An Object-Oriented Framework for the Creation of Customized Expert System for CAD

by

Parasuram Narayanan

Thesis Submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Mechanical Engineering

APPROVED:

S. Jayaram

Dr. S. Jayaram, Chairman

[illegible]

Dr. A. Myklebust

[illegible]

Dr. C. F. Reinholtz

December 1993

Blacksburg, Virginia

c.2

LD
5655
V855
1993
N372
c.2

An Object-Oriented Framework for the Creation of
Customized Expert System for CAD

by

Parasuram Narayanan

Committee Chairman: Dr. S. Jayaram

Mechanical Engineering

(ABSTRACT)

With the advancements in the fields of Computer Aided Design (CAD) and Artificial Intelligence (AI), a number of CAD systems that are developed have built-in expert systems to aid the designer in the design process. Recently there has been a trend in industries and research organizations to custom create CAD software to satisfy specific in-house needs. But there are not enough tools available to custom create expert systems to meet the needs of CAD. Expert systems currently have to be developed from basics using languages that are not in everyday use by the CAD programmers. Expert system shells available in the market do not have the flexibility or portability to support the creation of an expert system for multi-disciplinary parametric design. Thus there is a lack of repeatable use software to support the creation of customized expert systems to meet the special needs of parametric deign in CAD.

In this thesis, the design of an object oriented framework which will aid in the creation of customized expert systems for CAD applications is presented. This framework, known as the Expert Consultation Environment, provides the CAD programmer with tools to create the expert system. This framework consists of various object-oriented classes which the programmer could use. The central part of this framework is the Expert Technician (ET) class. This class represents an expert in a real world situation. Each expert created by the programmer would have its own methods and knowledge in a domain of design. The ET would thus assist the designer using the expert system in that particular domain. The ET would be able to interact with the user in several different modes. These modes are the Consultant, the Transactor, the Observer, the Teacher and the Student modes. The method of interaction between the designer and the ET depends on the mode of operation of the ET.

A programmer of the expert system would be able to create these experts by providing knowledge and the design parameters to the ETs. In the case of a multi-disciplinary design the programmer would provide each ET with the knowledge regarding the specific domain of the design. In the case of concurrent engineering, each ET would be provided with knowledge regarding the a domain involved in the product cycle of a component.

The object-oriented design for the framework has been discussed in detail including class descriptions of all the classes in the framework. A prototype of the framework was developed using C++. The creation of an expert system using this prototype for a multi-disciplinary design application is also discussed in this thesis.

Acknowledgments

I would like to thank my wonderful parents A. P. Narayanan and Bhanumathi Narayanan without whose inspiration and support this graduate studies would never have been possible. I would also like to thank my sisters, Uma and Priya for all the encouragement and being there in times of need.

I would like to thank my advisor Dr. S. Jayaram for the support and guidance throughout my graduate studies. I would also like to thank Dr. Arvid Myklebust for the financial support and for agreeing to serve on my committee. I would like to thank Dr. C. F. Reinholtz for serving on this graduate committee.

I would like to thank Venky, Satya, Watson, Dilip, Ravi, Krishna, Srinivas, Suresh, Sai, Saikat and Vinita for making my stay in Blacksburg enjoyable. For taking the monotony away from all the work at the lab, thanks to John, Francisco, David, Andy, Jiewen and Scott. Finally thanks to all the friends from my Roorkee days who kept in touch.

Table of Contents

List of Illustrations

1. Introduction

The field of Computer Aided Design (CAD) has come a long way in the recent years in aiding the designers of engineering systems. The initial CAD systems that were developed and were available in the market mainly aided only in the drafting and drawing and in the visualization of the component or the system being designed. With the advancement in the field of the CAD, current systems are able to aid the designer in various fields of engineering analysis, finite element modeling and analysis, geometric modeling and manufacturing of the components being designed. Moreover, inroads made in the field of computer hardware and software have made current CAD systems faster and more efficient compared to the early CAD systems that were available. Thus, performing engineering design on CAD systems has become attractive to industry in terms of both cost and turnaround time.

Engineering design problems have always been domain specific. Designers performing a design require expert knowledge and information in specific domains of design to carry out the design process. Recent research in the field of Artificial Intelligence(AI) and Expert Systems have attacked this problem of providing the designer with the knowledge and information necessary to carry out design in a particular domain of engineering. Domain-specific knowledge in the form of design methodologies, constraints, equations, facts, design rules and rules of thumb are acquired from the experts in the field and stored in the system in the form of rules. Decisions regarding the design are made by the user or the computer with the help of these rules. Research is also being carried out in the field of machine learning wherein the computer would be able to learn to make decisions by acquiring knowledge from the application being used. With specific reference to engineering design, the computer would be able to capture domain-specific knowledge from the design being carried out and store it as a part of the computer knowledge. Thus, in later designs, the knowledge from the earlier designs would be used to make decisions regarding the design leading to an efficient design and reducing the time required for the design.

With the advancements in the fields of CAD and AI, there have been a number of CAD systems which have integrated expert systems to aid the designer in the design process. Since design process is domain-specific, these systems are created in-house and are tailored to comply with the functional requirements of the design. Since no CAD system can be so versatile as to satisfy the requirements of mechanical design needs of all companies and other organizations, these applications programs are custom-created to satisfy the needs of each domain.

In spite of these advancements in the fields CAD and AI, the high costs brought about by the creation of in-house software cannot be avoided since the requirements for these systems are so diverse. These costs are brought about because the software is developed from scratch and the turnaround time for the development for such software is usually high. Even though this activity of developing custom software to satisfy the specific needs of an organization is usually unavoidable, the cost and time can be greatly reduced by the creation of repeatable-use software. Research in this field is concentrating on the easy creation of specialized software, rather than the creation of large general purpose CAD software. Emergence of tools which aid the programmers in the easy creation of such software has become significant in recent years. Tools for the easy creation of graphics, user interfaces, etc. has greatly reduced the burden on the programmers. Emergence of tools like GKS and PHIGS as ISO standards for graphics have greatly enhanced the capabilities of custom CAD systems by providing full 3-D functionality along with device-independence.

