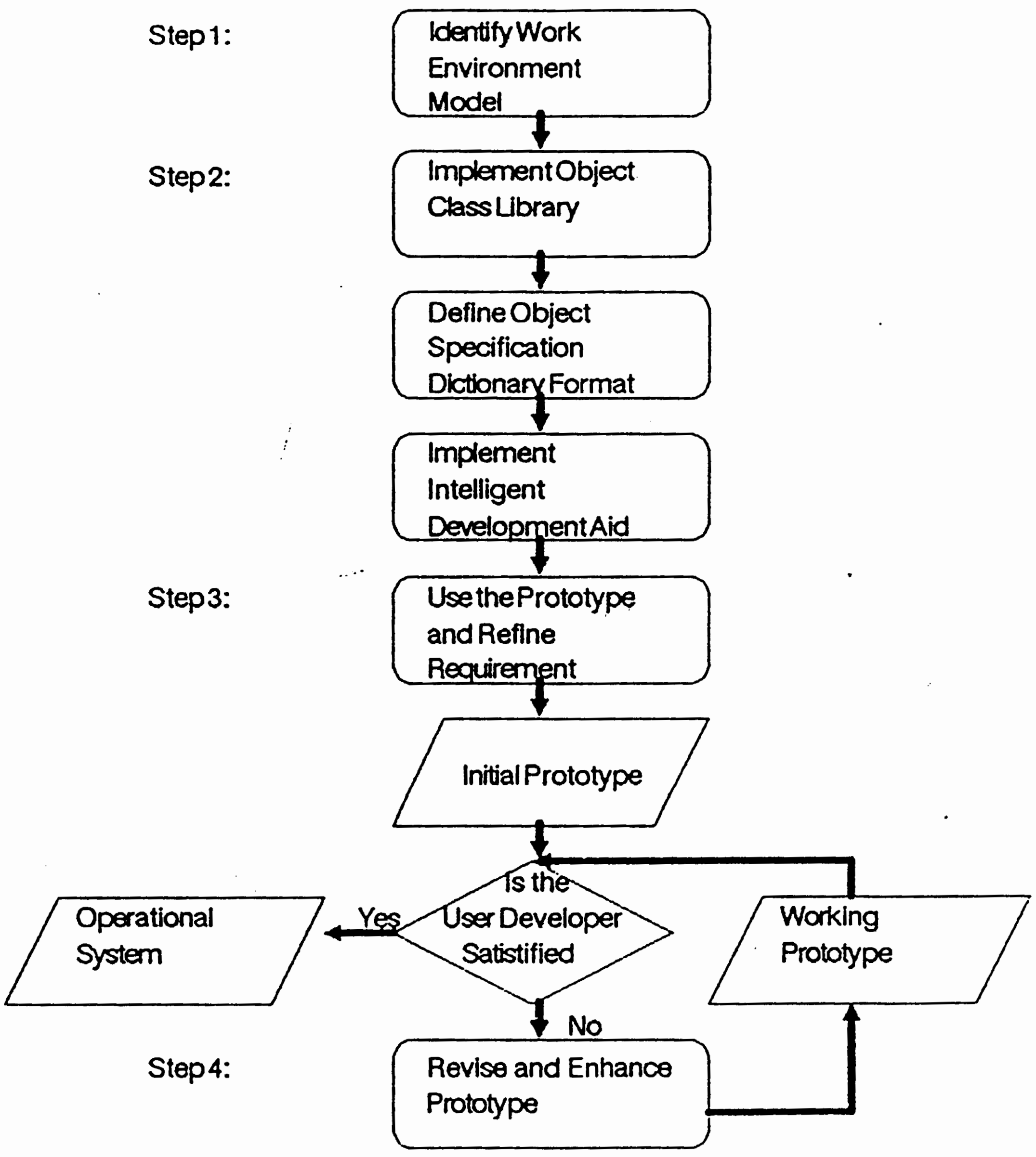

Step 1:
Identify Work Environment Model
Step 2:
Implement Object Class Library
Define Object Specification Dictionary Format
Implement Intelligent Development Aid
Step 3:
Use the Prototype and Refine Requirement
Initial Prototype
Is the User Developer Satistified
Yes
Operational System
Working Prototype
No
Step 4:
Revise and Enhance Prototype

Figure 10. The Development Procedure of Intelligent Database Application Development Aid

Step 2: Develop the initial prototype

1) Implement Object Class Library. A system that simulates the model of reality is implemented using suitable notations. In this research Prolog was used as the implementation vehicle.

2) Define Object Specification Dictionary formats. This determines the identification information and description information for the specifications in each object class and defines the OSD formats.

3) Implement Intelligent Development Aid. In this step the Specification Generator Objects that generate the OSD and the Design Support Objects that assist the design process are developed.

Step 3: Use the prototype and refine requirement. This allows the user(s) to experiment with the development aid to learn the system's functions as well as its limitations. If the prototype is accepted by users, it becomes the operational system and the development process is completed.

Step 4: Revise and Enhance Prototype. The developers modify the system based on users' request and return to Step 3.

Summary

This chapter has presented an architecture for the Intelligent Database Application Development Aid. The architecture suggests that the ability to support end-user developers results from the integration of a knowledge-based system (Intelligent Development Aid) and an object-oriented system (Object Class Library). The architecture suggests further that the knowledge-based component of the IDADA should serve to overcome the end-user developers' lack of professional development knowledge. Also, The Object Class Library of the IDADA should support the user's model of the work environment in order to minimize the knowledge required to develop application systems. In the next chapter, a prototype system is described which is used to evaluate this architecture and to demostrate the development procedure in the IDADA.

IV. PROTOTYPE DEVELOPMENT

The conceptual model described in Chapter Three defines the architecture for the Intelligent Database Application Development Aid. The purpose of this chapter is to describe a system prototype which demonstrates an implementation based upon that conceptual model. This prototype, the Intelligent Database Application Development Aid (IDADA), implements an intelligent environment for the development of database systems. Prolog was used as the implementation language, primarily because prolog possesses many convenient features for implementing the object-oriented concept. This chapter describes the user's model supported by this prototype, explains the prolog notations for object-oriented programming, and examines the structure of the prototype. Finally, an illustrative example is used to demonstrate operation of the system.

User's Model for Database System

The Intelligent Database Application Development Aid was developed to support the user's models of his work environment. The application system the user developed with the IDADA prototype is a database system with a menu interface. Its main purpose is to help its user to manage business forms or files and to generate reports based on these forms or files. For the research, several models were identified by different users through informal interviews. One of these models was chosen to be supported by this prototype.

Figure 11 summarizes the chosen model. Essentially, a database is an integrated collection of files. The user selects the database processing activity to perform through the menu interface. There are two kinds of database processing activities. First, when new information comes in, the user goes through certain processes to update the database. Second, when a report is needed, the user generates the report by selecting one of the report options from the menu interface. There are four object types (object classes) incorporated in this model. They are menus, display screens, reports, and database. A menu object can call another menu object, a report object, or a display screen object. When an display screen object is called, it display display screens to perform a process to update the database. When a report object is called, it retrieves data from the database and then generates the report.

Prolog Notations for Object-Oriented Programming

In the prototype, objects are represented by a flexible Prolog data type known as structure. Functionally, structure is a general purpose data type for definition of complex entities. The simplest form of a structure consists of

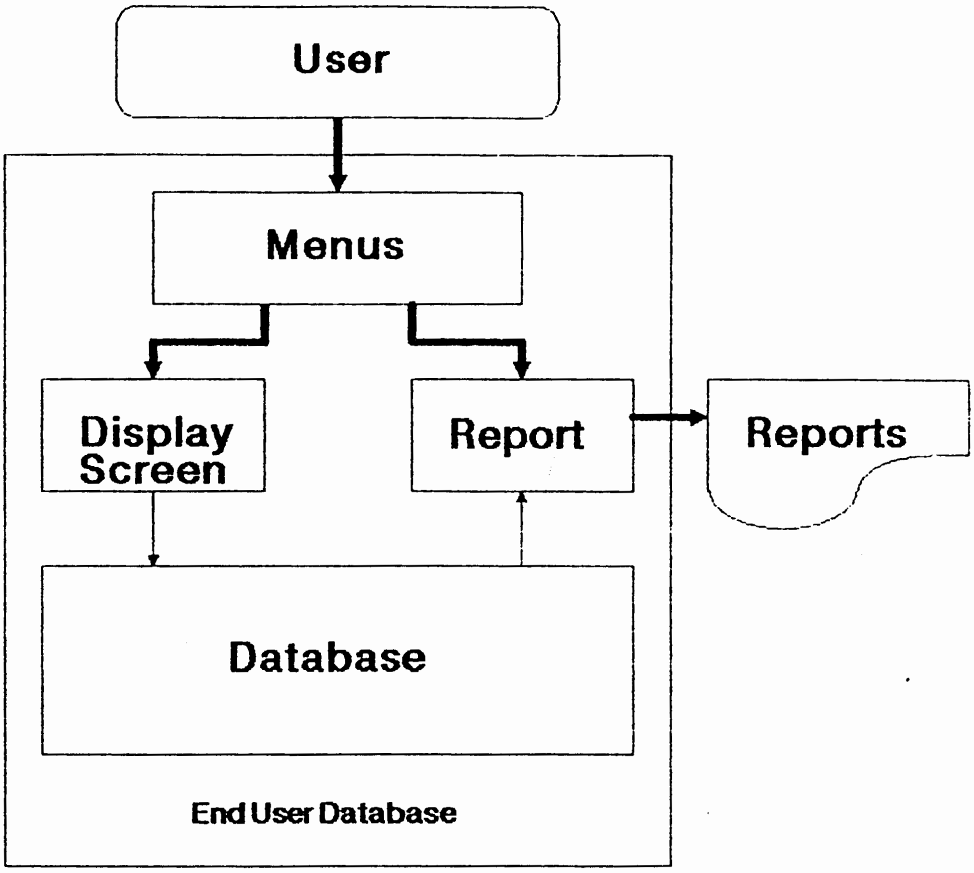


Figure 11. User's Model of the Database System

a functor and its associated arguments. For example, a structure,

report(class_rpt)

is an implementation of an object whose job is to generate report "class_rpt". The functor, "report" in this example, names the object class, and the argument, "class_rpt" identifies the object in the object class "report". An object class defines a group of object instances with similar properties. For example,

report(class_rpt)

is an object instance of

report(Rpt_Name).

Since object instances inherit the characteristics possessed by the object class they are associated with, it is only necessary to specify unique features in their definitions.

Structure is also used to implement messages. The functor names the message, whereas the arguments specify the parameters required for the corresponding operations to be performed. An example of message implementation is

add([s_id, s_name, s_addr, s_major]).

This message essentially asks the object that receives this message to add a record to the file.

Methods are represented by prolog predicates (either facts or rules). Methods are invoked as a response to a message reception. The outcome of a method processing could be operations or sending messages or a combination of both.

Message sending involves specifying the predicate "send" with two arguments. The first argument indicates the target object of the message, and the second specified the message.

For example, the predicate

```
send(database(student),
     add([s_id, s_name, s_addr, s_major]))
```

sends a message, add, to the object instance database(student). The operation requested here is, as described previously, adding the record [s_id, s_name, s_addr, s_major] into the database file - student.

Many benefits of object-oriented programming are derived in the fact that the entity, the methods, and the message-sending mechanisms are "encapsulated" into a package. Operations are invoked in reference to the entity. The following is an example of such an object specification:

```
database(Table) with [

        (add(Record)  :-  add(Table, Record)),
        (del(Record)  :-  del(Table, Record)),
        (retrieve(Record, Out_Rec)  :-
                  retrieve(Table, Record, Out_Rec)),
         description('Object class - Database')].
```

A message referring to an object instance, e.g., database (student), will cause one of the associated methods to be executed.

Prototype Description

System Structure

Figure 12 shows the structure of the system prototype as derived from the architecture described in Chapter Three. The database system developed by the prototype is a prolog internal database which is constructed by reading the object specification dictionary files and the object class library files. An Intelligent Development Aid was developed to assist users in developing or revising a database system by creating or revising five object specification dictionaries.

In this prototype, the development process is controlled by the object "dbde" which was implemented by the program DBDE.ARI (Appendix A). When provoked, it displays a menu (Figure 13) for the object options in both the operation and development phases. The user can select options under Operation Manager to get into the operation phase or select options under Development Manager to work with the Intelligent Development Aid. Available options are described below.

Operation Manager

Option 1: Test Run the Database System. The user can test his development by selecting this option. The Operation Manager constructs the prolog internal database with test data.

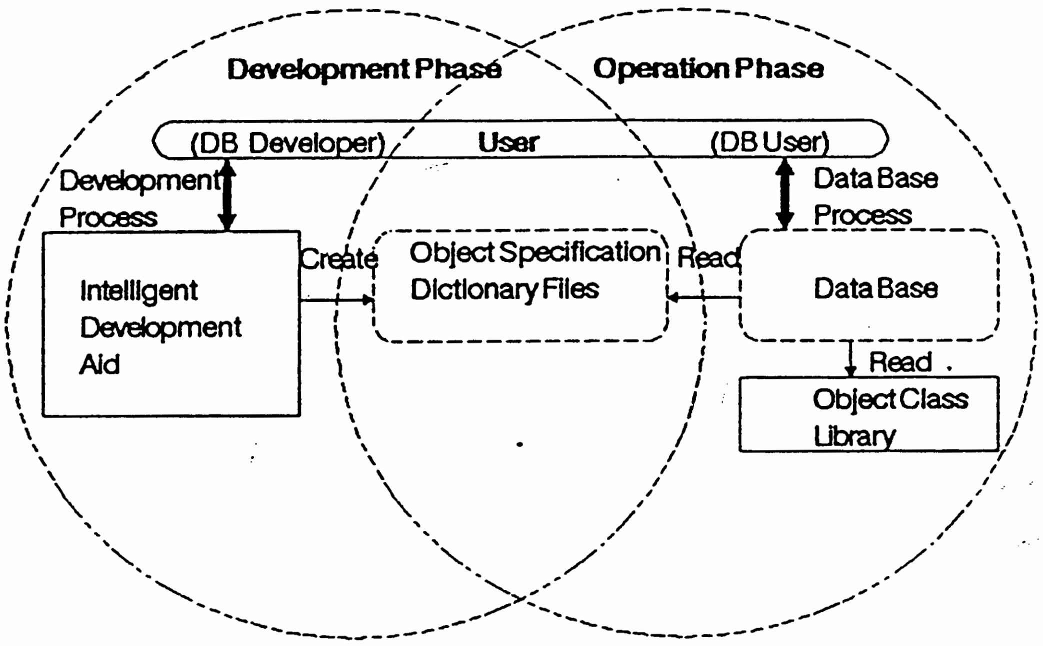
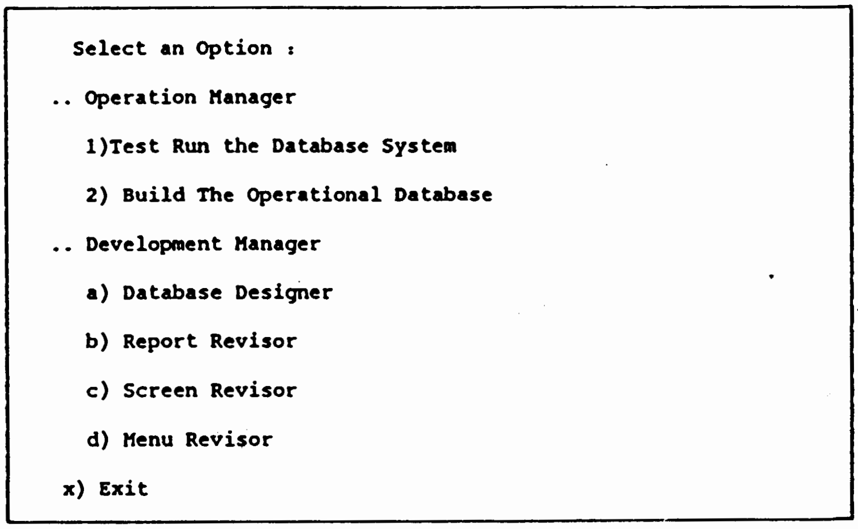


Figure 12. Structure of the IDADA Prototype



```
  Select an Option :
.. Operation Manager
   1)Test Run the Database System
   2) Build The Operational Database
.. Development Manager
   a) Database Designer
   b) Report Revisor
   c) Screen Revisor
   d) Menu Revisor
 x) Exit
```

Figure 13. Object "dbde" Screen

Option 2: Build the Operational Database. This option completes the development for the database system. The Operation Manager constructs the internal database without the test data and saves it under the name specified by the user.

Development Manager

Option a. Database Designer. The Development Manager calls the DB Designer to develop a new database system. This option will rewrite all the specification files in the Object Specification Dictionary.

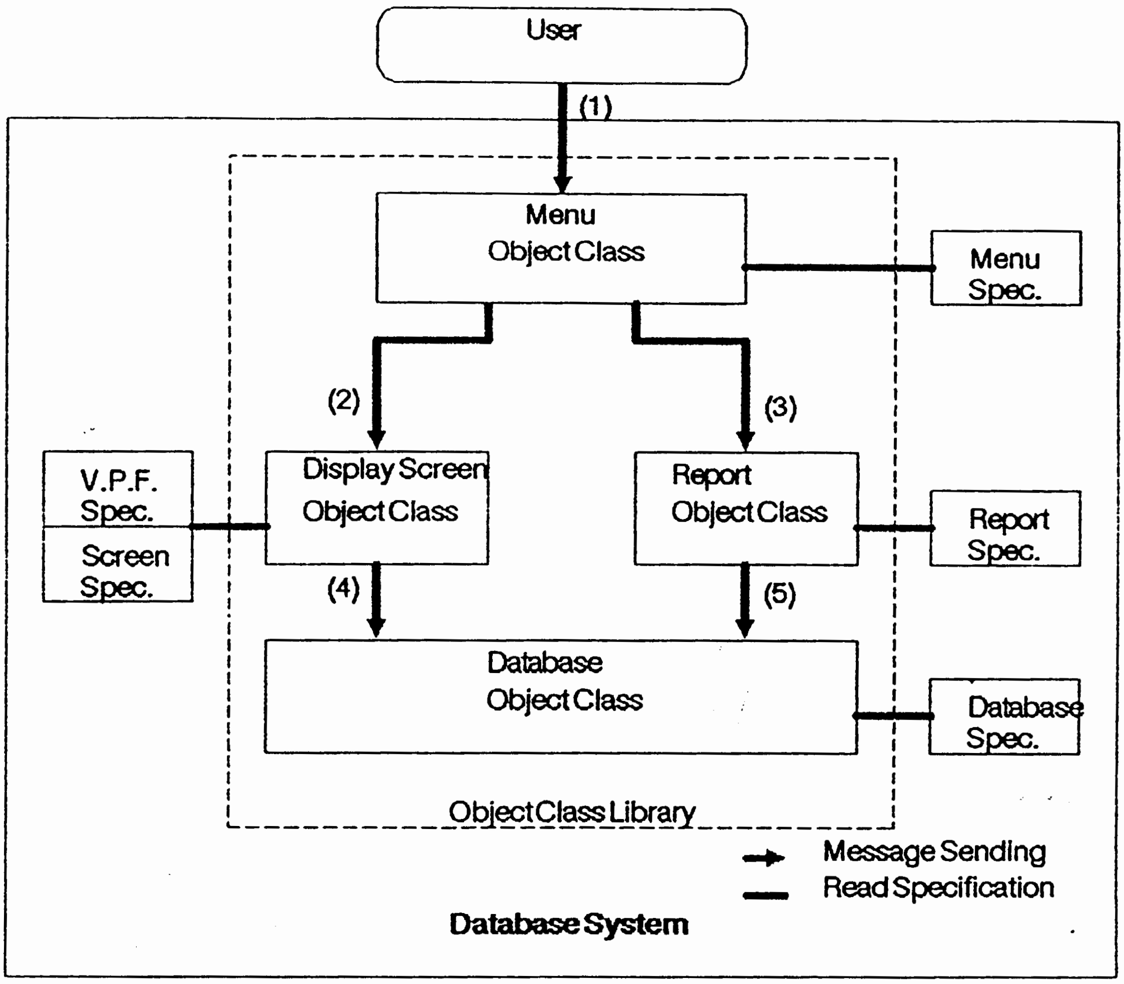
Option b. Report Revisor. The Development Manager calls the Report Revisor to redefine the report specification in OSD.

Option c. Screen Revisor. The Development Manager calls the Screen Revisor to redefine the specification file for the display screens.

Option d. Menu Revisor. The Development Manager calls the Menu Revisor to redefine the specification file for the menu interface.

Implementation of the Object Class Library

In the Object Class Library, four object classes with their methods were developed to simulate the user's model discussed in earlier. The resulting source listing is provided at Appendix B. Figure 14 shows the structure of the object class library and its relationship with the object specification dictionary files.



No.	Messages
(1)	DB
(2)	ADD, DEL, MOD
(3)	QUERY, REPORT
(4)	ADD, DEL, RETRIEVE
(5)	RETRIEVE

Figure 14. System Structure of the Database System

Table 1 lists these object classes and the messages they respond to. A typical message sent in this prototype includes two prolog structures - the object and the message. The object contains an object class name and an object ID that identifies the object in its object class. The message contains a message name and may or may not have several parameters. These parameters carry the data that is required by the receiver object to perform the associate method. For example, in the following message sending, send(database(student), add(['xyz', data])), database(student) is the object and add(['xyz', data]) is the message. Student is the object ID for an object in the object class "database". Add is the message name and ['xyz', data] is the parameter carried by this message.