Although the field of CAD has come a long way in recent years, the integration of CAD with engineering/mechanical design, especially the integration of CAD with conceptual design, has not been provided the importance that it deserves. Tools which facilitate the custom development of conceptual CAD software are still not available.

This is especially true in the case of intelligent CAD systems. This thesis describes the philosophy and design of a framework which will facilitate the creation of expert systems to support parametric, conceptual design. A full implementation of the framework would provide CAD programmers with a set of high-level tools for customizing expert systems for multi-disciplinary, conceptual parametric design.

2. Literature Review

In order to meet the specific in-house needs, industries and organizations have in recent years shown a tendency to develop their own custom CAD/CAM software. In a recent survey of 26 Fortune 500 companies, 88.5% of the surveyed companies reported development of custom CAD/CAM software to meet their special requirements [Penn91]. The advancements made in the field of hardware technology have made the computers powerful enough to meet the complex needs of CAD. Kidwell has discussed how workstations can be used to save time during the conceptual design stage of an aircraft, thereby reducing the cost of the aircraft [Kidw87]. CAD systems have come a long way in recent years. Advancements have been made in the fields of design, analysis and visualization. Research in the field of application of artificial intelligence to CAD has received a significant thrust in recent years. This is evident from the availability of publications regarding the research being done in this field. Research in this field is mainly concentrating on the application of knowledge-based and rule-based systems to CAD. This includes the development of expert systems to aid the designers using CAD systems.

CAD and Knowledge Based System (KBS) tools are changing the way mechanical engineering design is being carried out. Ullman and Dietterich have reviewed CAD and KBS systems and have detailed the current design methodology used for mechanical design [Ullm87]. The process of mechanical design right from the initial phase of problem definition until the final solution is discussed. The main thrust areas for CAD-KBS development are listed. These include design methodology where the system guides the user through the design process and design history recording where the various design stages are recorded. The authors also emphasize that the areas that the research should concentrate on are the integration of KBS and CAD tools and configuration of KBS tools for CAD.

This chapter reviews publications regarding research in the application of expert systems to CAD systems. The first section discusses the application of expert systems to engineering design. The second section discusses object-oriented design and programming especially with relation to AI.

Artificial Intelligence and CAD

While a number of conceptual design systems exist, artificial intelligence and expert systems technology have been recently applied to manage complexity and integrate these large design systems. Ohsuga has discussed the necessity for CAD systems to be intelligent and suggestions to make intelligent CAD systems useful [Ohsu87]. Bouchard et al have explored the possibilities of application of artificial intelligence technology to aeronautical system design [Bouc88]. This technology can be applied in automating the geometry for parametric design by the use of solid modeling and surface geometry and the use of object-oriented technology for the definition of geometry. The analysis of design space can also be automated using artificial intelligence techniques for search, performance relations, computational trees and constraint propagation. Optimization is another prime candidate for automation. An automated aircraft system would therefore consist of an advisor for selection of analysis methods, a design space analysis system and an optimization system.

An aircraft design system incorporating quasi-procedural methodology has been developed by Kroo and Takai [Kroo88]. This system consists of various procedural analysis routines. A computational path is generated using the inputs available and the outputs desired. A rule-based system working on a set of if-then rules, serves as a warning system and reports violations of design rules, constraints and analysis routines by examining the database at every stage of design. A knowledge-based system suggests solutions to identified problems by examining the database, rule-base and the warnings posted. The rule-base also advises on selection of analysis procedures. Integration of numerical optimization with the quasi-procedural methods using sequential quadratic programming and genetic algorithms has also been investigated by Kroo [Kroo92].

A system has been developed by Gillam [Gill92] wherein a knowledge-based paradigm is used to represent and use information to assist and guide the design of large space systems. Explicit storing of information makes the browsing of available information possible and makes the system model more credible, transparent and accessible. This also helps in the follow-up of a decision through the conclusion arrived at from the decision, or vice versa. This is done by evaluation of the conclusion and tracing back to the decisions and assumption on which they are based thus evaluating the impact of the decisions.

To facilitate in making conceptual design systems flexible, a constraint-based modeling system has been developed by Kolb [Kolb88, Kolb92] in which object-oriented techniques have been used to define design components. The components are described in terms of their attributes and their constraints and each component is an instance of the component class. To integrate the various components, design links, also defined as objects, relate the attributes of different components which govern common constraints. Constraint propagation methods are used to manage mathematical relationships which govern the geometry and physics of the components. Tong et al [Tong92] have integrated artificial intelligence and numerical optimization for the design and optimization of aerospace systems with the use of object-oriented techniques, expert systems and genetic algorithms.

Shortening of information feedback loops for the purpose of analysis and evaluation of design and presenting the information to the designer has been demonstrated in a system developed by Talukdar et al. [Talu90]. It consists of autonomous programs, called critics, each of which keeps track of a specific area of design. For example, the mechanical strength critic keeps track of the structural integrity aspects and the kinematic critic keeps track of the motion of the assembly and the tolerance between parts. As the designer creates a new design, each critic observes the progress of the design and evaluates the design independently by executing analysis programs where appropriate. If new and useful information is obtained or if flaws are detected in the design, the critics provide feedback to the designer by providing appropriate information.

A generic set of decision rules for the synthesis of new products, processes and manufacturing systems have been developed by Kim and Suh [Kim86]. An expert system architecture, "Axioms" has been developed and has been specialized to a particular domain. Design and Manufacturing advisor, a system with specific production knowledge has been developed using the Axiomatics advisor.

Yerramareddy and Lu have developed a system which refines the large initial design space through a series of multi-objective optimizations [Yerr93]. This aids the CAD systems in arriving at a fully specified design space. This system, known as HIDER, uses statistical techniques and machine learning to arrive at models which are then used for the multi-objective optimization. Thus, this system aids in the decision making process by reducing the time to proceed from the initial specifications to the final design.

Lu and Wilhelm have developed a system for automating tolerance synthesis that would take into account the various features of the part that is being designed [Lu91]. This system, CASCADE-T uses the constraint propagation method to evaluate the various parameters of design. The evaluation of parameters is defined through a network of constraints where the relationships are defined in terms of constraints. Calculating the values of parameters are carried out by propagating values through the constraint network. Subramaniam and Lu have discussed a methodology for the development of a CAD system for the design of components manufactured in small and medium lot sizes [Subr91]. This system uses the concept of simultaneous engineering wherein the design is carried out with the intention of keeping the manufacturing cost low. A knowledge based system is used wherein the knowledge regarding manufacturing is captured and stored in the system. This intelligent computer environment would advise the designers on the problems related to manufacturing during the design stage.

Modeling of the design is important from the point of view of the evaluation of the design obtained from CAD. Tcheng and Lu have described a methodology wherein models are represented as layered models with different levels of abstraction [Tche91]. Evaluation of the models could be done at different layers of abstraction. A system has been developed which uses AI techniques for the development of these layered models. The system uses inductive learning algorithms for the development of new models which would build models from examples.

Object -Oriented Design and Programming

A semantically rich object-oriented analysis method for the migration of existing systems using semantic object modeling approach has been detailed by Graham [Grah93]. This method, called SOMA, uses strongly encapsulated objects combined with expert system-like rules for object-oriented analysis. The various steps for performing analysis of existing systems are described.

Ibrahim and Woyak have described an object-oriented environment for the integration different programming paradigms [Ibra90]. This system, known as EDS/OWL, uses an object-oriented approach to program AI applications using different paradigms. The authors use the word paradigms in this paper to refer to the techniques of a particular method of programming. This system allows integration of various programming methods. New and exploratory paradigms for AI problems could be implemented using this environment.

Franke describes how features of object-oriented languages can facilitate the integration of applications and rule inferencing [Fran90]. Methods to imbed rule inferencing in applications is described in detail. A system known as CAD Inference Engine (CADIE) has been developed for integrating rule-based inferencing capabilities to CAD tools developed using object-oriented languages such as C++.

3. Problem Definition and Research Objectives

The development of custom CAD software involves the time-consuming development of functions such as user interfaces, modeling routines, etc. The future of CAD depends on the creation of tools or repeatable use software which aid in the easy creation of custom CAD software. These tools should be created to relieve the programmer of CAD systems from repetitive tasks. Repeatable use functions would be available as a part of the tool and these functions would be used by the programmer who will not need to build these functions from scratch whenever it is needed. PHIGS is an example of such repeatable-use software which allows the programmer to use the functions available to visualize the geometry of a design. The necessity to build tools for the easy creation of software is important especially in the case of AI-based CAD systems in which the user needs to provide all the functions of an AI system in the CAD environment. That is, AI functions such as the inference engine, search routines and management of the knowledge base have to be built by the programmer whenever there is a necessity to use such functions.

The importance of expert systems in CAD has grown in recent years. This is evident from the abundance of research being done in this field and the explosion of publications available. The process of designing a component or part requires expert knowledge in the particular domain in which the component is being designed. For example, the design of a kinematic linkage would involve knowledge in kinematics, structures, and possibly knowledge about materials, among other topics. A designer who is involved in the design would need knowledge of all the different domains to make decisions regarding the design.

With the help of an AI-based or a knowledge-based system, expert knowledge about the different domains can be incorporated into the system and would be utilized to make decisions regarding the design of a particular component. Expert knowledge in the form of rules, equations, facts, rules of thumb, design methodology, etc. could be acquired from the experts in particular domains and stored in the system as expert knowledge. This knowledge could be used by the computer or the user to make decisions about the design.

Currently, expert systems for CAD applications are developed from basic principles using AI languages such as LISP and Prolog or have to be programmed using a general purpose expert system shell. AI languages have a long learning curve and are not the everyday languages used by engineers who develop CAD systems. A programmer using these languages has to create functions for user interfaces and inferencing into the system being developed. Since these languages are different from the languages the CAD systems are built in, integration of the CAD system and the AI methods would be extremely difficult. General purpose expert system shells have the functions of user interfaces and inferencing built into them. But these shells are normally system-dependent. These are closed systems and do not allow integration with the overall CAD system. An ideal architecture for a CAD-based expert system would be a multiple-domain expert system but these shells do not provide the programmer with the flexibility to separate the expertise of the different experts within the same expert system.

This is especially true in the case of parametric, multi-disciplinary design in which the design process revolves around the manipulation of various parameters of the design.

Each of the parameters might directly belong to a different domain in the sense that the parameter would have the maximum effect in the determination of the final parameters of that particular domain of design. Separating out the different domains and their parameters is a difficult task in a conventional expert system shell or using an AI language.

Expert systems developed for a CAD application rarely emulate a real-world situation. In this situation the designer would have available to him the knowledge of experts in different domains. The designer would be able to delve into the expertise of the experts to arrive at decisions that need to be made regarding the design. Taking the earlier mentioned example for the design of a kinematic linkage, the designer would be able to discuss with a kinematics expert and make use of his knowledge and expertise regarding that particular domain. Similarly, the designer would be able to use the knowledge, data and experience of a strength expert to gain expertise in that domain and, finally, using the above knowledge from both the domains the designer would be able to make decisions regarding the overall design. In this situation it has to be noted that the knowledge and data of different domains are always kept separate and final decisions are arrived at taking into consideration both the domains and the interrelationships between them. Most of the expert systems for design which are developed currently seldom try to keep the knowledge of different domains apart.