A discussion of the object classes in the object class library follows. Their processes are summarized in Figure 15.

The object class "menu" performs the menu selection process. Two messages may be sent to this object class.

(1) menu: This message invokes method menu/1[1] to display specific menu objects and accept the user's selection. The name of the menu object comes with the message sending. Its format is defined by the menu specification in the object specification dictionary.

[1]menu/1 means a prolog predicates (function) that contains one argument (parameter). In this case this argument is the object ID.

Table 1

Object Class and Method List

Object Class	Method
menu	menu return
update	add del mod
b_table	add del retrieve
report	report query

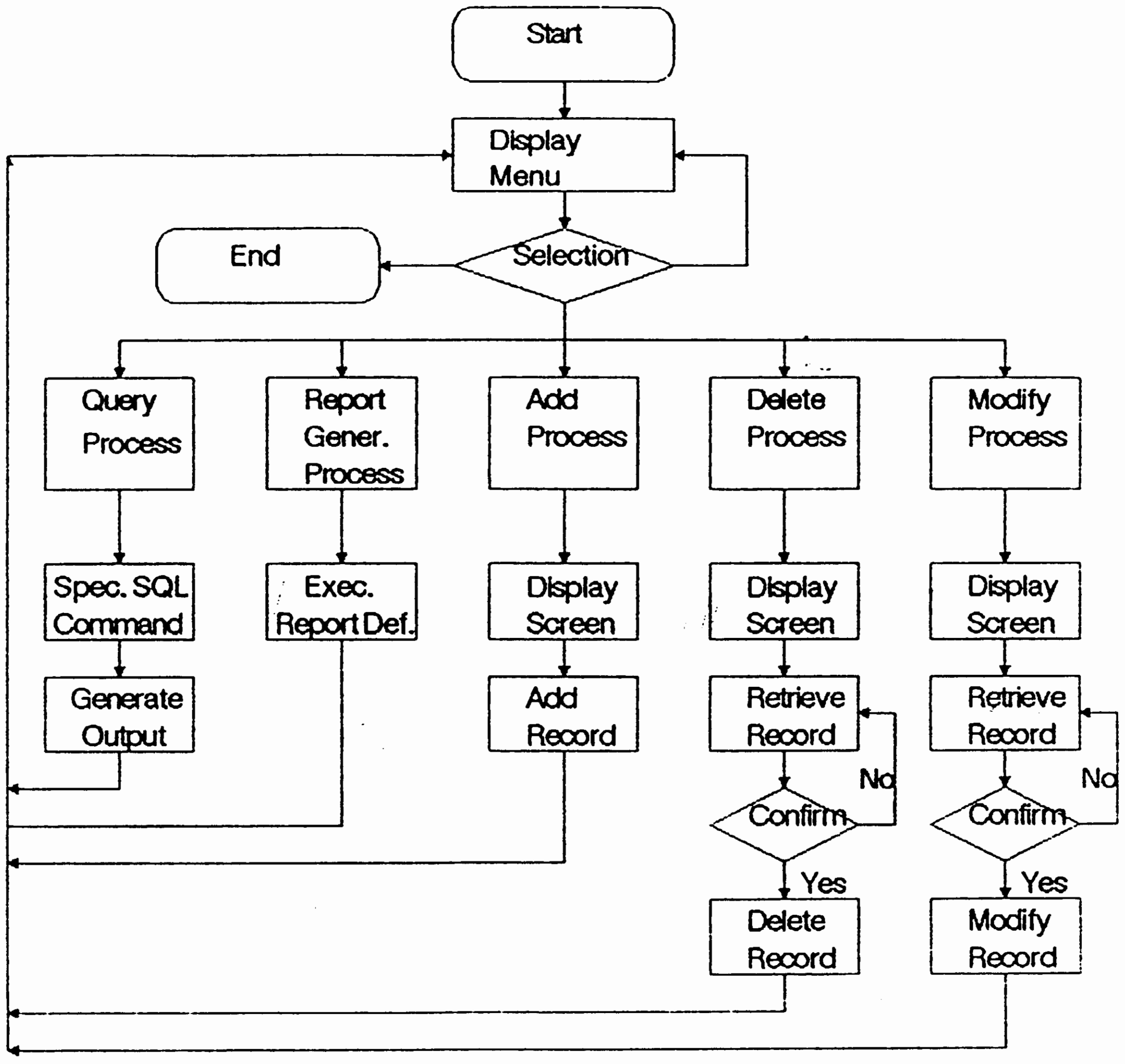


Figure 15. Processes of the Database System

(2) return: This message invokes method return/1. When responding to this message the object class closes the menu object currently in processing and returns control to the sender of the menu message.

The object class "display screen" performs the update process to the database. One of the following three messages may be sent to this object class:

(1) add: This message triggers method add/1 to perform the add record process. When processed, method add/1 displays the input screen of the update process object and asks the user to fill in the data field values. It then sends a message to object class "database" to insert this record into the correspondent database file.

(2) del: This message invokes method del/1 to delete a record from the database. When processed, method del/1 first displays the input screen of the update process object and asks the user to specify the record. The method then sends a message to object class "database" to retrieve the first record which meets the specification. After displaying the record to the user and user confirmation is obtained, the record is removed from the database; otherwise the next record which meets the specification is display.

(3) mod: This message calls method mod/1 to modify a record from the database. When executed, method mod/1 first displays the input screen of the update process object and asks the user to specify the record. The method

then sends a message to object class - database to retrieve the first record which meets the specification and displays it to the user. After examining the record the user can either make changes to this record or skip it and view the next record.

The object class "report" generates reports and performs a query process for the user. This object class is built base on the Arity/SQL Development Package [2] and responds to the following two messages:

(1) report: When responding to this message, the object class calls its report generator program (method). The specification for this report in the object specification dictionary is consulted and generates the report by retrieving data from the object class "database".

(2) query: This message triggers a query process which performs a database query. A menu-driven SQL facility is used as the interface to specify the query selecton.

The construction of object class "database" is based on Arity/SQL Development Package [3]. It performs three basic file processes: add, delete, and retrieve.

(1) add: This message comes with one parameter. The parameter contains the record that needs to be added to the database. When the method add/2 is called, the database

[2] Arity/SQL Development Package is a trademeark of Arity Corp.

[3] Arity/SQL Development Package is a trademeark of Arity Corp.

file specified by the message sent is found and the record is inserted into it.

(2) del: This message also comes with the record of interest as a parameter. Method del/2 is executed to remove this record from the correspondent database file.

(3) retrieve: This message triggers method retrieve/3. When processed, the method finds the first record that meets the specification in the message and returns it to the message sender. If the sender fails to return the message, the method finds the next record and returns it. This process continues until either no record is left or the message sender accepts the return record.

Implementation of the Intelligent Development Aid

The main purpose for the Intelligent Development Aid (IDA) is to provide assistance to user developers in generating their database systems. The prolog programs that implement the IDADA are listed in Appendic C. The development aid was constructed with two categories of objects - the design support objects and the specification generator objects. The design support objects include the following objects.

1. Development manager: Controls the other objects in the IDA. It calls the DB Designer when the user wants to develop a new database. When the user is prototyping the database, it calls objects - Report Revisor, Screen Revisor, or Menu Revisor - to modify the original design.

2. DB Designer: Helps the user to design the database structure. It prompts the user for the input data file format. It then works interactivly with the user to normalize the input data files into database files in the third normal form. After finishing the database design, it calls DB Developer to generate the object specification dictionary.

3. DB Developer: Calls the specification generator objects sequencially to construct the object specification dictionary.

4. Report Revisor: Loads and calls the Report Revisor to redefine the report specification.

5. Screen Revisor: Loads and calls the Screen Revisor to redefine the specifications for input screens.

6. Menu Revisor: Loads and calls the Menu Revisor to redefine the specifications for the menu interface.

The output of the Intelligent Database Development Aid is the object specification dictionary files. These are generated by the specification generator objects. The following is the list of these generator objects:

1. Database Specification Generator,
2. Report Specification Generator,
3. View Processing Facility Specification Generator,
4. Screen Specification Generator,
5. Menu Specification Generator.

When executed each of these generators work interactively with the user to generate a text file that contains the specification. Figure 16 shows the structure of the Intelligent Development Aid.

To develop a database, the user goes through a prototyping process. The steps in developing a database under the IDADA are:

1. Input file formats and normalize
2. Specify database
3. Specify report process
4. Specify update process
5. Specify menu interface
6. Test run
7. Revise and enhance

These steps are shown in Figure 17.

Step one is to input file formats and normalize the files. This is performed by the object DB designer. In this step, the user defines the files or forms to be stored in the database. The output is the database design. The user inputs three types of information to define a file or form. First, the user specifies the name and length of every field in this file or form. Second, the user specifies all the key fields in this file or form. Third, the user specifies the functional dependencies in this file or form. Figures 18, 19, and 20 show the input screens which input this information. DB Designer uses the information to normalize this file into the third normal form (3NF) database files and develops the database design.

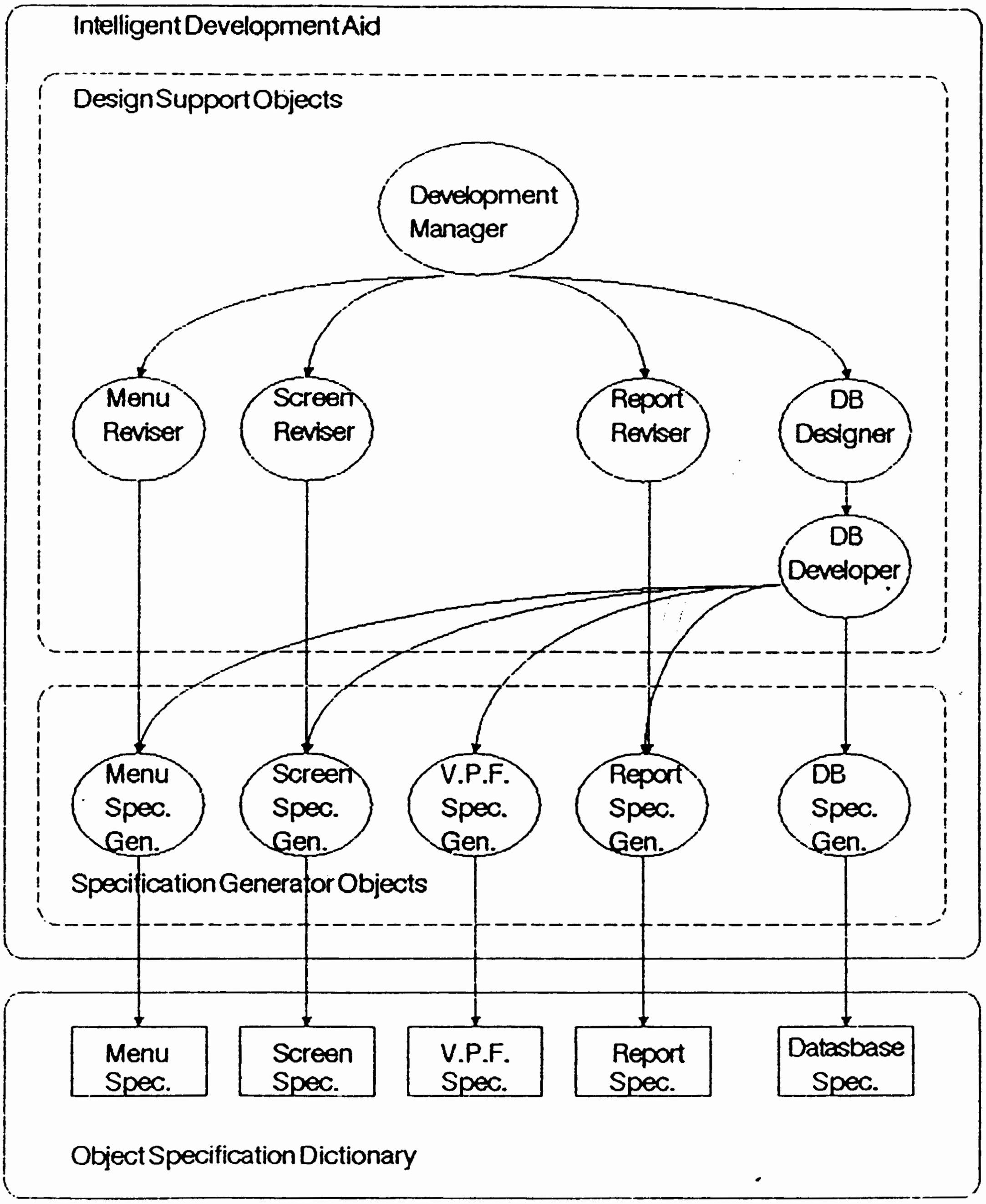


Figure 16. Structure of the Intelligent Development Aid

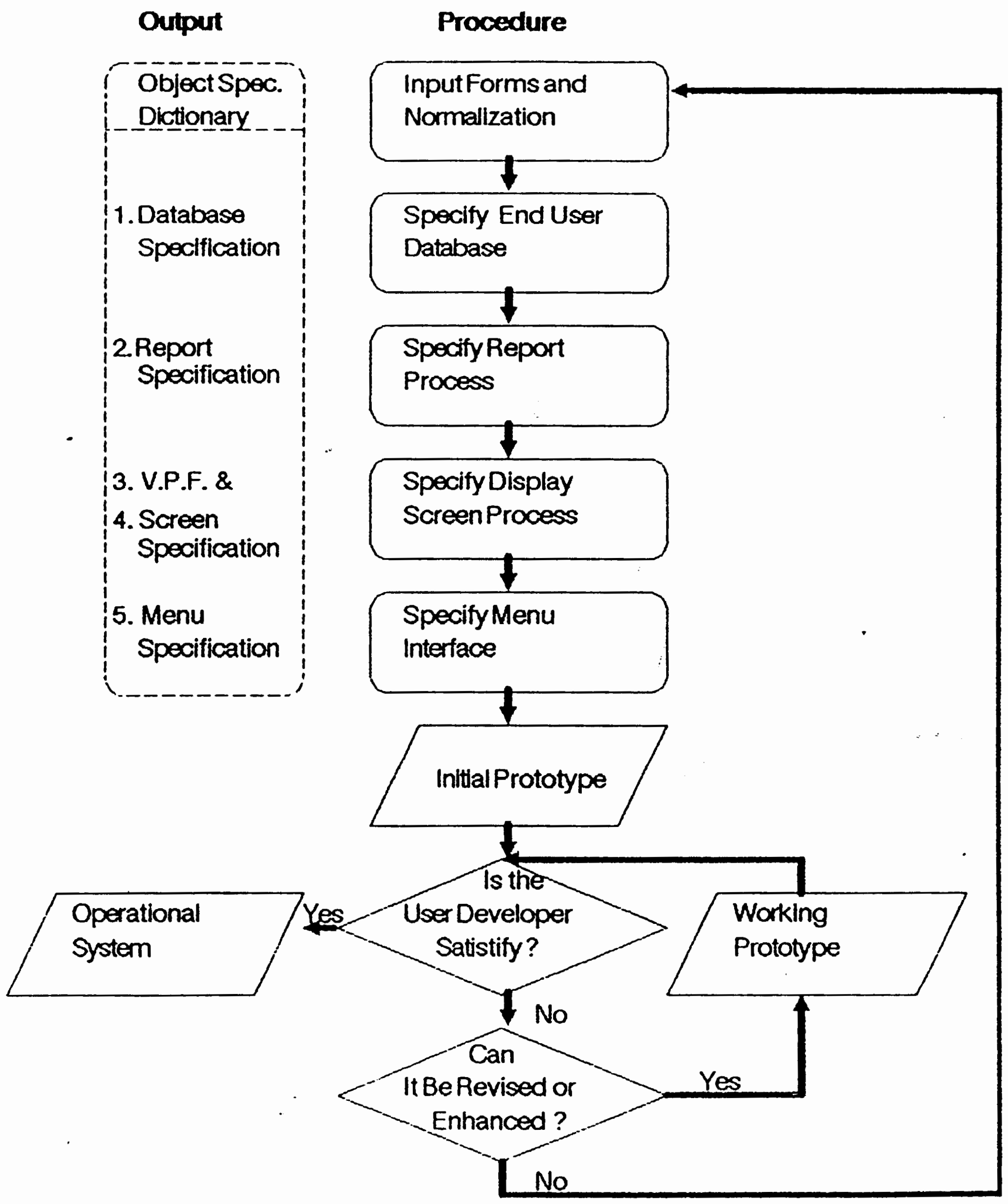
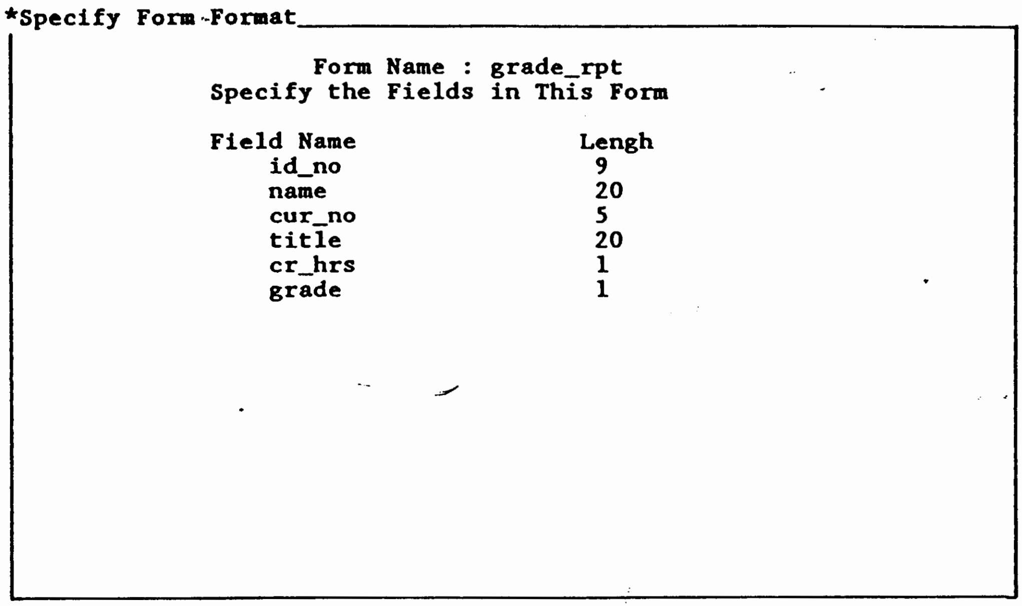


Figure 17. End-User Database Development Process

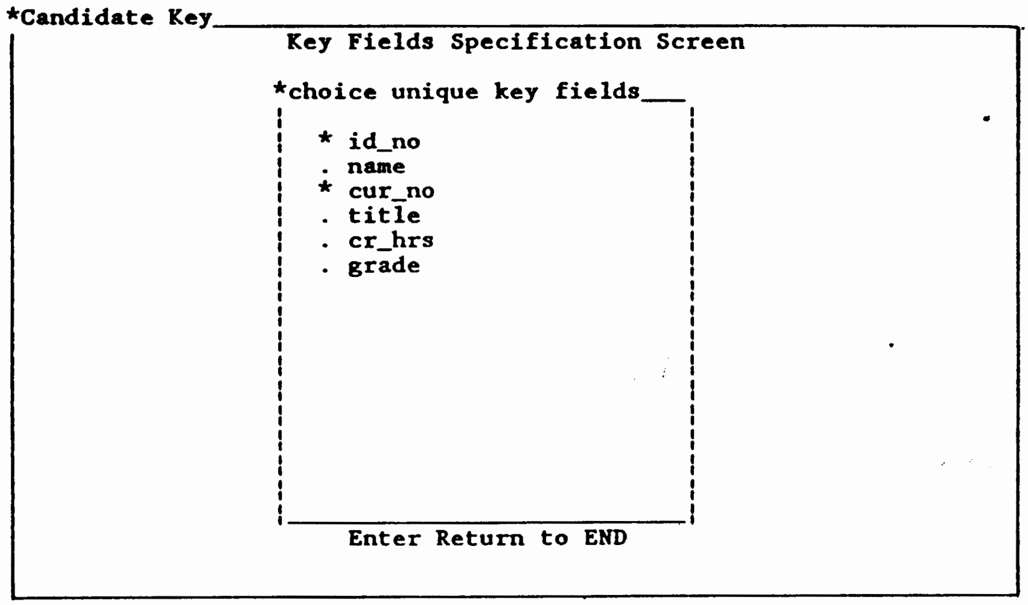


```
*Specify Form Format

                Form Name : grade_rpt
         Specify the Fields in This Form

         Field Name                 Lengh
             id_no                    9
             name                     20
             cur_no                   5
             title                    20
             cr_hrs                   1
             grade                    1
```