The availability of a high-level programming environment will reduce the burden on the programmer by removing the task of writing support software and other functions thereby reducing both the cost and the time for the development for custom CAD software. A collection of high-level CAD programming tools will provide an environment for easier, faster and efficient development of CAD applications and will allow the programmers to concentrate on the development of the overall system and the technology involved.

Research in the Computer Aided Laboratory at VPI has led to the design of a suggested high-level, device-independent programming environment to facilitate the development of custom CAD/CAM software. The environment has been named CADMADE (Computer Aided Design and Manufacturing Applications Development Environment) [Jaya89, Jaya90, Jaya93]. An implementation of this environment should include a library of procedures, class libraries, data structures and other CAD/CAM programming aids, and may be viewed as a high-level language for design and manufacturing applications programmers. This overall environment is divided into a number of sub-environments: User Interface Environment (UIE), PHIGS+ Environment, Design and Modeling Environment (DME), Virtual Manufacturing Environment (VME) and the Expert Consultation Environment (ECE). The central part of this ECE are the Expert Technicians (ET) who are the experts in different disciplines of design and would assist the designer by providing the knowledge and information regarding the design. Research is being carried out on the other environments [LinW93, Woya93, Uhor93a, Uhor93b, Flem92, Flem93]. This thesis involves the design and implementation of the ECE.

There is a growing interest in the industry in areas related to customized expert systems for CAD applications. The focus of this research is to develop a programming environment in which a programmer would be able to create an expert system for such an application.

The objectives of this research were:

1. To define the requirements for an object-oriented framework to support the creation of an expert system especially suited to the requirements of parametric CAD systems.

2. To design a high-level programming environment (framework) to aid in the creation of custom expert systems for CAD.

3. Create the requirements for the Expert Technicians, their operating modes, rule base and the inference engine.

4. Define the user interface methods to be built into the ECE.

5. Define and create a prototype of the framework and the Expert Consultation Environment

4. Object-Oriented Design

The art of programming depends on the breaking down of large problems into smaller components which makes development of software for the solution of the problem simpler. Until recent years programmers used to breakdown large codes into modular programs called subroutines, each of which represents a subset of the task performed by the whole program. This task of breaking down of the large code into smaller components is known as decomposition. Traditionally computer programs have always been "functionally" modular wherein large programs are decomposed into smaller tasks and sub-tasks.

In recent years, this process of decomposition of a large problem has shifted focus from tasks to objects. This approach, known as Object-Oriented Programming, concentrates on the representation of the problem by decomposition into objects. This represents a new method of thinking about the problem closely approximating the way problems are solved in a real-world situation. In this situation, objects are self-contained entities having their own data and methods. The data and methods of an object exist as internal details to the object and are not available to other objects in the program. This philosophy of keeping the data and methods of each object private to that particular object is known as encapsulation. Encapsulation is done so that the rest of the application program is insulated from the internal implementation details of the object. An object that needs to access the internals of another object merely sends a message to that object requesting it to carry out a specific process. The object that receives a message, executes a function that operates on data that is internal to itself.

To further illustrate the concepts of object oriented design and programming let us consider an example of representation of people-movers i.e. all modes of transportation used by people to go from one place to another. This example is illustrated by figure 1. People-movers can be broken down further into Land-based and Non land-based modes of transportation. Land-based transportation could be further classified as tracked and road vehicles. Non Land-based vehicles would be further classified into sea-vehicles or ships and planes.

In this case the representation of the problem has been decomposed using the object-oriented methods. In the rest of this chapter the key concepts of Object-Oriented design are further explained.

Classes and Objects

Classes represent groups of objects that are nearly identical. In other words objects are instances of classes. Classes contain all the bare data and methods that are common to a set of objects that are similar. In the example considered, all models of cars are objects of the class Cars. The class Cars would be contain all the methods and the data for the objects but the objects themselves would have specific values for the data. Consider the

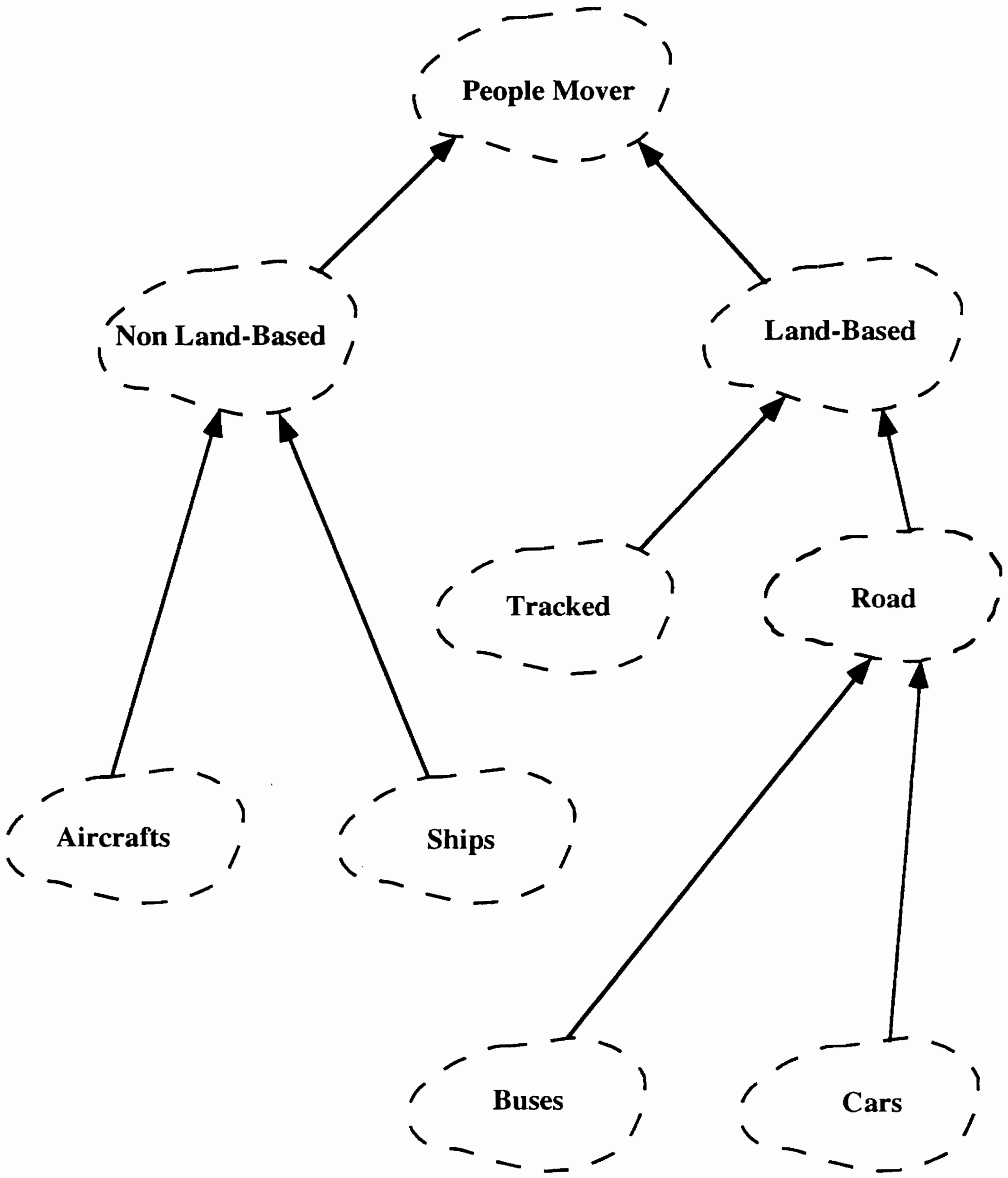


Figure 1 **Representation of a Transportation Problem**

class Cars having a variable representing the horsepower of the engine of the cars. The objects of that class would have a specific value for the variable.

Thus all the common methods and data for a set of objects are collectively represented as a class. When an object needs to be created, message is sent to that particular object to create an instance of itself, thus creating an object. The variables in the class known as class variables have no value attached to them, but the variables in the objects which are instances of the class variables have specific values. The methods of the classes are directly instanced in the objects without any changes.

Two different classes can inherit from another class. For example, Tracked vehicles and Road vehicles are derived from the Land-based vehicles class. The Land-based vehicles class is known as the base class and the Tracked and Road vehicles classes the derived classes. Similarly the People-Movers class is the base class for the Land-based and the Non Land-based vehicles class and the latter the derived classes. The base class contains the data and methods common to all the derived classes and these are inherited by the derived classes. Inheritance is discussed in detail in a following section.

Encapsulation

This is a method in which some of the data and functions of a particular class are hidden from the rest of the application. In effect, the implementation details of the class are insulated from the rest of the program. If an object wants to access the data or the method of another object a message is sent to the latter requesting the value of the data or requesting execution of a function. This is done such that the rest of the application program need not worry about the way data and functions are handled inside the object. For example, in the case of an aircraft, the number of passengers is calculated from the internal size of the length and width of the passenger deck of the aircraft. If the application program needs the number of passengers, it sends a message to the object of the class to calculate the number of passengers it can carry. This function to calculate the number of passenger is hidden inside the object and the rest of the program is insulated from this implementation. If the aircraft is modified to have two passenger decks, the function that determines the number of passengers the aircraft can carry might be changed. Still any other part of the program that requires this information will send the same message as earlier and does not need to know that the internal details of the object have changed. Thus the program can be adapted to future changes giving the programmer ease of maintenance of the code.

Inheritance

Inheritance is a mechanism in which data and methods are shared among different classes. Inheritance allows the data and methods of a super class or a base class to be inherited and used by a derived class or a subclass. As shown in figure 1, in object-oriented design, the problem is represented as a hierarchy of classes and subclasses. As we go down the hierarchy, the classes get more specific. Through the mechanism of inheritance, methods and data common to two or more classes are put in another class which will be the base class for these classes. The derived classes will further contain data and methods that makes them different from the base class.

In the example shown, the base class for the whole representation, the People Mover class, could contain the data common to all the subclasses derived from it. It could contain variables for the number of passengers, the data about the distance it can cover before refueling, the type of fuel used, methods for calculation of the total weight it can carry etc. These data and methods are inherited by the subclasses, Land-Based and the Non Land-Based vehicles. Further the Land-based vehicles class could have data and methods of its own, such as the number of wheels on the vehicle.

A class can inherit from a single class or multiple classes. For example if one wanted a class to represent an amphibious vehicle, it could be built from inheriting from both the Land-Based and the Non Land-Based vehicles class. Inheritance is thus a very powerful mechanism that is not found in the procedural type of programming. It helps in the abstraction of class by allowing the representation of classes as a hierarchy.

Polymorphism

Each object in an object-oriented environment responds to a message which it receives. The response of different objects to the same message might be different depending on the implementation of the class. This mechanism is known as polymorphism. For example, in the objects of all the classes shown if figure 1, calculate_weight might be a function that calculates the overall weight of the vehicle. Even though the message to all the objects to calculate the weight might be the same, the method in which the weight is calculated might be different for each case and is dependent on the internal implementation of the methods in each class. The method common to most of the classes is usually defined and made a part of the base class. Since all the other classes inherit from this class, the objects requiring a different method of calculation usually redefines the function. This coupled with inheritance provides a powerful method to abstract and build classes easily.

Object-oriented design and programming thus provide methods in which the representation of the problem can be more closely approximated to the way in which the objects exist and interact with each other in the actual world. The ideology of development of software shifts from programming tasks to programming objects and their individual behavior. Each of these objects exist as self-contained entities with their own data and methods which dictate the behavior of the objects. The objects interact with each other through messages sent from one object to another. Similar objects can be instantiated from the same class reducing the amount of code to be written for each program or software. The concepts of classes, inheritance, polymorphism, and encapsulation provide several advantages to object oriented programming and design, some of which are listed below.