Figure 18. Specify Form Format Screen



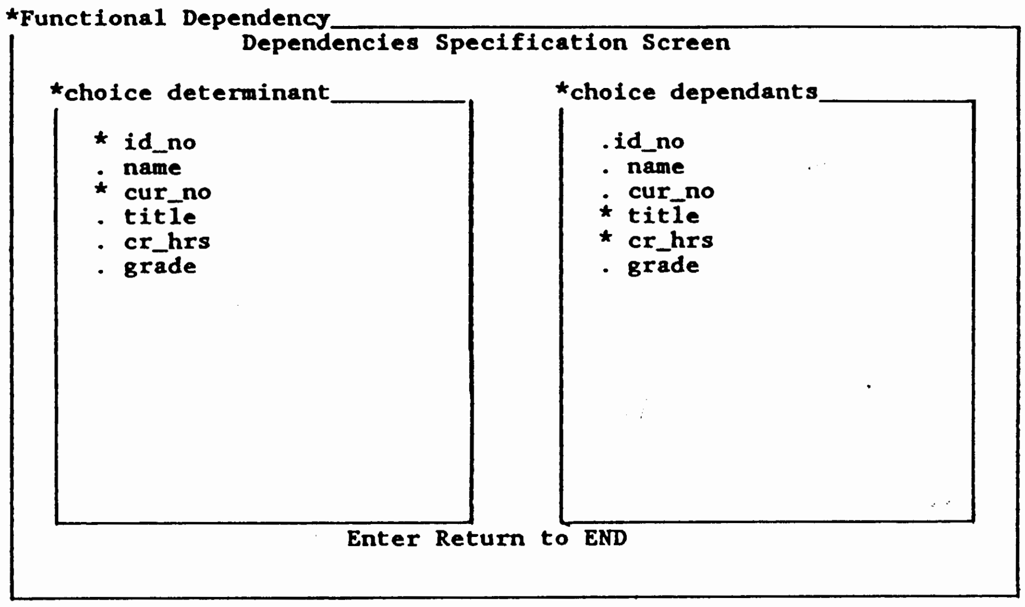
```
*Candidate Key
                    Key Fields Specification Screen

                   *choice unique key fields

                      * id_no
                      . name
                      * cur_no
                      . title
                      . cr_hrs
                      . grade

                        Enter Return to END
```

Figure 19. Specify Key Fields Screen



```
*Functional Dependency________________________________________________
|                   Dependencies Specification Screen                 |
|                                                                     |
|  *choice determinant__________      *choice dependants_____________ |
|  |                            |     |                              ||
|  |   * id_no                  |     |  .id_no                      ||
|  |   . name                   |     |  . name                      ||
|  |   * cur_no                 |     |  . cur_no                    ||
|  |   . title                  |     |  * title                     ||
|  |   . cr_hrs                 |     |  * cr_hrs                    ||
|  |   . grade                  |     |  . grade                     ||
|  |____________________________|     |______________________________||
|                          Enter Return to END                        |
|_____________________________________________________________________|
```

Figure 20. Specify Functional Dependency Screen

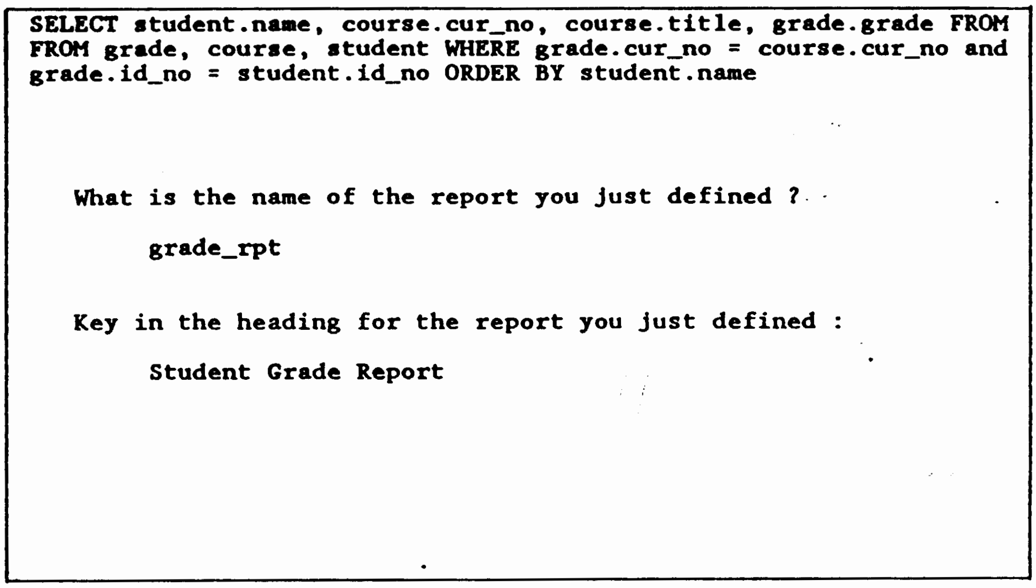
After step one, the object DB developer is called, which then calls specification generators sequentially to finish steps two through five. In step two, the database generator translates the database design into the database specification of the object specification dictionary. The system asks the user to name each 3NF database file and then to generate the database specification in a file called bt.osd.

The user defines the output reports in step three. The menu-driven inferface helps the user to specify the SQL SELECT commands used by these reports. The basic format for the SELECT command is:

```
SELECT (field names)
FROM (database file names)
WHERE (conditions)
GROUPED BY (field names)
ORDER BY (field names)
```

After specifying the SELECT command, the system asks the user to input the name, the main heading, and the headings and lengths for each selected field of this report (Figure 21).

Step four generates the View Processing Facility (VPF) Specification and the Input Screen Specification. The VPF Specification specifies the update processing for each database files defined in step two. It is directly derived from the Database Specification. The Screen Specification specifies the input screens that are used in those update processings. For each screen, the system asks the user to



```
SELECT student.name, course.cur_no, course.title, grade.grade FROM
FROM grade, course, student WHERE grade.cur_no = course.cur_no and
grade.id_no = student.id_no ORDER BY student.name



   What is the name of the report you just defined ?

        grade_rpt

   Key in the heading for the report you just defined :

        Student Grade Report
```

Figure 21. Specify Report Format Screen

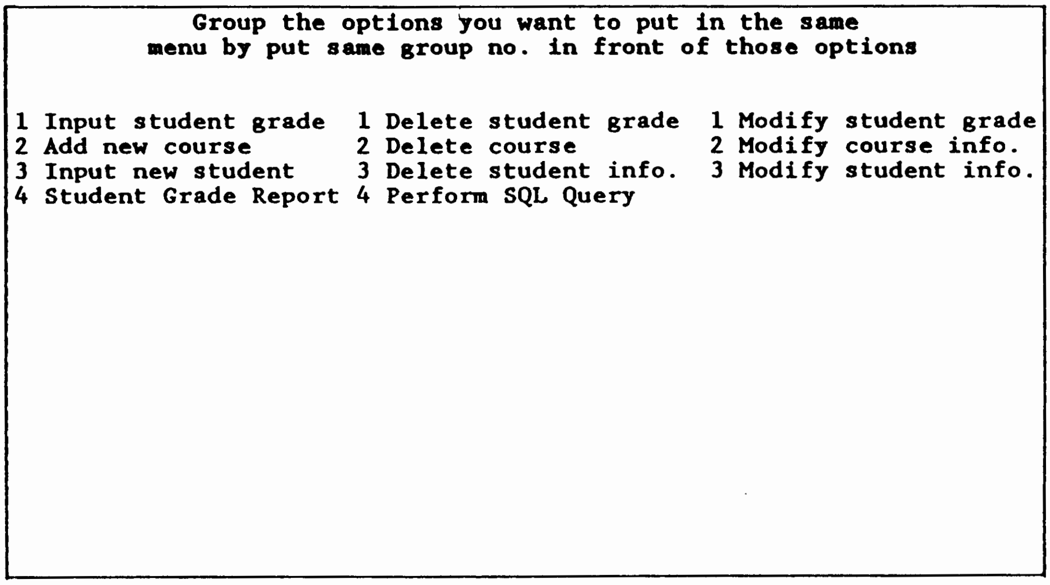
name the screen and input descriptions for each input field of the screen.

The output of step five is the specification for the menu interface. The system lists all the input and output processes available in this database (Figure 22). The user can group several types of processing into one menu by associating the same group number with each process or he can delete a process by associating a 'd' with it. The user can also change the option descriptions shown in the menu by changing the processing descriptions on the list.

With the completion of these previous steps, the initial prototype is ready for the test run. After each test run, the user can call the revisor objects to revise the Object Specification Dictionary until this development is acceptable. If the revision does not satisfy the user, the database system can be abandoned and redefined completely.

Implementation of the Object Specification Dictionary

The Object Specification Dictionary is constructed by five object specification dictionary files. They are BT.OSD for database specification, RPT.OSD for report specification, VPF.OSD for view processing facility specification, SCRN.OSD for input screen specification, and MENU.OSD for menu specification. Their formats are shown in Figure 23. Both the view processing facility (VPF) specification and screen specification are read by object class - display screen to construct its objects. VPF specification defines



```
          Group the options you want to put in the same
        menu by put same group no. in front of those options


1 Input student grade  1 Delete student grade  1 Modify student grade
2 Add new course       2 Delete course         2 Modify course info.
3 Input new student    3 Delete student info.  3 Modify student info.
4 Student Grade Report 4 Perform SQL Query
```

Figure 22. Processing List

```
Figure 23 - 1
---- Formats for System Dictionary ----

% menu definitioin

% menu definition format
begin_def(ACTIVE KEY).
% option fields
choice_box(begn, 0, 0, 24, 79, none, ' Menu ', radio, 3,_).
choice(rone, 10,20, 'OPTION 1', unchecked, _).
choice(rtwo, 12,20, 'OPTION 2', unchecked, _).
choice(rthree, 20,20, 'Exit', pushed, _).
% option fields end
choice_box_end.
end_def(ACTIVE KEY).

% static display
begin_def(STATIC KEY).
text(4,35, 'HEADING').
text(7,20, 'Select Option').
end_def(STATIC KEY).

% option process specification
begin_def(KEY).
menu_choice(1,'PROCESS FOR OPTION 1').
menu_choice(2,'PROCESS FOR OPTION 2').
menu_choice(3,'PROCESS FOR EXIT THIS MENU').
end_def(KEY).

% screen definition format

begin_def(ACTIVE KEY).
% input fields
efield(one, 5, 30, 9, $$, _).
efield(two, 7, 30, 15, $$, _).
efield(three, 9, 30, 20, $$, _).
efield(four, 11, 30, 10, $$, _).
% input fields end
end_def(ACTIVE KEY).

% static display
begin_def(STATIC KEY).
text(2,20, 'HEADING').
% notations for fields
text(5, 5, 'Id Number : ').
text(7, 5, 'Name : ').
text(9, 5, 'Address : ').
text(11, 5, 'Major : ').
% notation end
end_def(STATIC KEY).

% headings for different processing
begin_def(KEY).
operation('add','ADD NEW STUDENT PERSONAL INFORMATION').
operation('del','DELETE STUDENT PERSONAL INFORMATION').
```

Figure 23. OSD Format

```
Figure 23 - 2
operation('mod','MODIFY STUDENT PERSONAL INFORMATION').
end_def(KEY).

% View File Definition

begin_def(KEY).
view_file(
    [$insert command capsules; $,$insert command capsules; $,...],
    [$delete command capsules; $,$delete command capsules; $,...],
    ['p_file name', 'field no','p_file name', 'field no',.......]).
end_def(KEY).


% Report definition

begin_def(KEY).
report( $ select command capsules; $, $heading$).
end_def(KEY).


% Phisical File Definition

create table FILE NAME  (
        FIELD_NAME1    type(lengh)  not null,
        FIELD_NAME2    type(lengh),
        FIELD_NAME3    type(lengh),
                );
```

Figure 23. (Continued)

the update processes while the screen specification specifies the input screen format used in these processes.

A demonstration of the IDADA is presented in the following section.

An Example

Problem description

The following example application demonstrates the operation of the Intelligent Database Application Development Aid in developing a faculty service database. The description of the problem is as follows.

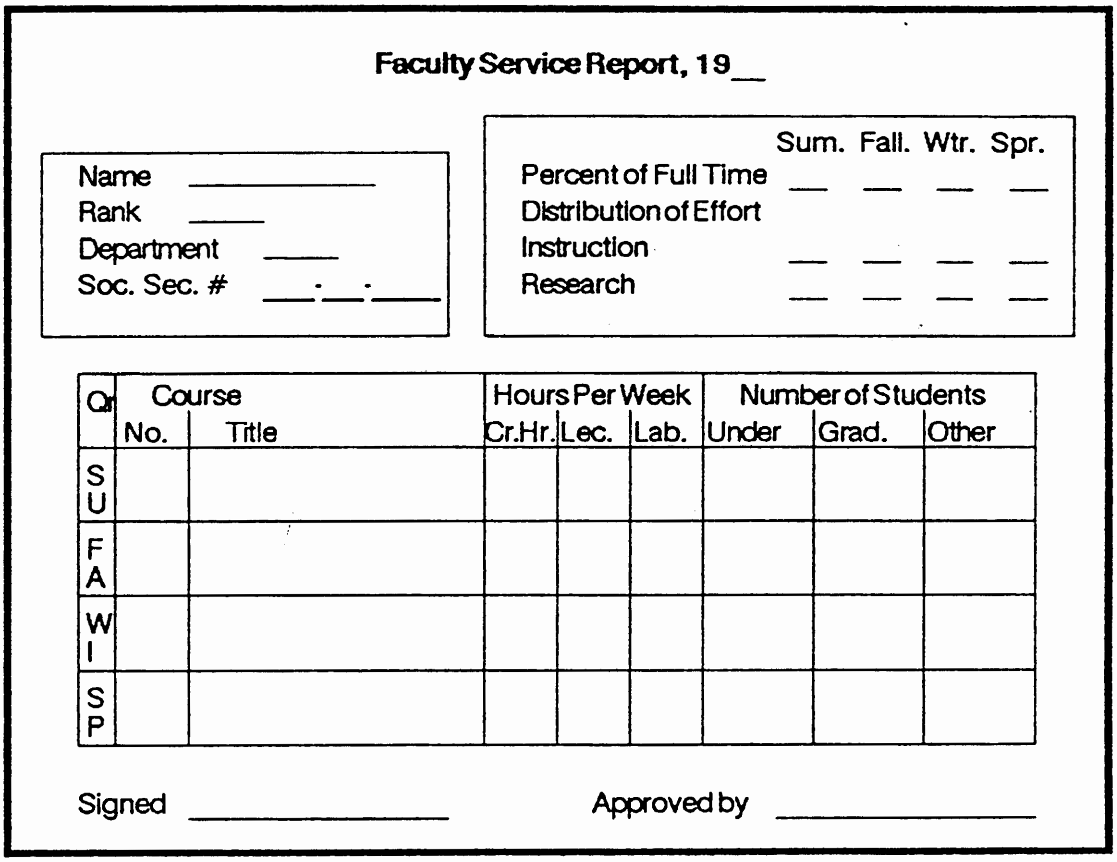
> A university manages its faculty services by forms called Faculty Service Reports. Each fiscal year faculty members are asked to report their service information by completing a new Faculty Service Report. These reports are then stored in the dean's office until the next fiscal year. Figure 24 shows the input - the Faculty Service Report. The three sections included are:
> 1. Faculty Personal Information
> 2. Service Information
> 3. Teaching Information
>
> The following reports are generated from these Faculty Service Reports:
> 1. Faculty Teaching Report.
> 2. Course Report.
> 3. Class Report.

Development process

The following is the user's input for development of this Faculty Service Database, described in five steps.

Step one:
(1) File fields:

Field Name	Lengh
name	20
rank	20
department	3
ss#	9
quarter	1



Faculty Service Report, 19__

Name	________
Rank	_____
Department	_____
Soc. Sec. #	___-___-____

	Sum.	Fall.	Wtr.	Spr.
Percent of Full Time	__	__	__	__
Distribution of Effort				
Instruction	__	__	__	__
Research	__	__	__	__

Qr	Course		Hours Per Week			Number of Students		
	No.	Title	Cr.Hr.	Lec.	Lab.	Under	Grad.	Other
SU								
FA								
WI								
SP								

Signed ____________ Approved by ____________

Figure 24. Faculty Service Report

serv_perc	3
instruction	3
research	3
cur_no	5
cur_name	20
cr_hrs	1
lec_hrs	1
lab_hrs	1
under_stu	3
grad_stu	3
other_stu	3

(2) Candidate keys
ss#
quarter
cur_no

(3) Functional dependencies

```
ss#                     -> name, rank, department
ss#, quarter            -> serv_perc, instruction,
                           research
cur_no                  -> cr_hrs, lec_hrs, lab_hrs
ss#, quarter, cur_no -> under_stu, grad_stu,
                           other_stu
```

Step two:

Database File Name	Fields
course	cur_no, cur_name, cr_hrs, lec_hrs, lab_hrs
service	ss#, quarter, serv_perc, instruction, research
faculty	ss#, name, rank, department
class	ss#, quarter,cur_no, under_stu, grad_stu, other_stu

Step three:

Report Name	Heading
teach_rpt	Faculty Teaching Report

```
SELECT quarter, name, cur_no, cr_hrs, cur_name
FROM faculty, class, course
WHERE faculty.ss# = class.ss# and
      class.cur_no = course.cur_no

ORDERED BY quarter, name
```

course_rpt   Courses Report

```
SELECT quarter, cur_no, cur_name, under_stu,
          grad_stu, other_stu
FROM class, course
WHERE class.cur_no = course.cur_no
GROUP BY quarter, cur_no
ORDERED BY quarter, cur_no
```

class_rpt   Classes Report

```
SELECT quarter, name, cur_no, under_stu,
          grad_stu, other_stu
FROM class, faculty
WHERE class.ss# = faculty.ss#
ORDERED BY quarter, name, cur_no
```

Step four:

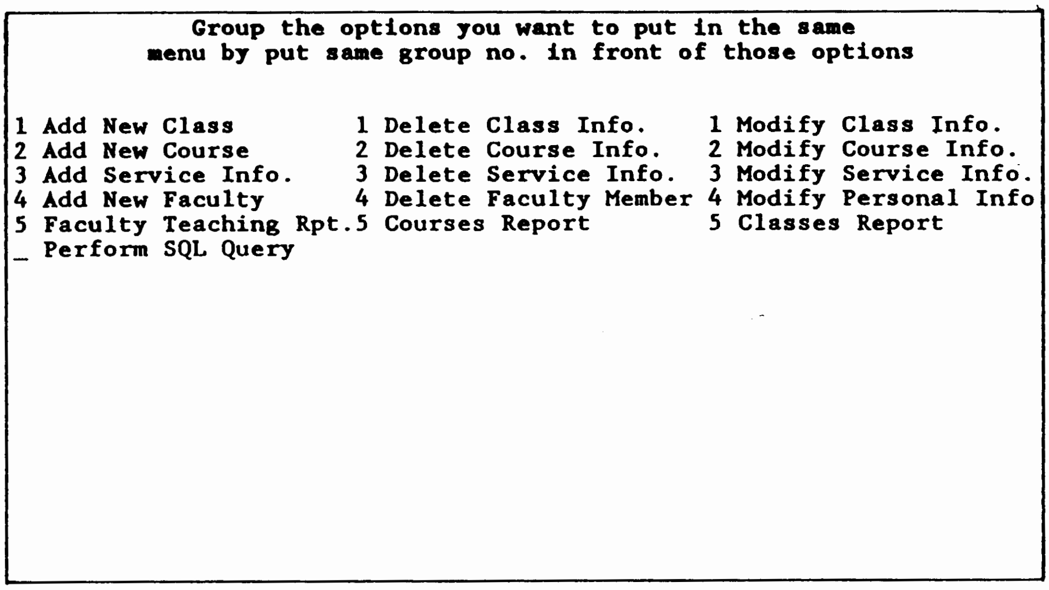
Name	Processing	Screen Title
faculty	add	Add New Faculty Member
	del	Delete Faculty Member
	mod	Modify Personal Infomation
service	add	Add Service Information
	del	Delete Service Information
	mod	Modify Service Information
course	add	Add New Course
	del	Delete Course Information
	mod	Modify Course Information
class	add	Add New Class
	del	Delete Class Information
	mod	Modify Class Information

Step five:

Processing List - Level 1 menus (Figure 25)

Group no	Menu name	Heading
1	up_class	Update Cl Info.
2	up_cur	Update Course Info.
3	up_serv	Update Service Info.
4	up_facu	Update Personal Info.
5	out_rpt	Reports Generation

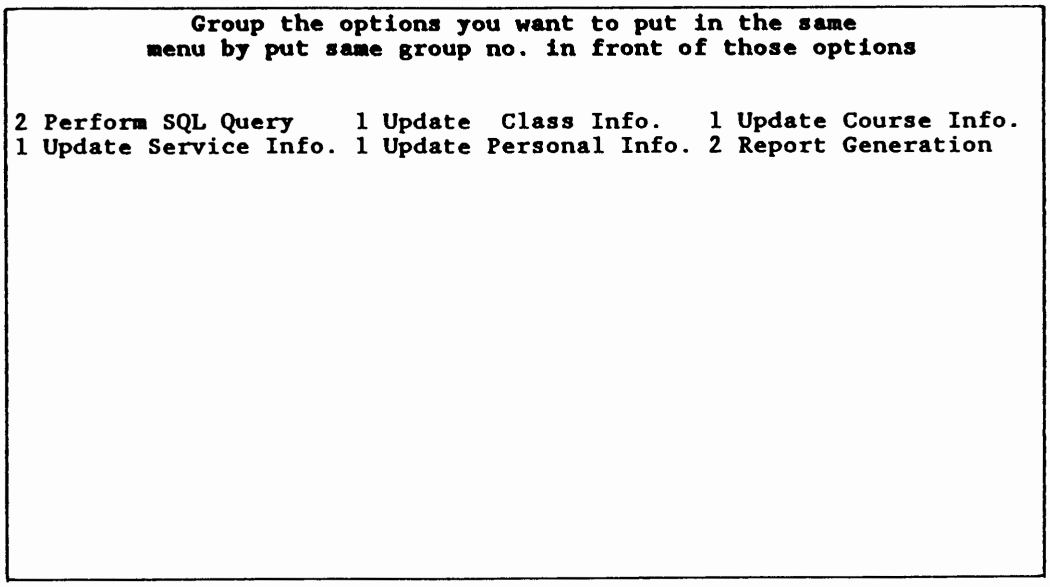
Processing List- Level 2 menus (Figure 26)



```
             Group the options you want to put in the same
            menu by put same group no. in front of those options


1 Add New Class            1 Delete Class Info.     1 Modify Class Info.
2 Add New Course           2 Delete Course Info.    2 Modify Course Info.
3 Add Service Info.        3 Delete Service Info.   3 Modify Service Info.
4 Add New Faculty          4 Delete Faculty Member  4 Modify Personal Info
5 Faculty Teaching Rpt.5 Courses Report            5 Classes Report
_ Perform SQL Query
```

Figure 25. Processing List 1 - Faculty Service Database



```
           Group the options you want to put in the same
         menu by put same group no. in front of those options


2 Perform SQL Query    1 Update  Class Info.    1 Update Course Info.
1 Update Service Info. 1 Update Personal Info. 2 Report Generation
```

Figure 26. Processing List 2 - Faculty Service Database

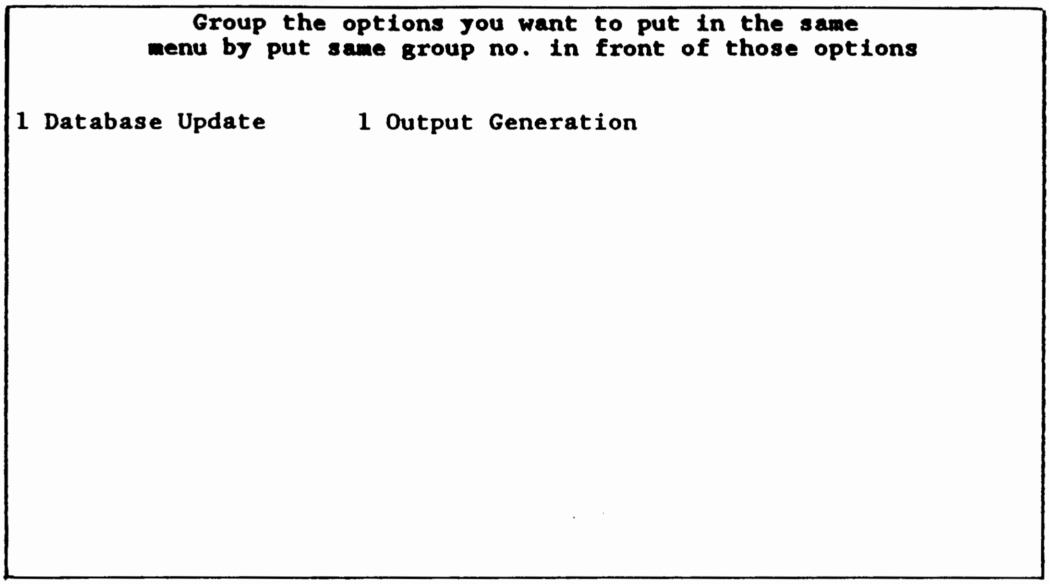
Group no	Menu name	Heading
1	up_menu	Database Update
2	outputs	Output Generation

Processing List - Level 3 menus (Figure 27)

Group no	Menu name	Heading
1	main	Faculty Service Database

Object Specification Dictionary

Appendix D is the listing of the object specification dictionary that is generated by the Intelligent Database Development Aid after the development process.


Group the options you want to put in the same
menu by put same group no. in front of those options

1 Database Update   1 Output Generation

Figure 27. Processing List 3 - Faculty Service Database

V. SUMMARY AND CONCLUSION

Summary

To help cope with the current software crisis, organizations are beginning to recognize the potential for end-users to function as their own application developers. This study explores the feasibility of using the object-oriented concept to develop an intelligent application system development aid and illustrates the development and implementation of such a system. The literature review summarizes studies in the area of end-user application development, identifies the advantages and limitations of this application development approach, and reviews the research in the area of intelligent development aid and object-oriented concepts. It was found that the latter two techniques have great potential for the implemention of an intelligent end-user development environment. The intelligent development aid and object-oriented concepts were therefore integrated into the development of the conceptual model for the database application development environment. The result is a system architecture for an intelligent system development aid called Intelligent Database Application Development Aid. The Intelligent Database Application

Development Aid is capable of providing intelligent support in developing database application systems. In order to test the feasibiliy of this architecture, a prototype was constructed which conformed to the architecture. This prototype demonstrated the validity of the architecture for the development of an intelligent end-user development environment. A hypothetical Faculty Service Database case was used to show the development process and to illustrated an implementation using the Intelligent Database Application Development Aid.

Contributions of This Study

The architecture of the Intelligent Database Application Development Aid represents an integrated application development environment with convenient features for many development considerations. The prototype, which was based on this architecture, demonstrated that the conceptual model can be implemented and that the resulting system does exhibit the kind of interaction process and design aid capabilities expected.

The principal contribution of this study is the integration of the intelligent (knowledge-based) development aid and object-oriented concepts. The integration provides a development environment for end-users who have little or no system development knowledge to perform system development tasks.

The Intelligent Database Application Development Aid represents a perspective on application development which is different from the perspective of the majority of computer-assisted development aids existing today. In the prototype system, the use of the object-oriented paradigm to mimic an end-user's model of reality for the work environment was demonstrated. Thus, both the development aid and the user's problem description can be based on the same model. In other words, the development aid can directly support the user's problem description without having it translated by professional developers. Furthermore, intelligent assistance is provided by the IDADA to help end-users develop a better design for their system. For example, IDADA provides a normalization facility to help users normalize their database into third normal form files to avoid anomalies. These efforts minimize the knowledge requirement of application development and thus eliminate the programmer "middle man" in the application system development process.

A second contribution of the study is that the presented architecture provides a very high level interface for the end-user developers. In the IDADA, all the instances in the user's model of reality are represented as objects in the development aid. This automatically provides users with a friendly interface for their development effort. In other words, IDADA uses the user's model of reality as its interface design. This may significantly reduce the requirement of user training as well as fear of computer usage.

<u>The last major contribution is that this study establishes a base for managing software assets in an organization</u>. As previouly pointed out, when compared to traditional functional decomposition, the object-oriented decomposition has higher reusability (Meyer, 1987). The use of IDADA breaks the original system development life cycle into two different development processes. The IDADA development process (Figure 10) is performed mainly by MIS professional, and the application development process (Figure 17) is performed by users with assistance from IDADA. Their responsibilities - shown in Figure 27 - are clearly defined by the presented architecture. In addition, these two processes, IDADA development and application development, interface with each other by the nonprocedural object specification dictionary, allowing end-user developers and MIS personnel to work with minimal interference with each other.

Suggestions for Further Research

Results of this study provide an excellent basis for further research, which may be conducted in two areas:

1. Enhancement of the current study; and
2. Extension of the current study.

To alleviate limitations of the current study, several areas appear to be promising for further investigation.

First, system performance. The processing performance of the IDADA suffered from two factors: first, the use of object-oriented programming; and second, the use of prolog as the implementation language. Object-oriented programming causes lower performance since all the processing has to be performed through massage sending (Stefik & Bobrow, 1986). Prolog programming requires programmers to write programs by describing known facts and relationships about a problem (Clocksin & Mellish, 1984). The computer then runs the program by applying certain logic on the program codes. This approach may cause unnecessary loops or processings and therefore lower the performance. The performace may be improved at least three ways; first, by reexaming the prolog programs to eliminate unnecessary processing; second, by compiling the object class library and intelligent development aid into object programs instead of using an interpreter; third, by using a lower level language as the implementation language.

Second, the development of a more sophisticated IDADA. Because of the limitation of resources, the system prototype developed in this study does not provide complete processing capabilities, as a real database system have. To develop a full-functioning development aid, methods in the object class library need to be more sophisticated and increased in number. For example, in the prototype, the relationship between each view processing facility and database file is a one-on-one relationship. In a more advanced system prototype both one-to-many and many-to-one relationships should be supported.

Third, the development of muti-user IDADA. Although users in the same work environment may have similar models of reality, it is possible for different users to have different models. Under this circumstance, the IDADA should be able to build different views to accommodate different user models. Since the work environment is the same, the basic methods should be the same for every user, but different object classes and method lists should be built to represent different views.

Fourth, the development of requirements analysis for object-oriented development. As discussed in Chapter Two, the object-oriented paradigm does not provide a comprehensive system development methodology. It focuses upon the design and implementation stage of the system development life cycle. It is therefore necessary to couple object-oriented development with appropriate requirements analysis and specification techniques (Booch, 1986). Further studies are required to identify and/or develop these techniques.

Perhaps the most obvious extension to this study would be the implementation of more prototype systems. This would further validate the conceptual model and extend the research concerning the effects of the development aid in different application situations. Furthermore, additional implementations would likely lead for some degree of refinement and modification of the architecture, resulting in a more pratical design of the IDADA.

In addition to providing additional implementations, a controlled experiment focusing specifically on problem formulation would be interesting and valuable. After a more sophisticated IDADA prototype is built, a study should be conducted to compare the prototype to a traditional database management system. The major outcome of such a study would be evaluations of the effects and contributions of the IDADA to system development productivity.

Finally, although the IDADA focuses on database application, its conceptual model can be applied to any application development such as Decision Support Systems and Expert Systems. Studies in different application areas may contribute to the development of an end-user development environment which will enable user developers to create different kinds of application systems.

BIBLIOGRAPHY

Abbott, R. "Report on Teaching Ada," Science Applications, Inc., Rep. SAI-81-312WA, Dec. (1980).

Aggarwal, S., Barbara, D. and Meth, K. Z. "A Software Environment for the Specification and Analysis of Problems of Coordination and Concurrency," IEEE Trans. on Software Engineering, Vol. 14, No. 3, (March 1988).

Alford, M. W. "Software Requirements Engineering Methodology (SREM) at the Age of Two," (Proceeding COMPSAC 1978, IEEE, Nov. 1978, pp. 332-339.

An Introduction to Arity Prolog, Arity Corp. 1986.

Appleton, D. S. "Information Asset Management." Datamation, pp 71-76, Feb., 1986.

Arity Prolog Programming Language, Arity Corp. 1986.

Barder, J., D. Hannaford, A. Cochran, and J. Edwards "Intellipse: Towards Knowledge Based Tools for The Design of Commercial Data Processing Systems." Information and Software Technology, Vol. 29, No. 8, pp 431-439, Oct., 1987.

Batini, C., Demo, B. and DiLeva, A. "A Methodology for Conceptual Schema Design of Office Databases," Infomation Systems, Vol. 9, No. 3, pp. 251-263, (1984).

Benson, D. H. "A Field Study of End-User Computing: Findings and Issues," MIS Quarterly, Vol. 9, No. 3, (Dec. 1983), pp. 35-45.

Bharath, Ramachandran 'Logic Programming : A Tool for MS/OR ?' INTERFACES, 16: 5, Sep.-Oct. 1986, pp. 80-91.

Biggerstaff, T. and C. Richter "Reusability Framework, Assessment, and Directions." IEEE Software, pp 41-49, March, 1987.

Blaha, M. R., Premerlani, W. J. and Rumbaugh, J. E. "Relational Datadase Design Using an Object-Oriented Methodology," Comm. of the ACM, Vol. 31, No. 4, (1988).

Boehm, B. W. "Software and its Impact: A Quantitative Assessment," Datamation, Vol. 19, No. 4, (May 1973), pp. 48-59.

Bohem-Davis, D. and Ross, L. "Approach to Structuring the Software Development," (Process. Report, GEC/DIS/TR-84-B1 V-1, General Eletric Co., Oct. 1984).

Booch, G. "Object-Oriented Development." IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, pp 211-221, Feb., 1986.

Borgida, A. "Features of Language for the Development of Information Systems at the Conceptual Level," IEEE Software, Vol. 2, No. 1, (Jan. 1985), pp 63-72.

Brown, D. "A Decision Support System for Reliable Software Develoipment." IEEE Transactions on Systems, Man, and Cybernetics, Vol. 17, No. 1, pp 86-91, Jan./Feb., 1987.

Bryce, M., "The IRM Idea" DATAMATION, April 15, 1987, pp. 89-92.

Building Arity/Prolog Application, Arity Corp. 1986.

Case, A. F., Information Systems Devenlopment, New Jersey, Prentice-Hall, 1986.

Choobineh, J., M. V. Mannino, J. F. Nunamaker, and B. R. Konsynski, "An Expert Database Design System Based on Analysis of Forms." IEEE Transactions on Software Engineering. Vol. 14, No. 2, Feb. 1988, pp 242-253.

Clocksin, W. F. and C. S. Mellish Programming in Prolog, Berlin, Springer-Verlag, 1984.

Covington, M. A. Programming in Prolog:, ACMC Research Report, The University of Georgia, 1986.

Covington, M. A. Prolog on The IBM PC, ACMC Research Report, The University of Georgia, 1986.

Covington, M. A. Expressing Procedural Algorithms in Prolog, ACMC Research Report, The University of Georgia, 1986.

Covington, M. A. On Looping in Prolog:, ACMC Research Report, The University of Georgia, 1986.

Cox, B. J. Object-Oriented Programming: Evolutionary Approach. Reading, Mass., Addison-Wesley, (1986).

De, P. and A. Sen "A New Methodology for Database Requirements Analysis." MIS Quarterly, pp 179-193, Sep., 1984.

Dickson, G. W., Leitheser, R.L., Wetherbe, J.C., and Nechis, M. "Key Information Systems Issues for the 1980's." MIS Quarterly, Vol. 8, No. 3, 1984, pp. 135-159.

Dock, D. R. and R. A. Kirsch II "A Relational Information Resource Dictionary System" Communication of the ACM, Vol. 30, No. 1, pp 48-61, Jan., 1987.

Dolotta, T. A. et. al. Data Processing in 1980-1985. John Wiley and Sons, NY, (1976).

Dunning, B. B. "Expert System Support for Rapid Prototyping of Conventional Software," (Proceeding Autotestcon '85, IEEE NY, USA, Oct. 1985).

Dyer, C. A. "Expert System in Software Maintainability," (Proceeding Annual Reliability and Maintainability Symp. San Francisco, CA, USA, Jan. 1984), pp 295-299.

Easterby, R. "Trillium: an Interface Design Prototyping Tool." Information and Software Technology, Vol. 29, No. 4, pp 207-213, May, 1987.

Feuche, M. "Object DBMS for the '90s," MIS Week, (March 21, 1988).

Garner, B. J. "Expert Systems: from Database to Knowledge Base." Information and Software Technology, Vol. 29, No. 2, pp 60-65, March 1987.

Gerstein, M. and H. Reisman "Creating Competitive Advantage With Computer Technology." Journal of Business Strategy, pp 53-60, Summer, 1982.

Gibson, M. L., "Component of User Work Station." Journal of Systems Management, Feb. 1988, pp 6-14.

Gibson, M. L., and C. T. Hughes, "The User Designer/Developer and the User Work Station." Journal of Systems Management, Feb. 1988, pp 36-41.

Gibson, M. L. and Corman, L. S. "User Programmer and Costs of the Misinformed User," Journal of Systems Management, (May 1987), pp. 23-29.

Giddings, N. and Colburn, T. "An Automated Software Design Evaluater," (Proceeding Annual Conferrence-ACM 84 San Francisco, CA, (Oct. 1984).

Glorfeld, L. W. "Exploring Database Concepts Using Prolog." The Journal of CIS, pp 2-4, Summer 1987.

Goldberg, A. T. "Knowledge-Based Programming: A Survey of Program Design and Construction Techniques." IEEE Transaction, Vol. SE-12, No. 7, pp 752-, July, 1986.

Goldberg, A. Smalltalk - The Interactive Programming Environment. Reading, Mass., Addison-Wesley, (1984).

Hall, P. A. V. "Software Components and Reuse - Getting More Out of Your Code." Information and Software Technology, Vol.29, No. 1, pp 38-43, Feb., 1987.

Hamilton, S. and B. Ives "Knowledge Utilization Among MIS Researchers." MIS Quarterly, pp 61-77, Dec., 1982.

Henry, S. M. "Information Flow Metrics for the Evolution of Operating Systems' Structure," (Ph. D. dissertation, Iowa State University, 1979).

Hewitt, C. " Viewing Control Structures as Patterns of Passing Messages." Artificial Intelligence, 8, pp 323-364, 1977.

Hogger, C. J. Introduction to Logic Programming, London, Academic Press, 1984.

Holapple, C., Shen, S. and Whinston, A. " A Consulting System for Database Design," Information Systems, Vol. 7, No. 3, pp. 281-296, (1982).

Hughes, C. T., "Managing the Work Station Environment." Journal of Systems Management, Feb. 1988, pp 30-35.

Hull, M. C. E. and F. G. Metcalfe "An Automated systems Design Tool." Information and Software Technology, Vol. 29, No. 5, pp 257-264, June, 1987.

Hurst, R. S. Frewin, G. D. and Hamer, P. G. "A Rule-Based Approach to a Software Production and Maintenance Management System," Esprit '84, North-Holland, Amsterdam, (1985), pp. 127-144.

Jarke, M. and J. Shalev "A Database Architecture for Supporting Business Transactions." Journal of MIS. Vol. 1, No. 1, pp 63-80, Summer, 1984.

Jarvepaa, S. L., G. W. Dickson and G. DeSanctis "Methodological Issues in Experimental Research: Experiences and Recommendations" MIS Quarterly, pp 141-156, June, 1985.

Journal of Systems Management Report, "Knowledge Workers and Office Automation," Journal of Systems Management, (Dec, 1985).

Kamel, R. F. "Effect of Modularity on System Evolution." IEEE Software, pp 48-54, Jan., 1987.

Karimi, J. "An Automated Software Design Methodology Using CAPO," Journal of MIS, Vol. 3, No. 3. (1987).

Karimi, J., and Konsynski, B. R., "An Automated Software Design Assist" IEEE Transactions on Software Engineering. Vol. 14, No. 2, Feb. 1988, pp 194-210.

Kasper, G.M. "The Effect of User-Developed DSS Applications on Forcasting Decision-Making Performance in an Experimental Setting." Journal of MIS, Vol.11, No. 2, pp. 26-39, 1985.

Kozar, K. L. and Mahlum, J. M. "A User Generated Information System: An Innovative Development Approach," MIS Quarterly, (June 1987), pp. 163-173.

Kroenke, D. M. "Teaching Database Processing With an Object Orientation." SRA MIS Newsletter, Vol. 1, No. 1, Fall, 1987.

Leavitt, Don "The Proper Design Tools Can Bring Improved Productivity." Software News, pp 80-83, Feb., 1985.

Leitheiser, R. L. and Wetherbe, J. C. "Service Support Levels: An Organized Approach to End-User Computing," MIS Quarterly, (Dec. 1986), pp. 337.

Leung, C. H. E. "A Knowledge-dase for Effective Software Specification and Maintenance," (Proceeding Third International Workshop Software Specification and Des. London, UK, Aug. 1985), pp. 139-142.

Lientz, B.P., Swanson, E.B., and Tompkins, G.E. "Characteristics of Application Software Maintenance," Communications of the ACM, Vol. 21, No. 6, June 1978, pp. 466-471.

Luo. D and Yao, S.B. "Form Operation by Example-A Language for Office Information Processing," (Proceeding ACM SIGMOD, 1981, pp 212-233)

Madhavji, N.H. "Fragtypes: A Basis for Programming Environments," IEEE Trans. on Software Engineering, Vol. 14, No. 1, pp. 85-97, (1988).

Madhavji, N.H., Pinsonneault, L. and Toubache, K. "Modula-2/MUPE-2: Language and Environment Interactions," IEEE Software, Vol. 3, No. 6, pp. 7-17, (1986).

McCracken, D. D. "Software in the 1980's: Perils and Promises," Compuiterworld, Vol. 14, No. 38, (Sep. 1980).

McLean, E. R. "End-Users as Application Developers," MIS Quarterly, (Dec. 1979), pp. 37-46.

Meyer, B. "Reusability: The Case for Object-Oriented Design." IEEE Software, pp 50-64, March, 1987.

Munro, A. T. D. "SADIST: an Interactive Editor for Structure Analysis" Computer Graphics Forum, Vol. 2, No. 2/3 (1983), pp. 104-114.