Reusability: Classes are self-contained entities whose existence is not dependent on other parts of the program. Objects can be created from classes easily by instantiation.

Thus objects can be plugged into the code wherever necessary by creating instances of the class. If classes are well designed they could be used and reused wherever necessary.

Maintainability: Since classes and objects are self-contained, changes made to any of these do not affect the rest of the code. Thus the interaction between objects are not affected whenever the internal implementation details of objects are changed. This makes maintenance of the program an easier and simpler task.

Extendibility: New objects can be built and introduced into the program with the least amount of effort. Declaration of classes as data types provides a powerful mechanism by which new objects can be easily introduced. Development of new classes could also be done by using older classes and making necessary changes to them. Inheritance provides a powerful mechanism by which new classes can be built by inheriting from older classes.

Conceptual Consistency: With careful design of the problem, objects can be made to approximate the way objects exist in the real world. This makes the problems easier to represent, program and understand.

5. Requirements for the Framework

The tools which are created to help in the development of the expert system should relieve the programmer of time consuming activities like building of inference engines, forward and backward chaining techniques, etc. These should be built as a part of the system and the programmer should be able to invoke such functions rather than create one whenever it is necessary. The expert system created from such tools should emulate a real-world situation or the design session from such an expert system should closely approximate the way experts are consulted by a designer/engineer to design a particular component or system. This necessitates the separation of knowledge and data of different domains of expertise. In other words, there should be an expert for every domain of expertise and these experts should be self-contained in every sense. The experts should carry their knowledge and data regarding the design, their own user interfaces allowing them to interact with the user independently and their own methods which will allow them to reply to the queries of the user. Thus, the user will be able to get expert advise from all the different experts and arrive at a decision regarding the design.

The experts created should be able to emulate the behavior of real world "experts". This necessitates that the each expert should have the expertise pertaining to a specific domain of design. Thus, the user of the expert system can directly interact with these experts regarding problems and queries about the design. The experts should be designed in a manner that the interaction between the user and the experts should simulate the interaction between a designer and an expert in the real world. This means that the experts should be able to reply to all the queries of the user, warn the user if the parameters of the design are not compatible with the system and lead the expert through a design process so as to teach him the concepts of design pertaining to that particular domain of design. To make the system more adaptable and flexible, the experts should be able to learn from the design process being carried on so that the knowledge from previous design experiences are stored and could be used in later designs.

Each expert should be able to act in the different modes described above and contribute to the design process based on the knowledge and methods contained in each expert. The expert system developed using this framework should be tailored to meet the special needs and requirements of a CAD system, especially a parametric design system. The expert system developed using these tools should have methods to manipulate the parameters being used for the design process. It should have methods to get and store values of the parameters, calculate the values of the different parameters at any given time, should have equations for the calculations of the different parameters and maybe even be able to optimize the value of a parameter based on the design constraints.

The parameters to be used for the design and the equations and methods to calculate the different parameters should be input to the system by the programmer and should be stored as the knowledge of the different experts depending upon the domains which are affected by the parameters. The application programming environment should provide the programmer with the tools and the flexibility to perform the above-mentioned functions. The user interfaces which would include menus to setup and delete the various experts, menu items for the different experts, pop-up menus for the experts and dialogue areas for the interaction with the experts should be built into the system. Thus the programmer would have all the different user interfaces for the end user to interact with the experts available to him and would not have to go through the time-consuming tasks of building the user interfaces for the system.

The language chosen to develop this framework should provide the flexibility to allow the easy creation different domain-specific experts who are self-contained with their own knowledge and methods. Figure 2 shows the structure of the expert system that would be developed from the framework. This figure shows the knowledge, parameters, inference engine and the user interfaces belonging to the ET and interacting with the user. Figure 3 shows the interaction between the user and the experts in the expert system in a typical session. The framework developed should be able to create experts by using domain specific knowledge. These experts should be able to interact with the user to aid in the design. The language should also be one which is easily understood and used by a programmer/engineer. The language used to develop this system is important in the sense it should allow easy integration of the expert system developed with the overall CAD application being developed.

In summary, the requirements for the framework are:

1. The expert system created using the framework should closely approximate a real-world design process by having separate experts in separate domains. The experts should also be self-contained in the sense that each expert should have its own knowledge and methods about the design.

2. The experts should be able to interact with the user in different modes namely the consult, transact, observe, Teacher and the Student modes.

3. The programming environment should be developed using a language which is easily understood and used by engineers and programmers.