Navathe, S. B. and M. Schkolnich "View Representation in Logical Database Design." pp 144-156,

Nolan, R.L. and J. C. Wetherbe "Toward a Comprehensive Framework for MIS Research." MIS Quarterly, pp 1-19, June, 1980.

Prieto-Diaz, R. and Freeman, P. "Claaifying Software for Reusability." IEEE Software, pp 6-16, Jan., 1987.

Ramamoorthy, C. V. Prakash, A. Tsai, W. and Usuda, Y. 'Software Engineering: Problems and Perspectives' IEEE Computer, (October 1984), pp 191-209

Rentsch, T. "Object-Oriented Programming," SIGPLAN Notices, Vol. 17, No. 9, (1982), pp. 51.

Richards, R. M., and J. C. Windsor, "Documentation in a User Work Station Environment." Journal of Systems Management, Feb. 1988, pp 23-29.

Rivard, S. and Huff, S. L. "An Empirical Study of Users as Application Developers," Information and Managfement, Vol. 8, No. 2, (1985), pp. 89-102.

Sammet, J. E. Programming Language: History and Fundamentals. Prentice Hall, Englewood Cliffs, NJ, (1969).

Schmucker, K. Object-Oriented Programming for the Macintosh, New Jersey, Hayden, 1986.

Shu, N. C., Lum, V. Y., Tung, F. C. and Chang, C. L., "Specification of Forms Processing and Business Procedures for Office Automation," IEEE Trans. Software Engineering, Vol. SE-8, No. 5, pp. 499-511, Sep. 1982.

Simon, H. A. "Whether Software Engineering Needs to Be Artificial Intelligent." IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, pp 726- , July, 1986.

SPECIF, Institut de Genie Logical, Toulouse, France (1984).

Spence, J. W. "End-User Computing - The Human Interface." Journal of Systems Management, Feb. 1988, pp 15-21.

Stabeer, E. P. "Objectj-Oriented Programming in Prolog." AI Magazine, pp 46-57, Oct., 1986.

Stefik, Mark and D. G. Bobrow "Object-Oriented Programming: Themes and Variations." The AI Magazine, pp 40-62, 1986.

Stein, J. and Maier, D. "Concepts in Object-Oriented Data Management," Database Programming and Design, (April 1988), pp. 58-67.

Teichroew, D. and Hershey, E. "PSL/PSA: A computer aided technique for structured documentation andf analysis of information processing systems," IEEE Trans. Software Engineering, Vol. SE-3, No. 1, pp. 41-48, Jan. 1977.

The Arity Screen Design Toolkit, Arity Corp. 1986.

The Report of Alvey Committee HMSO, London UK (1982)

Tsichritzis, D. "Form Management," Comm. ACM, Vol. 25, No. 7, pp. 453-478, July 1982.

Vitalari, N. P. "Knowledge as a Basis for Expertise in System Analysis: An Empirical Study." MIS Quarterly, Vol. 9, No. 3, (Sep. 1985), pp. 221-241.

Wong, W. "Prolog - A Language for Artificial Intelligence." agazine, pp 247-263, Oct., 1986.

Wartik, S. P. "Fillin: A Reusable Tool for Form-Oriented Software." IEEE Software. pp 61-69, March, 1986.

Wetherbe, J. C. and Leitheiser, R. L. "Information Centers: A Survey of Services, Decisions, Problems, and Successes," Journal of Information Systems Management, Vol. 2, No. 3. (Summer 1985), pp. 3-10.

Wiederhold, G. "Databases." Computer, pp 211-223, Oct., 1984.

Wiederhold, G. "Knowledge and Database Management." IEEE Software, pp 63-, Jan., 1984.

Zaniolo, C. "Object-Oriented Programming in Prolog." Internation Logic Programming Spectrum, pp 265-270, 1984.

Zaniolo, C., H. Ait-Kaci, D. Beech, S. Cammarata, L. Kerschberg, and D. Maier "Object Oriented Database Systems and Knowledge Systems." Expert Database Systems, ed Larry Kerschberg, pp 49-65, 1984.

APPENDICES

APPENDIX A

DATABASE DEVELOPMENT ENVIRONMENT

SOURCE LISTING

```
dbde :- [uti],
        introd2,
        restore,
        [dbde],
        prototyping.

pt :- prototyping.

prototyping :-
        dsp_option,
        get(Option),
        next(Option).

dsp_option :-
        cls,
        tmove(6,10),
        write('.. Operation Manager'),
        tmove(8,13),
        write('1) Test Run the Data Base System'),
        tmove(10,13),
        write('2) Build up Operational Data Base'),
        tmove(12,10),
        write('.. Development Manager'),
        tmove(14,13),
        write('a. Data Base Designer'),
        tmove(16,13),
        write('b. Report Revisor'),
        tmove(18,13),
        write('c. Screen Revisor'),
        tmove(20,13),
        write('d. Menu Revisor'),
        tmove(22,10),
        write('x) Exit'),
        tmove(3,15),
        write('Choice an Option : ').

set_up_1 :-
         cls,
         tmove(5,25),
         write('Set The Data Base To Work'),
         tmove(10,20),
         write('Please Name This Data Base System : '),
         read_line(0, Begn),
         atom_string(Begn1,Begn),
         tmove(12,20),
         write('Please Enter The Main Menu Name : '),
         read_line(0, Menu),
         atom_string(Menu1,Menu),
         asserta((run :- send(menu(Menu1),menu) )),
         save(Begn1),
         tmove(15,20),
         write('You Now Can Run The System by Type'),
         tmove(16,25),
         write('restore('),
         write(Begn1),
         write('). and'),
         tmove(17,25),
         write('run.'),
         tmove(18,25),
         write('under api command entry mode'),
         get0(A).

next(`1) :-
        [start],
        start,
        go.

next(`2) :-
        [start],
        start1,
        set_up_1.

next(`x).

next(`X).

next(`a) :-
        [aid],
        aid.

next(`b) :-
        [idb_out],
        start.

next(`c) :-
        [idb_scrn],
        start.

next(`d) :-
        [idb_menu],
        start.

next(`A) :-
        [aid],
        aid.

next(`B) :-
        [idb_out],
        start.

next(`C) :-
        [idb_scrn],
        start.

next(`D) :-
        [idb_menu],
        start.

introd2 :- introd($WELCOME TO END USER DATA BASE DEVELOPMENT ENV.$)
```

```
              [$   This prototype is an Object-Oriented System for$,
$ End User Data Base Development. It was developed to demostrate the$,
$ feasibility of appling Object-Oriented Development in a End User$,
$ Designer/Developer Environment.  $]).
```

APPENDIX B

OBJECT CLASS LIBRARY SOURCE LISTING

```
/* MENU-DRIVEN SQL QUERY INTERFCE                                    */
/* SQL QUERY - chang from SQLDEMO.ARI                                */
/*              Copyright (C) 1986 Arity Corporation.                */


run_query :-
        cls,
        set_up_qrys,
        repeat,
        gc(full),
        cls,
        tmove(0,2),
        write('SELECT FROM '),
        tables(Y),
        rest(Y),
        done.
rest([]) :- !.
rest(Y) :-
        columns(Y,L,Z,X),
        crest(Y,L,Z,X), !.

crest(Y,_,_,[]) :-
        !,
        tmove(5,0),
        exec_it([],Y,[],[],[]).
crest(Y,L,Z,X) :-
        where(L,W),
        group(Z,G,G1),
        order(Z,O,G1),
        tmove(5,0),
        exec_it(X,Y,W,G,O).
write_string_list(Handle,A) :-
        write_s_list(Handle,A,'').

write_s_list(Handle,[],Comm) :-  !.

write_s_list(Handle,[A,B,C|List],Comm) :-
        write(Handle,Comm),
        write(Handle,A),
        write(Handle,','),
        write(Handle,B),
        write(Handle,','),
        write_string(Handle,C),
        fit(Comm1, ','),
        write_s_list(Handle, List, Comm1).

/* do_banner :-
        cls,
        tmove(10,23),
        write('Arity/SQL Demo program'),
        tmove(11,23),
        write('Copyright (C) 1986, Arity Corporation'),
        set_up_qrys,
        get0_noecho(X).*/
set_up_qrys :-
```

```
        eraseall(select),
        eraseall(sw),
        eraseall(sg),
        eraseall(so),
        eraseall(swg),
        eraseall(swo),
        eraseall(sgo),
        eraseall(swgo),
        make_query($sel(X,Y);=select #X from #Y;$,Sel),
        recorda(select,Sel,_),
        make_query(
         $sw(X,Y,Z);=select #X from #Y where %Z;$,Whr),
        recorda(sw,Whr,_),
        make_query(
         $sg(X,Y,Z);=select #X from #Y group by #Z;$,Sg),
        recorda(sg,Sg,_),
        make_query(
         $so(X,Y,Z);=select #X from #Y order by #Z;$,So),
        recorda(so,So,_),
        make_query(
         $swg(W,X,Y,Z);=select #W from #X
          where %Y group by #Z;$,Swg),
        recorda(swg,Swg,_),
        make_query($swo(W,X,Y,Z);=select #W from #X
                    where %Y order by #Z;$,Swo),
        recorda(swo,Swo,_),
        make_query($sgo(W,X,Y,Z);=select #W from #X
                    group by #Y order by #Z;$,Sgo),
        recorda(sgo,Sgo,_),
        make_query($swgo(V,W,X,Y,Z);=select #V from #W
                    where %X group by #Y order by #Z;$,Swgo),
        recorda(swgo,Swgo,_),
        make_query($sa(X);=select * from #X;$,Sa),
        recorda(sa,Sa,_).


tables([H|T]) :-
        findall(itm(X,6),table(X,_,_,_),L),
        tmove(4,0),
        write('From clause'),
        get_list([itm('DONE',6)|L],_,_,List,6),
        List = [H|T],
        tmove(4,0),
        write('           '),
        tmove(0,9),
        write(' FROM '),
        write(H),
        write_list(T), !.
tables([]).

write_list([]).
write_list([H|T]) :-
        write(', '),
        dwrite(H),
        write_list(T).
```

```
get_list(X,L2,L1,L,W) :-
        region_c((3,10),(24,30),R),
        get_lst(X,L2,L1,L,W,1),
        region_c((3,10),(24,30),R),
        !.

get_lst(X,L2,L1,L,W,C) :-
        box_vmenu(5:10,X,W,0,Choice),
        unhighlight(W),
        handle_choice(X,Choice,L2,L1,L,W,C).

handle_choice(X,'DONE',[],[],[],_,_).
handle_choice(X,'Expression',L1,L2,L3,W,C) :-
        expression(X,L1,L2,L3,W,C).
handle_choice(X,T:C,[itm(T:C,W)|L1],[.(T,C)|L2],
                      [.(T,C)|L],W,C1) :-
        inc(C1,C2),
        get_lst(X,L1,L2,L,W,C2).
handle_choice(X,C,[itm(C,W)|L1],[C|L2],[C|L],W,C2) :-
        inc(C2,C1),
        get_lst(X,L1,L2,L,W,C1).

expression(X,[itm(con(C),W)|L1],[A|L2],[B|L],W,C) :-
        tmove(20,0),
        write('Enter the expression : '),
        read_line(0,A),
        tmove(20,0),
        write('
                                                      '),

        A \= $$,
        error(se,A),
        scalar_expression(A,B),
        inc(C,C1),
        get_lst(X,L1,L2,L,W,C1).
expression(X,L1,L2,L,W,C) :-
        get_lst(X,L1,L2,L,W,C).

error(_,_).
error(se,A) :-
        tmove(20,0),
        write('Error in expression':A),
        get0_noecho(_),
        tmove(20,0),
        write('
                                                     '),

        fail.
error(wh,A) :-
        tmove(5,0),
        write('Error in condition':A),
        get0_noecho(_),
        tmove(5,0),
        write('
                                                     '),

        fail.

where((R,C),Y) :-
        tmove(5,0),
        write('Enter your where clause : '),
        read_line(0,X),
        tmove(5,0),
        write('
                                                '),
        X \= $$,
        error(wh,X),
        predicate(X,Y),
        tmove(R,C),
        write(' WHERE '),
        write(X), !.
where(_,[]).

where1((R,C),X) :-
        tmove(5,0),
        write('Enter your where clause : '),
        read_line(0,X),
        tmove(5,0),
        write('
                                               '),
        X \= $$,
        error(wh,X),
        tmove(R,C),
        write(' WHERE '),
        write(X), !.
where1(_,[]).

get_columns([],[]).
get_columns([H|T],Y) :-
        findall(itm(H:X,15),column_table(X,H,_,_,_),Cols),
        !,
        get_columns(T,C1),
        append(Cols,C1,Y).

append([],L,L).
append([H|T],L,[H|L1]) :- append(T,L,L1).

columns(L0,(R,C),L,L1) :-
        get_columns(L0,Cols),
        tmove(4,0),
        write('Select clause'),
        get_list([itm('DONE',15),
                 itm('Expression',15)|Cols],L,L2,L1,15),
        tmove(4,0),
        write('             '),
        write_c(L2),
        write_t(L0),
        tget(R,C),
        !.
columns(_,_,_,[]).

write_c([H|T]) :-
        tmove(0,9),
        dwrite(H),
        write_list(T), !.

write_t([H|T]) :-
```

```
        write(' FROM '),
        write(H),
        write_list(T),
        !.

group(Cols,G1,L) :-
        remove_expressions(Cols,Cols1),
        tmove(4,0),
        write('Grouping clause'),
        get_list([itm('DONE',15)|Cols1],_,_,G1,15),
        tmove(4,0),
        write('                    '),
        write_g(G1,L),
        !.

remove_expressions([],[]).
remove_expressions([itm(con(_),_)|T],T1) :-
        !,
        remove_expressions(T,T1).
remove_expressions([H|T],[H|T1]) :-
        remove_expressions(T,T1).

write_g([],(2,9)).
write_g([H|T],(R,C)) :-
        tmove(2,10),
        write('GROUP BY '),
        dwrite(H),
        write_list(T),
        tget(R,C),
        !.

order(Cols,G1,L) :-
        tmove(4,0),
        write('Order by clause'),
        get_list([itm('DONE',15)|Cols],_,_,G1,15),
        tmove(4,0),
        write('                    '),
        write_o(G1,L),
        !.

write_o([],_).
write_o([H|T],(R,C)) :-
        tmove(R,C),
        write(' ORDER BY '),
        dwrite(H),
        write_list(T),
        !.


exec_it(A,B,C,D,E) :-
        execit(A,B,C,D,E,T,Cap),
        exec_capsule(T,Cap,Key,Err,Var),
        report_error(Err,Var),
        !.
```

```
execit([],Y,_,_,_,sa(Y),Cap) :-
        recorded(sa,Cap,_).
execit(X,Y,[],[],[],sel(X,Y),Cap) :-
        recorded(select,Cap,_).
execit(X,Y,[],[],W,so(X,Y,W),Cap) :-
        recorded(so,Cap,_).
execit(X,Y,[],W,[],sg(X,Y,W),Cap) :-
        recorded(sg,Cap,_).
execit(X,Y,[],G,O,sgo(X,Y,G,O),Cap) :-
        recorded(sgo,Cap,_).
execit(X,Y,W,[],[],sw(X,Y,W),Cap) :-
        recorded(sw,Cap,_).
execit(X,Y,W,[],O,swo(X,Y,W,O),Cap) :-
        recorded(swo,Cap,_).
execit(X,Y,W,G,[],swg(X,Y,W,G),Cap) :-
        recorded(swg,Cap,_).
execit(X,Y,W,G,O,swgo(X,Y,W,G,O),Cap) :-
        recorded(swgo,Cap,_).

report_error(0,_).
report_error(Err,Var) :-
        nl,
        nl,
        write('Error number':Err),
        tab(3),
        write('Values':Var).

done :-
        nl,
        nl,
        write('Another SQL SELECT ? (y/n) '),
        doneaux.

doneaux :-
        get0_noecho(C),
        handle_it(C).

handle_it(`y) :- !, fail.
handle_it(`n).
handle_it(_) :-
        put(7),
        doneaux.



dwrite(T:C) :-
        !,
        write(T),
        put(`.),
        write(C).
dwrite(.(T,C)) :-
        !,
        write(T),
        put(`.),
        write(C).
```

```
dwrite(con(X)) :-
        !,
        cwrite(X).
dwrite(X) :-
        write(X).

cwrite(X) :-
        string(X),
        !,
        put(`'),
        write(X),
        put(`').
cwrite(X) :-
        write(X).
```

```
/* Object-oriented  programming message sending program  */

send(Obj,Msg) :-
        isa_chain(Obj,Obj1),
        Obj1 with Mthds,
        get_method(Msg,Mthds,Mthd),
        call(Mthd).
get_method(Msg,[First|Rest],Mthd) :-
        fact_or_rule(Msg,First,Mthd), !.
get_method(Msg,[_|Rest],Mthd) :-
        get_method(Msg,Rest,Mthd).
fact_or_rule(Msg,Msg,true).
fact_or_rule(Msg,(Msg :- Body),Body).
isa_chain(Obj,Obj).
isa_chain(Obj1,Obj3) :-
        Obj1 isa Obj2,
        not(Obj1 = Obj2),
        isa_chain(Obj2,Obj3).
```

```
/* Object Class Library                                                */
/*             - Methods List                                          */

fms(Menu_name) :- menu(Menu_name).

menu(Menu_name) :-
        repeat,
        trans_to(Menu_name, M_static, M_active),
        dialog_run((0,0),(24,79),clear,none,'',
                   1,M_static,M_active,Exit),
        dialog_val(M_active, box, X),
        recorded(Menu_name, menu_choice(X, Action),_),
        not(call(Action)),
        ifthen(Action \== send(menu(return),return),
              do_assert).

screen(Screen_name) :-
        trans_to(Screen_name, S_static, S_active),
        dialog_run((0,0),(24,79),clear,none,
                   '',1,S_static,S_active,Exit).

add(Vf_name) :-
       trans_to(Vf_name, Static, Active),
       set_null(Active),
       recorded(Vf_name, operation('add',Title),_),
       recorded(Static, text(2,20,_), Ref),
       replace(Ref, text(2,20,Title)),
       dsp_screen(Vf_name),
       get_values(Active, Value_list),
       send(b_table(Vf_name), add(Value_list)).

del(Vf_name) :-
       trans_to(Vf_name, Static, Active),
       set_null(Active),
       recorded(Vf_name, operation('del',Title),_),
       recorded(Static, text(2,20,_), Ref),
       replace(Ref, text(2,20,Title)),
       dsp_screen(Vf_name),
       get_values(Active, Value_list),
       send(b_table(Vf_name),
            retrive(Value_list, Value_list1)),
       set_value(Active, Value_list1),
       dsp_screen(Vf_name),
       ask,
       send(b_table(Vf_name), del(Value_list1)).

mod(Vf_name) :-
       trans_to(Vf_name, Static, Active),
       set_null(Active),
       recorded(Vf_name, operation('mod',Title),_),
       recorded(Static, text(2,20,_), Ref),
       replace(Ref, text(2,20,Title)),
       dsp_screen(Vf_name),
       get_values(Active, Value_list),
       send(b_table(Vf_name),
        retract(row(_)), fail.

build_rpt_body(Report_name, Row, Pos_list) :-
        asserta(row(Row)),
        recorded(rpt_result, A, Ref),
        A =.. [P|List],
        P \== heading,
        row(Row2),
        retract(row(Row2)),
        retract(hold(Hold_list)),
        list_body(List, Hold_list, Row2, Pos_list, Row3),
        asserta(hold(List)),
        Row1 is Row3 + 1,
        asserta(row(Row1)),
        fail.

build_rpt_body(Report_name, Row, Pos_list) :- !.

list_body([],[],Row,Col,Row) :- !.

list_body([Field|F_list], [], Row,[Col|Col_list], Row1) :-
        tmove(Row,Col),
        write(Field),
        list_body(F_list, [], Row,Col_list, Row1).

list_body([Field|F_list], [Hold_field|H_list],
          Row,[Col|Col_list],Row1) :-
        Field == Hold_field,
        list_body(F_list, [], Row, Col_list,Row1).

list_body([Field|F_list],
          [Hold_field|H_list], Row,[Col|Col_list],Row1) :-
        Field \== Hold_field,
        Row2 is Row + 1,
        tmove(Row2,Col),
        write(Field),
        list_body(F_list, [], Row2, Col_list, Row1).

query :- run_query.

trans_to(Menu_n, Static_name, Active_name) :-
        atom_string(Menu_n,Menu_name),
        concat(Menu_name, $_static$, Static0),
        concat(Menu_name, $_active$, Active0),
        atom_string(Static_name, Static0),
        atom_string(Active_name, Active0).

dsp_screen(Lf_name, add, Scr_name) :-
        add_suf(Lf_name,$_a$,Scr_name),
        screen(Scr_name).

dsp_screen(Lf_name, del, Scr_name) :-
        add_suf(Lf_name,$_d$,Scr_name),
        screen(Scr_name).
```

```
add_suf(Atom_in, Str, Atom_out) :-
        atom_string(Atom_in, Str_in),
        concat(Str_in, Str, Str_out),
        atom_string(Atom_out, Str_out).

get_values(Key, End_list) :-
        retract(value_list(_)),
        asserta(value_list([])),
        parm_list(Key, End_list).

value_list([]).

parm_list(A_key, End_list) :-
        dialog_val(A_key, F, Value),
        value_list(List),
        retract(value_list(List)),
        asserta(value_list([Value | List])),
        fail.

parm_list(A_key, End_list) :-
        value_list(List),
        reverse_list(List, [], End_list),!.

reverse_list([],Y,Y) :- !.

reverse_list([X|Tail],Y,Z) :-
        reverse_list(Tail,[X|Y],Z).

build_ins(Value_list, Str, Sql_str, Rest_list) :-
        set_value(Str, Value_list, Sql_str, Rest_list).

build_del(Value_list, Str, Sql_str, Rest_list) :-
        set_value(Str, Value_list, Sql_str, Rest_list).

set_value(Str, [Value|Value_list], Sql_str, Rest_list) :-
        string_search($@^$,Str,A),
        A1 is A + 2,
        string_length(Str,T),
        A3 is T - A1,
        substring(Str,0,A,Pref),
        substring(Str,A1,A3,Suf),
        concat([Pref, Value, Suf], Str1),
        set_value(Str1, Value_list, Sql_str, Rest_list).

set_value(Sql, List, Sql, List) :- !.

build_comm(Str,List,Str) :-
        write('Error in Building SQL Command'),
        get0(_).
screen_process([Process,A,B,C,D,E | Tail]) :-
        processit(Process,A,B,C,D,E,T,Cap),
        exec_capsule(T,Cap,_,Err,Var),
        report_error(Err,Var),
        screen_process(Tail).
```

```
processit('select',A,B,C,D,E,T,Cap) :-
          execit(A,B,C,D,E,T,Cap).
processit('insert',A,B,C,D,E,T,Cap) :-
          insertit(A,B,C,D,E,T,Cap).
processit('delete',A,B,C,D,E,T,Cap) :-
          deleteit(A,B,C,D,E,T,Cap).
processit('update',A,B,C,D,E,T,Cap) :-
          updateit(A,B,C,D,E,T,Cap).

return :- fail.

dsp_screen(Vf_name) :- screen(Vf_name).

set_value(Active, Value) :- retract(list1(_)), fail.

set_value(Active, Value_list) :-
        asserta(list1(Value_list)),
        recorded(Active,efield(A,B,C,D,V,E),Ref),
        list1([Value | Value_list1]),
        retract(list1(_)),
        asserta(list1(Value_list1)),
        replace(Ref, efield(A,B,C,D,Value,E)),
        fail.

set_value(Active, List) :- !.

ask :-
        tmove(23,20),
        write($Enter 'Y' to confirm : $),
        get(Ans),
        determ(Ans).

determ(121) :- !.
determ(89) :- !.
determ(Ans) :- fail.

process_file(Vf_name, [], []) :- !.

process_file_i(Vf_name, List, [Insert | Insert_list]) :-
        build_ins(List, Insert, Command, Rest_list),
        exec_sql(Command),
        process_file(Vf_list, Rest_list, Insert_list).

process_file_d(Vf_name, List, [Delete | Delete_list]) :-
        build_del(List, Delete, Command, Rest_list),
        exec_sql(Command),
        process_file(Vf_list, Rest_list, Delete_list).

get_record([], [], New_list, New_list).

get_record(List, [File, Args | File_list],
            Work_list, New_list) :-
        find_value(List, Args, [], Sub_list, Rest_list),
        append([File], Sub_list, Rec),
        Record =.. Rec,
        recorded(File, Record, _),
        Record =.. Rec_list1,
        kick(Rec_list1, Rec_list2),
        reverse_list(Rec_list2, [], Rec_list3),
        append(Rec_list3, Work_list, Work_list1),
        get_record(Rest_list, File_list, Work_list1,
                   New_list).

find_value(List, 0, Work_list, Sub_list, List) :-
        reverse_list(Work_list, [], Sub_list).

find_value([A|List], Args, Work_list, Sub_list, Rest_list) :-
        A == $$,
        Args1 is Args - 1,
        find_value(List, Args1, [B | Work_list], Sub_list,
                   Rest_list).

find_value([A|List], Args, Work_list, Sub_list, Rest_list) :-
        A \== $$,
        Args1 is Args - 1,
        find_value(List, Args1, [A | Work_list], Sub_list,
                   Rest_list).

kick([A | Rec_list], Rec_list).

append([],L,L) :- !.

append([X|L1], L2, [X|L3]) :- append(L1,L2,L3).

set_null(Active) :-
        recorded(Active, efield(A,B,C,D,E,F), Ref),
        replace(Ref, efield(A,B,C,D,$$,F)),
        fail.

set_null(Active) :- !.

do_assert :-
        cls, tmove(10,20),
        write('NOTICE : Last Selection DID NOT Perform'),
        tmove(12,20),
        write('       Processes on Data Base'),
        get0(A), fail.
```

```
/* Operation Manager                                              */
/*          - Set up programs                                     */
/*            for Object-Oriented Application Dictionary          */

start :- cls,
         write('Load Object-Oriented Data Base System'),nl,
         [set_op], set_op,
         [o_o,dialog1,editfld,diccon,obj,objdb],
         write('Consulting Object Specification Dictionary'),
         nl,
         dic_consult('rpt.osd'),
         dic_consult('vpf.osd'),
         dic_consult('scrn.osd'),
         dic_consult('menu.osd'),
         sql_consult('bt.osd'),
         sql_consult('data.tst'),
         write('Load Query Module '),nl,
         [+query].

start1 :- cls,
         write('Load Object-Oriented Data Base System'),nl,
         [set_op], set_op,
         [o_o,dialog1,editfld,diccon,obj,objdb],
         write('Consulting Object Specification Dictionary'),
         nl,
         dic_consult('rpt.osd'),
         dic_consult('vpf.osd'),
         dic_consult('scrn.osd'),
         dic_consult('menu.osd'),
         sql_consult('bt.osd'),
         write('Load Query Module '),nl,
         [+query].

go :-
         cls,
         tmove(5,30),
         write('Test   Run'),
         tmove(10,20),
         write('Please input your main menu name : '),
         read_line(0, Begn),
         atom_string(Begn1,Begn),
         send(menu(Begn1), menu).
```

APPENDIX C

INTELLIGENT DEVELOPMENT AID SOURCE LISTING

```
/* Intelligent Development Aid                                    */
/*      - Development Manager                                     */
/*        Set up program for IDA                                  */


aid :-
        [dsp_menu,bld_aux,sql_qry,dialog1, editfld, diccon],
        [uti,begin,bt,input,output,vpf,scrn,menu],
        dic_consult('kb_inp.res'),
        dic_consult('kb_menu.res'),
        main.
```

```
/* Intelligent Development Aid                              */
/*     - DB Developer                                       */
/*       Program to call OSD Generarors                     */


main :-
        def_input,
        def_output,
        def_vpf,
        def_screen,
        def_menu.
```

```
/* Intelligent Development Aid                                     */
/*      - DB Designer                                              */


def_input :-
        introd1,
        create_bt(Handle),
        repeat,
        inp_obj(Obj, Field_list),
        name_only(Field_list, [], Name_list),
        find_key(Obj, Name_list, Key_list),
        find_dep(Obj, Name_list, Dep_list),
        normalization(Obj, Field_list, Name_list,
              Key_list, Dep_list, Vpf_list),
        name_vpf(Obj, Vpf_list, Field_list),
        build_fd(Field_list),
        build_bt(Obj),
        ask.

inp_obj(Obj, Field_list) :-
        clear(obj_active),
        dialog_run((0, 0),(24, 79), clear, single,
         $Specify Form Format$, 1, obj_static, obj_active, X),
        dialog_val(obj_active, name, Obj),
        get_value(obj_active, Field),
        kick(Field, Field1),
        clear_null(Field1, [], Field_list).

find_key(Obj, Name_list, Key_list) :-
        introd3,
        key_groups(Obj, Name_list, [], Key_list).

find_dep(Obj, Name_list, Dep_list) :-
        introd4,
        dep_groups(Obj, Name_list, [], Dep_list).


normalization(Obj, Field_list, Name_list, Key_list,
              Dep_list, Vpf_list) :-
        nf2(Obj, Field_list, Name_list,
            Key_list, Dep_list, Vpf_list1),
        nf3(Obj, Field_list, Vpf_list1,
            Dep_list, Vpf_list).

nf2(Obj, Field_list, Name_list,
    Key_list, Dep_list, Vpf_list1) :-
        find_part_key([Name_list],
                      Key_list, Dep_list, Vpf_list2),
        gar_col(Vpf_list2,Vpf_list1).

find_part_key(Name_list, [], Dep_list, Name_list) :- !.

find_part_key(Name_list, [Key|Key_list],
              Dep_list, Vpf_list1) :-
       check_det(Name_list,Key,Dep_list,Name_list2,Dep_list),
        find_part_key(Name_list2,Key_list,Dep_list, Vpf_list1).

check_det(Name_list,Key,[],Name_list, Whole_dep_list) :- !.

check_det(Name_list,Key,[[Det,Dep]|Dep_list],Name_list1,
          Whole_dep_list) :-
        ifthenelse(part_of(Det,Key), build_vpf(
          Det,Dep,Name_list,Name_list,Name_list2,
          Whole_dep_list),fit(Name_list2,Name_list)),
        check_det(Name_list2,Key,Dep_list,
                  Name_list1,Whole_dep_list).

[illegible], Field_list, Name_list, Dep_list, Vpf_list1) :-
        find_tran_dep(Name_list, Dep_list,
                      Dep_list, Vpf_list2),
        gar_col(Vpf_list2,Vpf_list1).

find_tran_dep(Name_list, [], Dep_list, Name_list) :- !.

find_tran_dep(Name_list, [[Det|Dep]|Dep_list1],
              Dep_list, Vpf_list1) :-
        check_dep_l(Name_list,Dep,Dep_list,
                    Name_list2,Dep_list),
        find_tran_dep(Name_list2,Dep_list1,
                      Dep_list, Vpf_list1).

check_dep_l(Name_list,[[]], Dep_list,
            Name_list, Whole_dep_list) :- !.

check_dep_l(Name_list,[[Dep|Dep_list1]],
            Dep_list, Name_list3, Whole_dep_list) :-
        check_dep(Name_list,[Dep],Dep_list,
                  Name_list2,Whole_dep_list), !,
        check_dep_l(Name_list2, [Dep_list1],
                    Dep_list, Name_list3,Whole_dep_list).

check_dep(Name_list,Dep,[],Name_list,Whole_dep_list) :- !.

check_dep(Name_list,Dep1,[[Det,Dep]|Dep_list],
          Name_list1,Whole_dep_list) :-
        ifthenelse(same_as(Dep1,Det), build_vpf(
                  Det,Dep,Name_list,Name_list,
                  Name_list2,Whole_dep_list),
                  fit(Name_list2,Name_list)),
        check_dep(Name_list2,Dep1,Dep_list,
                  Name_list1,Whole_dep_list).

build_vpf(Det,Dep,[], Name_list,Name_list,
          Whole_dep_list) :-
        write_center([$error, error$]),!.

build_vpf(Det,Dep,[N1|Name_list],Name_list2,
          Name_list1,Whole_dep_list) :-
        part_of(Dep,N1),
        reverse(Det, Det1),
        reverse(Dep, Dep1),
        append(Det1,Dep1,New_list1),
        split(Dep, N1, New_list2),
        fit([New_list1, New_list2|Name_list], Name_list1), !.

build_vpf(Det,Dep,[N1|Name_list], Name_list2,
      [N1|Name_list1],Whole_dep_list) :-
        build_vpf(Det,Dep,Name_list, Name_list2,
                  Name_list1, Whole_dep_list).

split([], List3, List3).

split([A|List1],List2,List3) :-
        d_det(A,List2,New_list2),
        split(List1,New_list2, List3).

gar_col([],[]) :- !.

gar_col([[A,B|F_list]|List],[[A,B|F_list]|List1]) :-
        gar_col(List,List1).

gar_col([[A]|List],List1) :-
        atom(A),
        gar_col(List,List1).

part_of(A,B) :-
        not(same_as(A,B)),
        partof(A,B).

partof([],Key) :- !.

partof([F1|Det],Key) :-
        in(F1,Key), !,
        partof(Det,Key).

in(F1,[]) :- !,fail.

in(F1, [F1|Det1]) :- !.

in(F1,[F2|Det]) :-
        F1 \== F2,
        in(F1,Det).

same_as([],[]) :- !.

same_as([],B) :- B \== [],!,fail.

same_as(A,[]) :- A \== [],!,fail.

same_as([F1|A], B) :-  !,
        d_det(F1,B,B1),
        same_as(A, B1).

d_det(F1, [F1|Det1], Det1) :- !.
d_det(F1,[F2|Det],[F2|Det1]) :-
        F1 \== F2,
        d_det(F1,Det,Det1).

name_vpf(Obj,Vpf_list,Field_list) :-
        ask_name(Obj, Vpf_list, Name_vpf_list),
        recorda(obj,obj(Obj,Name_vpf_list,Field_list),_).

ask_name(Obj,[],[]).

ask_name(Obj,[Vpf|Vpf_list],[[Name,Vpf]|Name_vpf_list]) :-
        cls,
        tmove(10,10),
        write('Please Name The File with Fields '),
        write(Vpf), nl,
        tmove(12,15),
        read_line(0,Name_string),
        atom_string(Name, Name_string),
        ask_name(Obj,Vpf_list,Name_vpf_list).


dep_groups(Obj, Name_list, [ [] | Work], Work) :- !.

dep_groups(Obj, Name_list, Work, Key_list) :-
        get_fu_dep(Name_list, Group),
        dep_groups(Obj, Name_list, [Group | Work], Key_list).


find_p_key(Obj, Name_list, Key_list) :-
        get_p_group(Name_list, Key_list).

key_groups(Obj, Name_list, [ [] | Work], Work) :- !.

key_groups(Obj, Name_list, Work, Key_list) :-
        get_group(Name_list, Group),
        key_groups(Obj, Name_list, [Group | Work], Key_list).

get_fu_dep(Name_list, Group) :-
        clear_choice(func_active),
        set_value_2(func_active, Name_list),
        dialog_run((0, 0),(24, 79), clear, single,
                   $Functional Dependency$, 1,
                   func_static, func_active, X),
        get_check_2(func_active, Group).

get_group(Name_list, Group) :-
        clear_choice(key_active),
        set_value(key_active, Name_list),
        dialog_run((0, 0),(24, 79), clear, single,
                   $Candidate Key$, 1,
                   key_static, key_active, X),
        get_check(key_active, Group).

get_p_group(Name_list, Group) :-
        clear_choice(p_active),
        set_value(p_active, Name_list),
        dialog_run((0, 0),(24, 79), clear,
                   single, $Primary Key$, 1,
                   p_static, p_active, X),
        get_check(p_active, Group).

clear_null([],Work_list,End_list) :-
        reverse_list(Work_list, [], End_list).

clear_null([A,B|List],Work_list,End_list) :- A == $$,
        reverse_list(Work_list, [], End_list).

clear_null([A,B|List],Work_list,End_list) :- A \== $$,
        clear_null(List,[B,A|Work_list], End_list).

clear(Key) :-
        recorded(Key, efield(A,B,C,D,E,F), Ref),
        replace(Ref, efield(A,B,C,D,$$,F)),
        fail.

clear(Key) :- !.

clear_choice(Key) :-
        recorded(Key, choice(A,B,C,D,E,F), Ref),
        replace(Ref, choice(A,B,C,$$,greyed, F)),
        fail.

clear_choice(Key) :- !.

find_key_rep(Obj, Field_list, Name_list, Name_list1) :-
        asserta(name_list(Name_list)) , fail.

find_key_rep(Obj, Field_list, Name_list, Name_list1) :-
        find_y(Field_list, Name, Lengh, Times),
        name_list(Name_list3),
        remove_r(Name, Name_list3, [], Name_list2),
        retract(name_list(Name_list3)),
        asserta(name_list(Name_list2)),
        concat(
          [$Most Often Used as a Key to Retrive The Field '$,
                Name, $'$], String),
        write_center(
            [$In The Following Screen Please $,
             $Specified the Field or Fields That$,
                       String]),
        get_p_group(Name_list2, Group),
        append(Group, Name, Vf),
        asserta(vf(Obj,Vf)),
        fail.

find_key_rep(Obj, Field_list, Name_list, Name_list1) :-
        name_list(Name_list1),
        retract(name_list(Name_list1)),!.
remove_r(Name, [], Work, Name_list2) :-
        reverse_list(Work, [], Name_list2).
```

```
remove_r(Name, [Name|Name_list3], Work, Name_list2) :-
        remove_r(Name, Name_list3, Work, Name_list2).

remove_r(Name, [A|Name_list3], Work, Name_list2) :-
        Name \== A,
        remove_r(Name, Name_list3, [A|Work], Name_list2).


get_check(Key, Group) :- retract(key_group(_)), fail.

get_check(Key, Group) :-
        asserta(key_group([])),
        recorded(Key, choice(A,B,C,D,checked,F), _),
        key_group(List),
        retract(key_group(List)),
        asserta(key_group([D|List])),
        fail.

get_check(Key, Group) :-
        key_group(Group), !.

get_check_2(Key, Group) :- retract(key_group(_)), fail.

get_check_2(Key, Group) :-
        asserta(key_group([])),
        recorded(Key, choice(A,B,C,D,E,F), _),
        ifthen(A == 'sone', store),
        E == 'checked',
        key_group(List),
        retract(key_group(List)),
        asserta(key_group([D|List])),
        fail.

get_check_2(Key, Group) :-
        key_group(Group1),
        det_group(Group2),
        match([Group2,Group1], Group), !.

match([[],A], []).

match([A,[]], []).

match(A, A).
store :-
        key_group(List),
        retract(key_group(List)),
        asserta(key_group([])),
        asserta(det_group(List)), !.

store_list(Value) :-
        list1(List),
        retract(list1(List)),
        asserta(list1(Value)),!.
```

```
name_only([], Work, Name_list) :-
        reverse_list(Work, [], Name_list).

name_only([Name,_|Field_list], Work, Name_list) :-
        name_only(Field_list, [Name|Work], Name_list).

get_value(Key, End_list) :-
        retract(value_list(_)),
        asserta(value_list([])),
        parm_list(Key, End_list).

value_list([]).

parm_list(A_key, End_list) :-
        dialog_val(A_key, F, Value),
        value_list(List),
        retract(value_list(List)),
        asserta(value_list([Value | List])),
        fail.

parm_list(A_key, End_list) :-
        value_list(List),
        reverse_list(List, [], End_list),!.

kick([A | Rec_list], Rec_list).

set_value_2(Active, Value_list) :-
        asserta(list1(Value_list)),
        recorded(Active,choice(A,B,C,D,V,E),Ref),
        ifthen(A == 'sone', store_list(Value_list)),
        list1([Value | Value_list1]),
        retract(list1([Value|Value_list1])),
        asserta(list1(Value_list1)),
        replace(Ref, choice(A,B,C,Value,unchecked,E)),
        fail.

set_value_2(Active, List) :- retract(list1(_)), !.

set_value(Active, Value_list) :-
        asserta(list1(Value_list)),
        recorded(Active,choice(A,B,C,D,V,E),Ref),
        list1([Value | Value_list1]),
        retract(list1(_)),
        asserta(list1(Value_list1)),
        replace(Ref, choice(A,B,C,Value,unchecked,E)),
        fail.

set_value(Active, List) :- retract(list1(_)), !.

add_list(Work_value1, Value_list, Work, End_list) :-
        add_1(Value_list,Work_value2),
        append(Value_list,Work_value2, End_list).

add_1(Value_list,Work) :-
        count_mem(Value_list, 0, No),
        No1 is 16 - No,
        add_2(Value_list, No1, Work1).

add_2(Work, 0, Work).

add_2(Work_list, No, Work) :-
        append([$S], Work_list, Work1),
        No1 is No - 1,
        add_2(Work1, No1, Work).
reverse_list([],Y,Y) :- !.

reverse_list([X|Tail],Y,Z) :-
        reverse_list(Tail,[X|Y],Z).

count_mem([], No, No) :- !.

count_mem([A|List], Count, No) :-
        Count1 is Count + 1,
        count_mem(List, Count1, No).

build_fd(Field_list) :-
        not(field_def(A)),
        asserta(field_def([])), fail.

build_fd(Field_list) :-
        field_def(F_list1),
        retract(field_def(F_list1)),
        append(Field_list, F_list1, New_list),
        asserta(field_def(New_list)).

close_all(19) :- !.

close_all(A) :-
        close(A),
        A1 is A + 1,
        close_all(A1).

introd1 :- introd($DEFINE FORM FORMAT$,
  [$   In the next screen, you will have to specify the$,
  $ field names and their maximum lengh for the form$,
  $ you want to store in this data base. After you finished$,
  $ specify this form, hit enter$,
  $ and this step will$,
  $ be terminated.   $]).

introd3 :- introd($SPECIFY UNIQUE KEY(S) IN THIS FORM$,
  [$   In the next screens, you will have to specify the$,
  $ unique key(s) for the forms you just specified.$,
  $ A unique key is a field or fields that can be used to$,
  $ identify every record in your file for this form.$,
  $ Specify these keys one by one. Hit enter$,
  $ after you finish specified one of them. If you don't$,
  $ have any more, just hit enter again and this step will$,
  $ be terminated.   $]).
```

```
introd4 :- introd($DEFINE DEPENDENCY$,
  [$   In the next screens, you will have to specify the$,
  $ dependencies in this form.$,
  $ A dependency is a field or fields (determinant)$,
  $ that is not a unique key$,
  $ but by identify its or their value(s) the value(s) of$,
  $ same other field or fields (dependant) can be determined.$,
  $ Hit ENTER after you finish specified one of them. If you$,
  $ Don't have any more, just hit enter again and this step$,
  $ will be terminated.   $]).
```

```
/* Intelligent Development Aid                                    */
/*      - Report Specification Generator                          */


bld_vpf :- restore, [idb_vpf], start, def_vpf.

def_output :-
        create(H, 'rpt.osd'),
        close(H),
        def_report.

def_report :-
        introd5,
        cls,
        set_up_qrys,
        repeat,
        gc(full),
        cls,
        tmove(0,2),
        write('SELECT FROM '),
        tables(Y),
        rest1(Y),
        done.

rest1([]) :- !.
rest1(Y) :-
        columns(Y,L,Z,X),
        crest1(Y,L,Z,X), !.

crest1(Y,_,_,[]) :-
        !,
        tmove(5,0),
        write_it([],Y,[],[],[]).
crest1(Y,L,Z,X) :-
        where1(L,W),
        group(Z,G,G1),
        order(Z,O,G1),
        tmove(5,0),
        write_it(X,Y,W,G,O),
        cls.
/*      exec_it(X,Y,W,G,O). */

write_it(A,B,C,D,E) :-
        name_rpt(Rpt_name,Heading),
        write_def(Rpt_name,Heading,A,B,C,D,E),
        !.

name_rpt(Rpt_name,Heading) :-
        tmove(10,10),
        write(
     'What is the name of the report you just defined : '),
        nl,
        tmove(12,15),
        read_line(0,Rpt_name),
        tmove(15,10),
        write(
     'Key in the heading for the report you just defined : '),
        nl,
        tmove(17,15),
        read_line(0,Heading).

write_def(Rpt_name, Heading, A,B,C,D,E) :-
        open(Handle, 'rpt.osd', a),
        write(Handle,begin_def(Rpt_name)),
        write(Handle,'.'),
        nl(Handle),
        build_sub_pos(A, Sub_list, Pos_list),
        write_report(Handle,A,B,C,D,E,Heading,
                     Sub_list,Pos_list),
        write(Handle,'.'),
        nl(Handle), nl(Handle),
        write(Handle,end_def(Rpt_name)),
        write(Handle,'.'),
        nl(Handle),
        nl(Handle),
        nl(Handle),
        close(Handle).

build_sub_pos([], [], []) :-
        retract(pos(A)),!.

build_sub_pos([[Table|Col]|Field_list],
              [Head|Sub_list], [Hold_pos|Pos_list]) :-
        cls,
        tmove(10,10),
        write('Key in the heading for '),
        write(Col),
        write(' from file - '),
        write(Table),
        tmove(12,15),
        read_line(0,Head),
        tmove(15,10),
        write('Key in the width of field '),
        write(Col),
        write(' in this report '),
        tmove(17,15),
        read_line(0,Len1),
        int_text(Len,Len1),
        ifthenelse(pos(Hold_pos), retract(pos(Hold_pos)),
                  Hold_pos is 5),
        Pos is Hold_pos + Len + 1,
        asserta(pos(Pos)),
        build_sub_pos(Field_list, Sub_list, Pos_list).


write_report(Handle,A,B,C,D,E,Heading,Sub_list,Pos_list) :-
        nl(Handle),
        write(Handle,'report('),
        nl(Handle),
        write(Handle,'$ select '),
        write_field_list(Handle,A),
        write(Handle,'  from '),
        write_atom_list(Handle,B),
        ifthen(C \== [], write(Handle,'  where ')),
        write_exp_list(Handle,C),
        ifthen(D \== [], write(Handle,'  group by ')),
        write_field_list(Handle,D),
        ifthen(E \== [], write(Handle,'  order by ')),
        write_field_list(Handle,E),
        write(Handle,'; $ ,'),
        nl(Handle),
        write_atom(Handle,Heading),
        write(Handle,','),
        nl(Handle),
        write(Handle,'['),
        write_a_list1(Handle,Sub_list,''),
        write(Handle,'],'),
        write(Handle,Pos_list),
        write(Handle,')').

introd5 :- introd($DEFINE REPORT OBJECTS$,
        [$   In the next process, you will have to specify$,
        $ the format of reports. This prototype provides a$,
        $ menu-driven SQL SELECT interface for you.    $,
        $ GOOD LUCK.   $]).
```

```
/* Intelligent Development Aid                                              */
/*      - Base-table specification generator                                */


build_bt(Obj) :-
        open(Handle,'bt.osd',a),
        recorded(obj,obj(Obj,Name_list,Field_list),Ref),
        write_bt_def(Handle,name_list,Field_list),
        close(Handle).

write_bt_def(Handle,[],Field_list).

write_bt_def(Handle,[[Name,F_list]¦Name_list],Field_list) :-
        nl(Handle),
        write(Handle,'create table '),
        write(Handle,Name),
        write(Handle,' ('),
        write_bt_field(Handle,F_list, Field_list, ''),
        nl(Handle),
        write(Handle, '        );'),
        input_test_data(Name, F_list, Field_list),
        write_bt_def(Handle,Name_list, Field_list).

write_bt_field(Handle,[], Field_list, Comm) :- !.

write_bt_field(Handle,[F¦F_list], Field_list, Comm) :-
        write_field_def(Handle, F, Field_list, Comm),
        fit(',',Comm1),
        write_bt_field(Handle,F_list, Field_list, Comm1).

write_field_def(Handle,F1,[F1,Len1¦Field_list], Comm) :-
        write(Handle,Comm),
        nl(Handle),
        write(Handle,'     '),
        atom_string(F,F1),
        atom_string(Len,Len1),
        write(Handle,F),
        write(Handle,'   char('),
        write(Handle,Len),
        write(Handle,')').


write_field_def(Handle,F,[A,B¦Field_list], Comm) :-
        F \== A,
        write_field_def(Handle,F,Field_list, Comm).

input_test_data(Name, F_list, Field_list) :-
        cls,
        tmove(10,10),
        write('Do you want to input some test data for '),
        write(Name),
        write(' ? '),
        tmove(12,15),
        get(Ans),
        ifthen(Ans == `y,
                 data_spec(Name, F_list, Field_list)).

data_spec(Name, F_list, Field_list) :-
        open(Handle, 'data.tst', a),
        nl(Handle),
        data_loop(Handle, Name, F_list, Field_list),
        close(Handle).

data_loop(Handle, Name, F_list, Field_list) :-