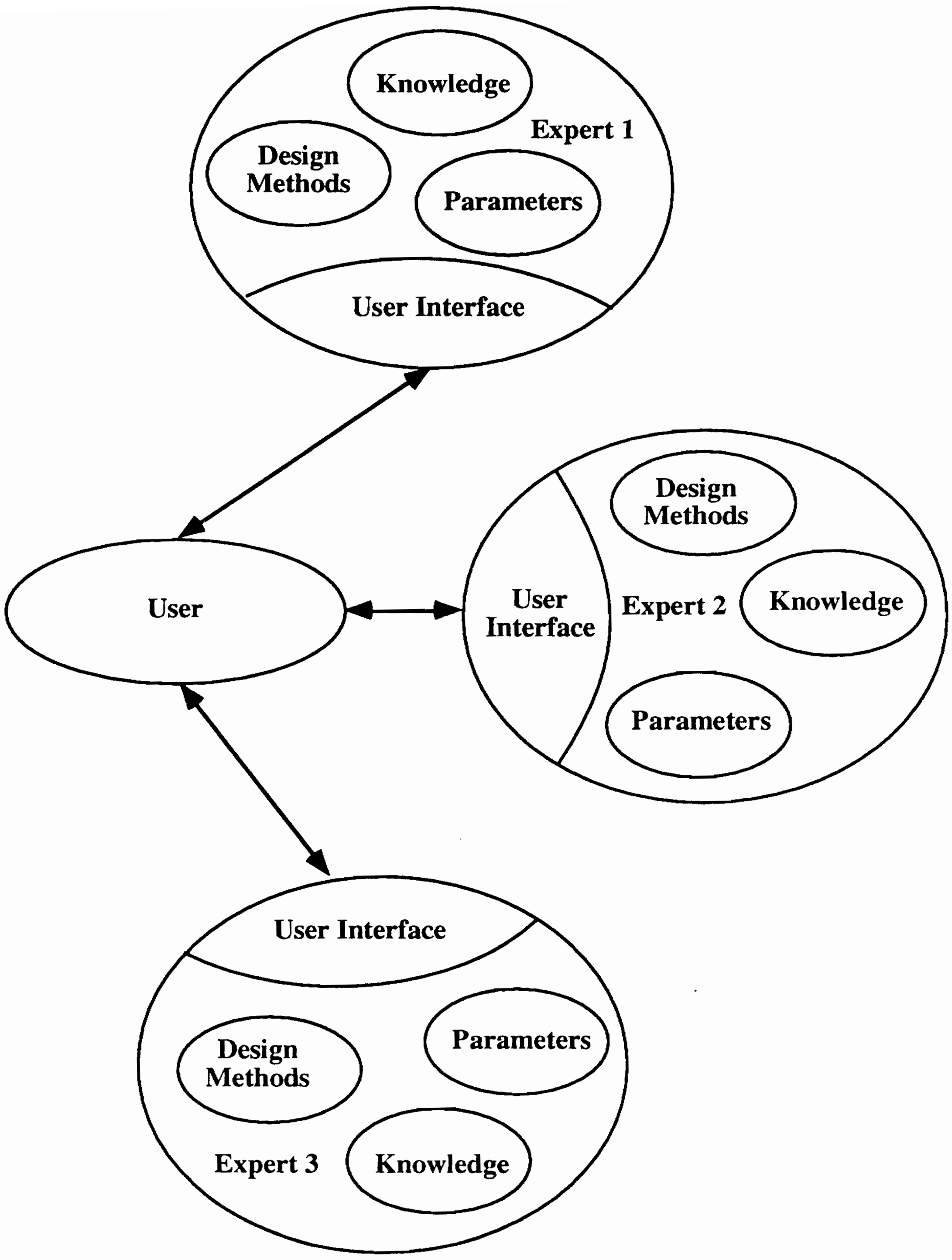


Figure 2 Interaction Between User and Expert System

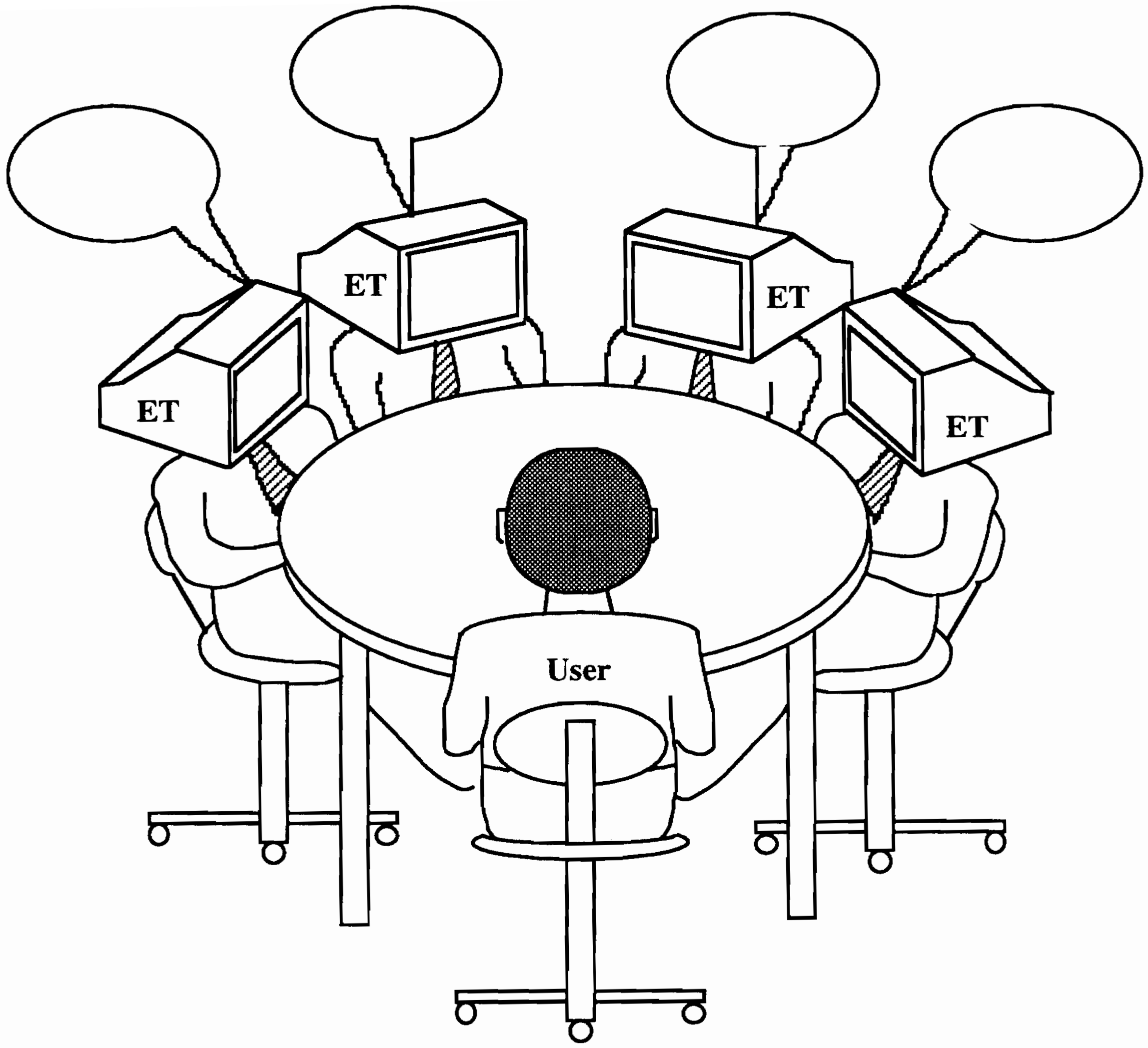


Figure 3 A Typical Expert System Session

4. The programmer should be able to tailor the expert system to satisfy the requirements of parametric design in CAD systems.