        nl(Handle),
        write(Handle, 'insert into '),
        write(Handle, Name),
        write(Handle, ' values ('),
        cls,
        get_data_val(Handle, F_list, Field_list, ''),
        write(Handle, ');'),
        next_data(Handle, Name, F_list, Field_list).

get_data_val(Handle, [], Field_list, Comm) :- !.

get_data_val(Handle, [Field|F_list], Field_list, Comm) :-
        write(Field),
        write(' : '),
        write(Handle, Comm),
        read_line(0,Value),
        write_atom(Handle, Value),
        fit(',', Comm1),
        get_data_val(Handle, F_list, Field_list, Comm1).

next_data(Handle, Name, F_list, Field_list) :-
        cls,
        tmove(10,10),
        write(
           'Do you want to input another test data record ?'),
        tmove(12,15),
        get(Ans),
        ifthen(Ans == `y,
               data_loop(Handle, Name, F_list, Field_list)).

ask :-
        cls,
        tmove(10,20),
        write($Enter 'Y' to Define Another Form: $),
        get(Ans),
        determ(Ans).

determ(Ans) :- Ans \== 121, Ans \== 89.

create_bt(Handle) :-
        create(H1,'data.tst'),
        close(H1),
        create(H2,'bt.osd'),
        close(H2).
```

```
/* Intelligent Development Aid                                  */
/*     - View Processing Facility Specification Generator       */


bld_scrn :- restore, [idb_scrn], start, def_screen.

def_vpf :-  create(H,'vpf.osd'), close(H),
            retract(hold_table(A)), fail.

def_vpf :-
        introd6,
        open(Handle, 'vpf.osd', a),
        asserta(hold_table('')),
        def_vpf_l(Handle), !.

def_vpf_l(Handle) :-
        column_table(Col,Table,Pos,Type,_),
        retract(hold_table(Hold_table)),
        asserta(hold_table(Table)),
        ifthenelse(Hold_table == Table,
                   conti(Handle,Col,', '),
                   end_vpf(Handle, Hold_table, Table, Col)),
        fail.

def_vpf_l(Handle) :-
        hold_table(Table),
        end_vpf_def(Handle,Table),
        retract(hold_table(Hold_table)),
        close(Handle), !.

end_vpf(Handle, Hold_table, Table, Col) :-
        ifthen(Hold_table \== '',
              end_vpf_def(Handle, Hold_table)),
        begin_vpf_def(Handle, Table),
        conti(Handle,Col,''), !.

begin_vpf_def(Handle,Table) :-
        atom_string(Table,Table_string),
        assert_st,
        nl(Handle),
        make_info(Table, Vpf_name),
        write(Handle, begin_def(Vpf_name)),
        write(Handle,'.'),
        nl(Handle),
        write(Handle, 'vpf('),
        fit(Ins_s_f, $[$$insert into $),
        concat(Ins_s_f, Table_string, Ins_s_f1),
        concat(Ins_s_f1, $ ($, Ins_s_f2),
        retract(ins_s_f(_)),
        asserta(ins_s_f(Ins_s_f2)),
        fit(Ins_s_v, $ values ( $),
        retract(ins_s_v(_)),
        asserta(ins_s_v(Ins_s_v)),
        fit(Del_s, $[$$delete from $),
        concat(Del_s, Table_string, Del_s1),
        concat(Del_s1, $ where $, Del_s2),
        retract(del_s(_)),
        asserta(del_s(Del_s2)),
        fit(Bt_s,$ ['$),
        concat(Bt_s, Table_string, Bt_s1),
        concat(Bt_s1, $' $, Bt_s2),
        retract(bt_s(_)),
        asserta(bt_s(Bt_s2)), !.

conti(Handle, Col, Comm) :-
        retract(ins_s_f(Ins_s_f)),
        concat(Ins_s_f,Comm,Ins_s_f1),
        concat(Ins_s_f1,Col,Ins_s_f2),
        asserta(ins_s_f(Ins_s_f2)),
        retract(ins_s_v(Ins_s_v)),
        concat(Ins_s_v,Comm,Ins_s_v1),
        concat(Ins_s_v1,$ '@^' $ ,Ins_s_v2),
        asserta(ins_s_v(Ins_s_v2)),
        retract(del_s(Del_s)),
        ifthenelse(Comm \== '', fit(Comm1,' and '),
                   fit(Comm1,'')),
        concat(Del_s,Comm1,Del_s1),
        concat(Del_s1,Col ,Del_s2),
        concat(Del_s2,$ = '@^' $,Del_s3),
        asserta(del_s(Del_s3)), !.

end_vpf_def(Handle,Table) :-
        nl(Handle),
        retract(ins_s_f(Ins_s_f)),
        concat(Ins_s_f,$) $, Ins_s_f1),
        retract(ins_s_v(Ins_s_v)),
        concat(Ins_s_v,$) ; $$ ],$, Ins_s_v1),
        write(Handle,Ins_s_f1),
        write(Handle,Ins_s_v1),
        nl(Handle),
        retract(del_s(Del_s)),
        concat(Del_s,$ ; $$], $, Del_s1),
        retract(bt_s(Bt_s)),
        concat(Bt_s,$ , $, Bt_s1),
        table(Table,Cols,_,_),
        int_text(Cols,Col_s),
        concat(Bt_s1,Col_s, Bt_s2),
        concat(Bt_s2,$ ]). $, Bt_s3),
        write(Handle,Del_s1),
        nl(Handle),
        write(Handle,Bt_s3),
        nl(Handle),
        make_info(Table,Vpf),
        write(Handle, end_def(Vpf)),
        write(Handle,'.'),
        nl(Handle), !.

assert_st :-
        asserta(ins_s_f($$)),
        asserta(ins_s_v($$)),
        asserta(del_s($$)),
        asserta(bt_s($$)).

make_info(Table,Vpf) :-
        atom_string(Table, Table_s),
        concat(Table_s,$_info$, Vpf).

introd6 :- introd($DEFINE UPDATE OBJECTS$,
  [$   In the next process, you will have to specify the$,
  $ objects for the update processes. This prototype$,
  $ will generate all the posible processes for you.$,
  $ You can select or delete them later when you specify$,
  $ the menu interface.   $]).
```

```
/* Intelligent Development Aid                                       */
/*      - Screen Specification Generator                             */


bld_menu :- restore, [idb_menu], start, def_menu.

def_screen :-
        write_center([$define Input Screens for$,
                     $the Update Processes$]),
        create(H,'scrn.osd'),
        close(H),
        open(Handle, 'scrn.osd',a),
        def_scrn(Handle),
        close(Handle).

def_scrn(Handle) :-
        table(Table,Cols,_,_),
        def_table(Handle,Table),
        fail.

def_scrn(Handle) :- !.

def_table(Handle,Table) :-
        % def. static fields and start active field
        begin_efield(Handle,Table),
        def_efields(Handle,Table),
        end_efield(Handle, Table),
        begin_text(Handle,Table),
        def_texts(Handle,Table),
        end_text(Handle, Table),
        begin_operation(Handle,Table),
        def_operations(Handle,Table),
        end_operation(Handle, Table).

begin_efield(Handle,Table) :-
        nl(Handle),
        make_info(Table,Key),
        concat(Key,$_active$,Key1),
        write(Handle, begin_def(Key1)),
        write(Handle,'.').

def_efields(Handle, Table) :-
        column_table(Col,Table,Pos,string(Len),_),
        nl(Handle),
        write(Handle, 'efield('),
        int_text(Pos,Pos_s),
        concat($f_$, Pos_s, Fid),
        write(Handle,Fid),
        write(Handle,', '),
        Row is Pos * 2 + 4,
        write(Handle,Row),
        write(Handle,', '),
        write(Handle,30),
        write(Handle,', '),
        write(Handle,Len),
        write(Handle,', '),
        write(Handle,'$$'),
        write(Handle,', _).'),
        fail.

def_efields(Handle, Table) :- !.

end_efield(Handle,Table) :-
        nl(Handle),
        make_info(Table,Key),
        concat(Key,$_active$,Key1),
        write(Handle, end_def(Key1)),
        write(Handle,'.'),
        nl(Handle).

begin_text(Handle,Table) :-
        nl(Handle),
        make_info(Table,Key),
        concat(Key,$_static$,Key1),
        write(Handle, begin_def(Key1)),
        write(Handle,'.'),
        nl(Handle),
        write(Handle, 'text(2,20,'),
        write(Handle, '$$).').

def_texts(Handle, Table) :-
        column_table(Col,Table,Pos,string(Len),_),
        nl(Handle),
        write(Handle, 'text('),
        Row is Pos * 2 + 4,
        write(Handle,Row),
        write(Handle,', '),
        write(Handle,5),
        write(Handle,', '),
        key_in_desc(Table,Col,Desc),
        write_string(Handle,Desc),
        write(Handle,' ).'),
        fail.

def_texts(Handle, Table) :- !.

end_text(Handle,Table) :-
        nl(Handle),
        make_info(Table,Key),
        concat(Key,$_static$,Key1),
        write(Handle, end_def(Key1)),
        write(Handle,'.'),
        nl(Handle).

begin_operation(Handle,Table) :-
        nl(Handle),
        make_info(Table,Key),
        write(Handle, begin_def(Key)),
        write(Handle,'.').
```

```
def_operations(Handle, Table) :-
        key_in_multi_heading(Handle).

key_in_multi_heading(Handle) :-
        input_head_scrn(Handle,'add'),
        input_head_scrn(Handle,'del'),
        input_head_scrn(Handle,'mod').

input_head_scrn(Handle,Oper) :-
        nl(Handle),
        write(Handle,'operation('),
        write_atom(Handle,Oper),
        write(Handle,', '),
        key_in_headings(Oper,Head),
        write_string(Handle,Head),
        write(Handle,' ).').

end_operation(Handle,Table) :-
        nl(Handle),
        make_info(Table,Key),
        write(Handle, end_def(Key)),
        write(Handle,'.'),
        nl(Handle).

key_in_desc(Table,Col,Desc) :-
        cls,
        tmove(10,3),
        write('Key in the description for '),
        write(Col),
        write(' from file - '),
        write(Table),
        write(' for the input screen'),
        tmove(12,15),
        read_line(0,Desc).

key_in_headings(Oper,Head) :-
        tmove(17, 15),
        write('                              '),
        tmove(15,3),
        write('Key in the heading for '),
        write_atom(0,Oper),
        write(' operation '),
        write(' of the input screen'),
        tmove(17,15),
        read_line(0,Head).

make_info(Table, Key) :-
        atom_string(Table,T_s),
        concat(T_s,$_info$,Key).
```

```
/* Intelligent Development Aid                                        */
/*     - Menu Specification Generator                                 */


def_menu :-
        introd7,
        create(H,'menu.osd'),
        close(H),
        open(Handle,'menu.osd',a),
        list_vpf_op(1,End1),
        list_rpt_op(End1,End),
        find_group(Handle,End),
        close(Handle).

list_vpf_op(Beg, End) :-
        ctr_set(0,Beg),
        clear_old('scrn.osd'),
        dic_consult('scrn.osd'),
        recorded('scrn.osd',key_use(Key),R1),
        recorded(Key,operation(Op,Heading),R2),
        build_message1(Op,Key,Msg),
        ctr_inc(0,No),
        fid_build(No,Fid),
        eraseall_run(Fid),
        recordz(Fid,msg_op(Msg,Heading),_),
        fail.

list_vpf_op(Beg,End) :-
        ctr_is(0,End), !.

fid_build(No,Fid) :-
        fit($op_$,Fid1),
        int_text(No,No_s),
        concat(Fid1,No_s,Fid2),
        atom_string(Fid,Fid2), !.

clear_old(File) :-
        recorded(File,key_use(Key),R),
        eraseall(Key),
        fail.

clear_old(File) :- eraseall(File), fail.

clear_old(File) :- !.

eraseall_run(Fid) :-
        eraseall(Fid),!.

eraseall_run(Fid) :- !.

list_rpt_op(No,End) :-
        ctr_set(0,No),
        clear_old('rpt.osd'),
        dic_consult('rpt.osd'),
        recorded('rpt.osd',key_use(Key),R1),
        recorded(Key,report(Sql,Heading,_,_),R2),
        ctr_inc(0,No1),
        fid_build(No1,Fid),
        build_message2(Key,Msg),
        eraseall_run(Fid),
        recordz(Fid,msg_op(Msg,Heading),_),
        fail.

list_rpt_op(No,End) :-
        ctr_is(0,End),
        fid_build(End,Fid),
        eraseall(Fid),
        recordz(Fid,msg_op($send(report(query), query)$,
                'Perform SQL Query'),_),
        !.

build_message1(Op,Key,Msg) :-
        fit($send(update$,Msg1),
        atom_string(Op,Op_s),
        concat(Op_s,$)$, Msg2),
        atom_string(Key,Key_s),
        concat($($,Key_s,Msg4),
        concat(Msg4,$), $,Msg5),
        concat(Msg1,Msg5,Msg6),
        concat(Msg6,Msg2,Msg),!.

build_message2(Key,Msg) :-
        fit($send(report($,Msg1),
        atom_string(Key,Key_s),
        concat(Msg1,Key_s,Msg4),
        concat(Msg4,$), report)$,Msg).

build_message3(Key,Msg) :-
        fit($send(menu($,Msg1),
        concat(Msg1,Key,Msg4),
        concat(Msg4,$), menu)$,Msg).

find_group(Handle,End) :-
        display_group(End),
        read_group,
        build_menu_def(Handle,End,End1,End_flag),
        ifthen(End_flag \== 'y',find_group(Handle, End1)).

display_group(End) :-
        ctr_set(0,0),
        eraseall_run(menu_group_active),
        find_msg_op(1,End,Fid,Heading),
        ctr_inc(0,No),
        Row is No // 3 + 5,
        Col1 is (No mod 3) * 27,
        Col2 is Col1 + 2,
        string_length(Heading,Len),
        ifthenelse(Len > 23,
                substring(Heading,0,23,Heading1),
                fit(Heading,Heading1)),
        recordz(menu_group_active,
                efield(Fid,Row,Col1,4,$$,_),_),
        recordz(menu_group_active,
                efield(Fid,Row,Col2,24,Heading1,_),_),
        fail.

display_group(No) :-
        dialog_run((0, 0),(24, 79), clear, none,
                   $Specify Menu Options$, 1,
                   menu_group_static, menu_group_active, X),
        !.

read_group :-
        asserta(fid('')),
        recorded(menu_group_active,
                 efield(Fid,_,_,_,Group,_),Ref),
        retract(fid(Hold_fid)),
        ifthenelse(Fid == Hold_fid,build_group_rec(Fid,Group),
              new_group_rec(Fid,Group)),
        asserta(fid(Fid)),
        fail.

read_group :-
        retract(fid(Fid)),
        !.

build_menu_def(Handle,Fid_beg, Fid_end, Flag) :-
        ctr_set(1,0),
        seek_group(Handle,Fid_beg, 1, Fid_end, Flag),
        retract_all,!.

retract_all :-
        retract(group_rec(_,_,_)),
        fail.

retract_all :- !.

seek_group(Handle, Fid, 20, Fid, Flag) :-
        del_op_d,
        ifthen(not(more_group),
              fit(Flag,'y')).

seek_group(Handle,Fid_beg, Group, Fid_end, Flag) :-
        int_text(Group,G_string),
        count_op(G_string,0,No),
        ifthenelse(No @> 1,
           write_menu_count(Handle,G_string,Fid_beg,New_fid),
           fit(Fid_beg,New_fid)),
        ifthen(No == 1, ctr_inc(1,_)),
        Group1 is Group + 1,
        seek_group(Handle,New_fid,Group1,Fid_end,Flag).

more_group :-
        count_space,
        ctr_is(1,No),
        No @> 1.

count_space :-
        retract(group_rec(Fid,$S,_)),
        ctr_inc(1,_),
        fail.

count_space :- !.

count_op(G_string,No,Tot) :-
        ctr_set(0,No),
        group_rec(Fid,G_string,Heading),
        ctr_inc(0,No1),
        fail.

count_op(G_string,No,Tot) :-
        ctr_is(0,Tot), !.

del_op_d :-
        retract(group_rec(Fid,$d$,Heading)),
        recorded(Fid,msg_op(_,_),Ref),
        erase(Ref),
        fail.

del_op_d :-
        retract(group_rec(Fid,$D$,Heading)),
        recorded(Fid,msg_op(_,_),Ref),
        erase(Ref),
        fail.

del_op_d :- !.

write_menu_count(Handle,G_string,Fid_beg,New_fid) :-
        ctr_inc(1,NO),
        write_menu_def(Handle,G_string,Fid_beg,New_fid).

write_menu_def(Handle,G_string,Fid_beg,New_fid) :-
        cls,
        ctr_set(2,11),
        tmove(10,10),
        write('Please name the menu with options :'),nl,
        group_rec(Fid,G_string,Heading),
        ctr_inc(2,Line),
        tmove(Line,15),
        write(Heading), nl,
        fail.

write_menu_def(Handle,G_string,Fid_beg,New_fid) :-
        ctr_inc(2,_),
        ctr_inc(2,Line),
        tmove(Line,15),
        read_line(0,Menu_name),
        ctr_inc(2,_),
        ctr_inc(2,_),
        ctr_inc(2,Line1),
        tmove(Line1,10),
        write('Please key in the Heading for this menu'),nl,
        ctr_inc(2,_),
        ctr_inc(2,Line2),
        tmove(Line2,15),
        read_line(0,Heading),
        concat(Menu_name,$_active$,Active),
        concat(Menu_name,$_static$,Static),
        write_active(Handle,Active,G_string,Msg_list),
        write_static(Handle,Static,Heading),
        write_option(Handle,Menu_name,Msg_list),
        add_msg_op(Menu_name,Heading,Fid_beg,New_fid).

write_active(Handle,Active,G_string,Msg_list) :-
        write(Handle,begin_def(Active)),
        write(Handle,'.'),
        nl(Handle),
        write_box(Handle),
        ctr_set(0,10),
        asserta(msg_list([])),
        retract(group_rec(Fid,G_string,Heading)),
        recorded(Fid,msg_op(Msg,Op),R1),
        erase(R1),
        write(Handle,'choice('),
        write(Handle,Fid),
        write(Handle,','),
        ctr_inc(0,Row),
        ctr_inc(0,Row1),
        retract(msg_list(List)),
        append(List,[Msg],Msg_list1),
        asserta(msg_list(Msg_list1)),
        write(Handle,Row),
        write(Handle,$, 20, $),
        write_atom(Handle,Heading),
        write(Handle,', unchecked, _).'),
        nl(Handle),
        fail.

write_active(Handle,Active,G_string,Msg_list) :-
        retract(msg_list(Msg_list1)),
        append(Msg_list1,[$send(menu(return), return)$],
                 Msg_list),
        write(Handle,'choice(exit,20,20,'),
        write_atom(Handle,'Exit'),
        write(Handle,', unchecked, _).'),
        nl(Handle),
        write(Handle,'choice_box_end.'),
        nl(Handle),
        write(Handle,end_def(Active)),
        write(Handle,'.'),
        nl(Handle), !.

write_static(Handle,Static,Heading) :-
        write(Handle,begin_def(Static)),
        write(Handle,'.'),
        nl(Handle),
        write(Handle,'text( 4, 30, '),
        write_atom(Handle,Heading),
        write(Handle,').'),
        nl(Handle),
        write(Handle,$text( 7, 20, 'Select Option').$),
        nl(Handle),
        write(Handle,end_def(static)),
        write(Handle,'.'),
        nl(Handle), !.

write_option(Handle,Menu_name,Msg_list) :-
        write(Handle,begin_def(Menu_name)),
        write(Handle,'.'),
        nl(Handle),
        write_menu_choice(Handle,1,Msg_list),
        write(Handle,end_def(Menu_name)),
        write(Handle,'.'),
        nl(Handle),
        nl(Handle),!.

write_menu_choice(Handle,No,[]) :- !.

write_menu_choice(Handle,No,[Msg|Msg_list]) :-
        write(Handle,'menu_choice('),
        write(Handle,No),
        write(Handle,', '),
        write(Handle,Msg),
        write(Handle,').'),
        nl(Handle),
        No1 is No + 1,
        write_menu_choice(Handle,No1,Msg_list).

build_group_rec(Fid,Heading) :-
        retract(group_rec(Fid,G1)),
        assertz(group_rec(Fid,G1,Heading)).

new_group_rec(Fid,Group) :-
        asserta(group_rec(Fid,Group)).

find_msg_op(No,End,Fid,Heading) :-
        No @=< End,
        fid_build(No,Fid),
        recorded(Fid,msg_op(Msg,Heading),R1).

find_msg_op(No,End,Fid,Heading) :-
        No @=< End,
        No1 is No + 1,
        find_msg_op(No1,End,Fid,Heading).

write_box(Handle) :-
        write(Handle,'choice_box(box,0,0,24,79,none,
              $Menu$,radio,1,_).'),
        nl(Handle).
```

```
add_msg_op(Menu_name,Heading,Fid_beg,New_fid) :-
        New_fid is Fid_beg + 1,
        fid_build(New_fid, Fid),
        build_message3(Menu_name,Msg),
        recordz(Fid,msg_op(Msg,Heading),_).

introd7 :- introd($DEFINE MENU OJECTS$,
  [$   In the next process, you will have to design a$,
  $ menu interface for your data base. This prototype$,
  $ will list all the available input/output processes.$,
  $ You can group them into one menu by put same menu number$,
  $ infront of the processes. You can delete the unnecessary$,
  $ processes by put a 'd' infront of them. The option$,
  $ description can be changed by change the process$,
  $ descriptions   $]).
```

```
/* Intelligent Development Aid                                      */
/*      - Report Revisor                                            */


start :- [+query],
         [dialog1, editfld, diccon],
         sql_consult('bt.osd'),
         sql_consult('data.tst'),
         [uti,output],
         def_output.
```

```
/* Intelligent Development Aid                                          */
/*     -   Screen Revisor                                               */


start :-
          sql_consult('bt.osd'),
          [uti,scrn],
          def_screen.
```

```
/* Intelligent Development Aid                                           */
/*      - Menu Revisor                                                   */


start :-
        [dialog1, editfld, diccon],
        dic_consult('kb_menu.res'),
        [uti,menu],
        def_menu.
```

APPENDIX D

OBJECT SPECIFICATION DICTIONARY

FACULTY SERVICE DATABASE

```
create table course (
     cur_no     char(5),
     cur_name     char(20),
     cr_hrs     char(1),
     lec_hrs     char(1),
     lab_hrs     char(1)
        );
create table service (
     ss#   char(9),
     quarter     char(1),
     serv_perc     char(3),
     instruction     char(3),
     research     char(3)
        );
create table faculty (
     ss#   char(9),
     name   char(20),
     rank   char(20),
     department     char(3)
        );
create table class (
     ss#   char(9),
     quarter     char(1),
     cur_no     char(5),
     under_stu     char(3),
     grad_stu     char(3),
     other_stu     char(3)
        );
```

```
begin_def(teach_rpt).

report(
$ select        faculty.name,class.quarter,class.cur_no,
                    course.cr_hrs,course.cur_name
  from          faculty,class,course
  where         faculty.ss# = class.ss# and
                    class.cur_no = course.cur_no

  order by        class.quarter,faculty.name
; $ ,
'Faculty Teaching Report',
['Faculty','Q.','Cur.','Cr.','Title'],[5,26,30,37,42]).

end_def(teach_rpt).


begin_def(course_rpt).

report(
$ select        class.cur_no,course.cur_name,class.quarter,
                    class.under_stu,class.grad_stu,
                    class.other_stu
  from          class,course
  where         class.cur_no = course.cur_no
  group by      class.quarter,class.cur_no
  order by      class.quarter,class.cur_no
; $ ,
'Courses Report',
['Cur.','Course Title','Q.','Under','Grad.','Other'],
     [5,12,33,37,45,53]).

end_def(course_rpt).


begin_def(class_rpt).

report(
$ select        faculty.name,class.quarter,class.cur_no,
                    class.under_stu,class.grad_stu,class.other_stu
  from          class,faculty
  where         class.ss# = faculty.ss#

  order by      class.quarter,faculty.name,class.cur_no
; $ ,
'Classes Report',
['Faculty','Q.','Cur.','Under','Grad.','Other'],
     [5,26,30,37,45,53]).

end_def(class_rpt).
```

```
begin_def(class_info_active).
efield(f_1, 6, 30, 9, $$, _).
efield(f_2, 8, 30, 1, $$, _).
efield(f_3, 10, 30, 5, $$, _).
efield(f_4, 12, 30, 3, $$, _).
efield(f_5, 14, 30, 3, $$, _).
efield(f_6, 16, 30, 3, $$, _).
end_def(class_info_active).

begin_def(class_info_static).
text(2,20,$$).
text(6, 5, $Soc. Sec. No.$ ).
text(8, 5, $Quarter$ ).
text(10, 5, $Course No$ ).
text(12, 5, $No of Under Grad. Student$ ).
text(14, 5, $No of Graduate Student$ ).
text(16, 5, $No of Other Student$ ).
end_def(class_info_static).

begin_def(class_info).
operation('add', $Add New Class$ ).
operation('del', $Delete Class Information$ ).
operation('mod', $Modify Class Information$ ).
end_def(class_info).

begin_def(course_info_active).
efield(f_1, 6, 30, 5, $$, _).
efield(f_2, 8, 30, 20, $$, _).
efield(f_3, 10, 30, 1, $$, _).
efield(f_4, 12, 30, 1, $$, _).
efield(f_5, 14, 30, 1, $$, _).
end_def(course_info_active).

begin_def(course_info_static).
text(2,20,$$).
text(6, 5, $Course No$ ).
text(8, 5, $Course Title$ ).
text(10, 5, $Cr. Hours$ ).
text(12, 5, $Lecture Hours$ ).
text(14, 5, $Lab Hours$ ).
end_def(course_info_static).

begin_def(course_info).
operation('add', $Add New Course$ ).
operation('del', $Delete Course Information$ ).
operation('mod', $Modify Course Information$ ).
end_def(course_info).

begin_def(service_info_active).
efield(f_1, 6, 30, 9, $$, _).
efield(f_2, 8, 30, 1, $$, _).
efield(f_3, 10, 30, 3, $$, _).
efield(f_4, 12, 30, 3, $$, _).
efield(f_5, 14, 30, 3, $$, _).
```

```
end_def(service_info_active).

begin_def(service_info_static).
text(2,20,$$).
text(6, 5, $Soc. Sec. No.$ ).
text(8, 5, $Quarter$ ).
text(10, 5, $Percent of Full Time$ ).
text(12, 5, $% of Instruction$ ).
text(14, 5, $% of Research$ ).
end_def(service_info_static).

begin_def(service_info).
operation('add', $Add Faculty Service Information$ ).
operation('del', $Delete Service Information$ ).
operation('mod', $Modify Service Information$ ).
end_def(service_info).

begin_def(faculty_info_active).
efield(f_1, 6, 30, 9, $$, _).
efield(f_2, 8, 30, 20, $$, _).
efield(f_3, 10, 30, 20, $$, _).
efield(f_4, 12, 30, 3, $$, _).
end_def(faculty_info_active).

begin_def(faculty_info_static).
text(2,20,$$).
text(6, 5, $Soc. Sec. No.$ ).
text(8, 5, $Name$ ).
text(10, 5, $Rank$ ).
text(12, 5, $Department$ ).
end_def(faculty_info_static).

begin_def(faculty_info).
operation('add', $Add New Faculty Member$ ).
operation('del', $Delete Faculty Member$ ).
operation('mod', $Modify Perfonal Information$ ).
end_def(faculty_info).
```

```
begin_def(class_info).
vpf(
[$insert into class
(ss#, quarter, cur_no, under_stu, grad_stu, other_stu)
values (  '@^' ,  '@^' ,  '@^' ,  '@^' ,  '@^' ,  '@^' )
 ; $ ],
[$delete from class
where ss# = '@^'  and quarter = '@^'  and cur_no = '@^'
and under_stu = '@^'  and grad_stu = '@^'
and other_stu = '@^'
 ; $],
 ['class'  , 6 ]).
end_def(class_info).

begin_def(course_info).
vpf(
[$insert into course
(cur_no, cur_name, cr_hrs, lec_hrs, lab_hrs)
values (  '@^' ,  '@^' ,  '@^' ,  '@^' ,  '@^' )
 ; $ ],
[$delete from course
where cur_no = '@^'  and cur_name = '@^'  and cr_hrs = '@^'
and lec_hrs = '@^'  and lab_hrs = '@^'
 ; $],
 ['course'  , 5 ]).
end_def(course_info).

begin_def(service_info).
vpf(
[$insert into service
(ss#, quarter, serv_perc, instruction, research)
values (  '@^' ,  '@^' ,  '@^' ,  '@^' ,  '@^' )
 ; $ ],
[$delete from service
where ss# = '@^'  and quarter = '@^'  and serv_perc = '@^'
and instruction = '@^'  and research = '@^'
 ; $],
 ['service'  , 5 ]).
end_def(service_info).

begin_def(faculty_info).
vpf(
[$insert into faculty
(ss#, name, rank, department)
values (  '@^' ,  '@^' ,  '@^' ,  '@^' )
 ; $ ],
[$delete from faculty
where ss# = '@^'  and name = '@^'  and rank = '@^'
and department = '@^'
 ; $],
 ['faculty'  , 4 ]).
end_def(faculty_info).
```

```
begin_def(up_class_active).
choice_box(box,0,0,24,79,none,$Menu$,radio,1,_).
choice(op_1,10, 20, 'Add New Class', unchecked, _).
choice(op_2,12, 20, 'Delete Class Info.', unchecked, _).
choice(op_3,14, 20, 'Modify Class Info.', unchecked, _).
choice(exit,20,20,'Exit', unchecked, _).
choice_box_end.
end_def(up_class_active).
begin_def(up_class_static).
text( 4, 30, 'Update Class Info.').
text( 7, 20, 'Select Option').
end_def(up_class_static).
begin_def(up_class).
menu_choice(1, send(update(class_info), add)).
menu_choice(2, send(update(class_info), del)).
menu_choice(3, send(update(class_info), mod)).
menu_choice(4, send(menu(return), return)).
end_def(up_class).

begin_def(up_cur_active).
choice_box(box,0,0,24,79,none,$Menu$,radio,1,_).
choice(op_4,10, 20, 'Add New Course', unchecked, _).
choice(op_5,12, 20, 'Delete Course Info.', unchecked, _).
choice(op_6,14, 20, 'Modify Course Info.', unchecked, _).
choice(exit,20,20,'Exit', unchecked, _).
choice_box_end.
end_def(up_cur_active).
begin_def(up_cur_static).
text( 4, 30, 'Update Course Info.').
text( 7, 20, 'Select Option').
end_def(up_cur_static).
begin_def(up_cur).
menu_choice(1, send(update(course_info), add)).
menu_choice(2, send(update(course_info), del)).
menu_choice(3, send(update(course_info), mod)).
menu_choice(4, send(menu(return), return)).
end_def(up_cur).

begin_def(up_serv_active).
choice_box(box,0,0,24,79,none,$Menu$,radio,1,_).
choice(op_7,10, 20, 'Add Service Info.', unchecked, _).
choice(op_8,12, 20, 'Delete Service Info.', unchecked, _).
choice(op_9,14, 20, 'Modify Service Info.', unchecked, _).
choice(exit,20,20,'Exit', unchecked, _).
choice_box_end.
end_def(up_serv_active).
begin_def(up_serv_static).
text( 4, 30, 'Update Service Information').
text( 7, 20, 'Select Option').
end_def(up_serv_static).
begin_def(up_serv).
menu_choice(1, send(update(service_info), add)).
menu_choice(2, send(update(service_info), del)).
menu_choice(3, send(update(service_info), mod)).
menu_choice(4, send(menu(return), return)).
```

```
end_def(up_serv).

begin_def(up_facu_active).
choice_box(box,0,0,24,79,none,$Menu$,radio,1,_).
choice(op_10,10, 20, 'Add New Faculty Member', unchecked, _).
choice(op_11,12, 20, 'Delete Faculty Member', unchecked, _).
choice(op_12,14, 20, 'Modify Perfonal Informa', unchecked, _).
choice(exit,20,20,'Exit', unchecked, _).
choice_box_end.
end_def(up_facu_active).
begin_def(up_facu_static).
text( 4, 30, 'Update Personal Info.').
text( 7, 20, 'Select Option').
end_def(up_facu_static).
begin_def(up_facu).
menu_choice(1, send(update(faculty_info), add)).
menu_choice(2, send(update(faculty_info), del)).
menu_choice(3, send(update(faculty_info), mod)).
menu_choice(4, send(menu(return), return)).
end_def(up_facu).

begin_def(out_rpt_active).
choice_box(box,0,0,24,79,none,$Menu$,radio,1,_).
choice(op_13,10, 20, 'Faculty Teaching Report', unchecked, _).
choice(op_14,12, 20, 'Courses Report', unchecked, _).
choice(op_15,14, 20, 'Classes Report', unchecked, _).
choice(exit,20,20,'Exit', unchecked, _).
choice_box_end.
end_def(out_rpt_active).
begin_def(out_rpt_static).
text( 4, 30, 'Report Generation').
text( 7, 20, 'Select Option').
end_def(out_rpt_static).
begin_def(out_rpt).
menu_choice(1, send(report(teach_rpt), report)).
menu_choice(2, send(report(course_rpt), report)).
menu_choice(3, send(report(class_rpt), report)).
menu_choice(4, send(menu(return), return)).
end_def(out_rpt).

begin_def(up_menu_active).
choice_box(box,0,0,24,79,none,$Menu$,radio,1,_).
choice(op_17,10, 20, 'Update Class Info.', unchecked, _).
choice(op_18,12, 20, 'Update Course Info.', unchecked, _).
choice(op_19,14, 20, 'Update Service Info.', unchecked, _).
choice(op_20,16, 20, 'Update Personal Info.', unchecked, _).
choice(exit,20,20,'Exit', unchecked, _).
choice_box_end.
end_def(up_menu_active).
begin_def(up_menu_static).
text( 4, 30, 'Data Base Update').
text( 7, 20, 'Select Option').
end_def(up_menu_static).
begin_def(up_menu).
menu_choice(1, send(menu(up_class), menu)).
menu_choice(2, send(menu(up_cur), menu)).
menu_choice(3, send(menu(up_serv), menu)).
menu_choice(4, send(menu(up_facu), menu)).
menu_choice(5, send(menu(return), return)).
end_def(up_menu).

begin_def(outputs_active).
choice_box(box,0,0,24,79,none,$Menu$,radio,1,_).
choice(op_16,10, 20, 'Perform SQL Query', unchecked, _).
choice(op_21,12, 20, 'Report Generation', unchecked, _).
choice(exit,20,20,'Exit', unchecked, _).
choice_box_end.
end_def(outputs_active).
begin_def(outputs_static).
text( 4, 30, 'Output Generation').
text( 7, 20, 'Select Option').
end_def(outputs_static).
begin_def(outputs).
menu_choice(1, send(report(query), query)).
menu_choice(2, send(menu(out_rpt), menu)).
menu_choice(3, send(menu(return), return)).
end_def(outputs).

begin_def(main_active).
choice_box(box,0,0,24,79,none,$Menu$,radio,1,_).
choice(op_22,10, 20, 'Data Base Update', unchecked, _).
choice(op_23,12, 20, 'Output Generation', unchecked, _).
choice(exit,20,20,'Exit', unchecked, _).
choice_box_end.
end_def(main_active).
begin_def(main_static).
text( 4, 30, 'Faculty Service Data Base').
text( 7, 20, 'Select Option').
end_def(main_static).
begin_def(main).
menu_choice(1, send(menu(up_menu), menu)).
menu_choice(2, send(menu(outputs), menu)).
menu_choice(3, send(menu(return), return)).
end_def(main).
```