5. The tools should have functions for user interfaces and inference engines built into them.

6. The expert system created should allow the programmer to easily integrate it with the overall custom CAD/CAM system that is being built to suit specific needs of a particular company or organization.

6. Expert Consultation Environment

To meet the requirements of industry, academic institutions and research organizations that develop custom CAD systems, an Application Programming Interface [API] was suggested by Jayaram [Jaya89, Jaya90, Jaya93]. This design of the API is called CADMADE (Computer Aided Design and Manufacturing Application Development Environment). An implementation of CADMADE would allow the programmer of CAD/CAM systems to develop applications with the least amount of time and effort. CADMADE suggests that a set of high-level routines and programming interfaces be built into the system which a programmer would be able to use to set up the CAD system. These routines and interfaces are designed for repeatable use and would aid the programmer in getting rid of the time consuming tasks of having to build these routines from basics every time a CAD system is created. CADMADE is made up of a number of programming environments with corresponding data structures and two database managers, all of which can be accessed by the applications program. These environments are the User Interface Environment (UIE), PHIGS+ Environment, Design and Modeling Environment (DME), Virtual Manufacturing Environment (VME) and the Expert Consultation Environment (ECE). The UIE provides the applications programmer with procedures to create the graphical interface for the CAD/CAM application. The DME has procedures which can be used by the programmer for geometric modeling, design and analysis. The VME assists the programmer in creation of computer-aided manufacturing software and the ECE allows the programmer to create an expert system for CAD/CAM application.

The ECE would aid the programmer in developing an expert system as a part of the integrated CAD system. This environment would have methods to support forward and backward chaining, inference engines, search strategies and knowledge base containing rules for the custom expert systems. The expert system created using these tools would have the capability to provide replies to the queries of the user and provide expert suggestions based on the design parameters.

The central part of the ECE is the Expert Technician (ET). This ET consists of two parts that usually make up an expert system, the knowledge base and the inference engine. The knowledge base of the ET consists of the knowledge necessary for the ET to provide guidance to the designer about the design process. Typically this knowledge base would consist of information about the design in the form of definitions, equations, rules of thumb, design methodology, etc. This knowledge provides the ET with the expertise in a particular field of design.

An application using CADMADE has a knowledge base of its own and this knowledge is accessed with the routines available with the knowledge base manager. This knowledge base would typically consist of two parts - the rules and the parameters of design. The parameters which are used in the design process could be sent to the knowledge base with the use of the routines in the knowledge base manager. Each of these parameters could be given additional information in the form of constraints such as maximum and minimum values, default value, etc. The rules which are sent to the knowledge base are typically in the form of relationships between parameters. The rules are defined using an English like syntax.

The inference engine would consist of methods for forward and backward chaining and other search strategies. These methods built as a part of the ECE would be used by the programmer for reasoning and problem solving in the expert system. The inference engine would use the rules in the knowledge base and would directly act on the parameters of design to provide the user with the information for design.

CADMADE also suggests that the ET interact with the user in several different modes. These modes include the Transaction mode, the Dialog mode, the Observer mode, the Learn mode and the Teach mode. The transact mode is single-query single-reply mode. The consult mode is a multiple query-multiple reply mode wherein the expert replies to one or multiple queries with multiple replies . In the observe mode the expert acts as a person observing the design process and interrupts the design process with suggestions and warnings whenever necessary. In the case of the teach mode, the user teaches the ET about the design which in effect means that the knowledge is forced into the ET by the user. In the Learn mode the ET itself selectively adds to the knowledge base “learns” by using the output from the design process. These definitions of the “learn” and ‘teach mode” have been changed in this thesis .

An Object-Oriented Framework for the ECE

Based on the requirements for a high-level programming environment for the development of an expert system especially suited to meet the needs of the CAD industry and the guidelines provided by ECE in CADMADE, an object-oriented framework was designed. The framework is depicted in figure 4 which represents the class diagram for the framework. An object-oriented design specifically meets the requirements since one of the main objectives of the design was to emulate a real-world situation. This situation necessitates that the Expert Technician be the central part of the system and that each ET should be self-contained with its own data and methods. The expertise of the ET should be in a specific domain of design and for this the ET needs to carry the knowledge about the particular domain. Development of an ET class would aid the CAD programmer to setup experts for design by simply creating instances of the ET class. This would simplify the creation of the overall expert system for design.

This definition of the ET class closely approximates the real-world situation. Each instance of the ET represents a real “expert” who has the knowledge of a specific field of engineering. This “expert” has its own knowledge and methods for drawing inferences based on this knowledge. “Experts” also have their own methods for acquiring new knowledge and methods for teaching others. The communication skills of a person is also very important for communicating the knowledge to others. Since the methods are inherent in a person, they can be best simulated in the expert system by defining each expert or ET as an object with “skills” and knowledge built into them.

A programmer using this environment would just need to create instances of the classes to create an application suited to his needs. The functions necessary for the normal functioning of the system are built into the classes. This reduces the burden on the programmer every time an expert system needs to built. Functions for user interfaces, forward and backward chaining, etc. are built into the classes which can be used by the programmer and the designer wherever necessary. Moreover, an object-oriented approach would provide all the advantages listed in the previous chapter during the implementation of the software.

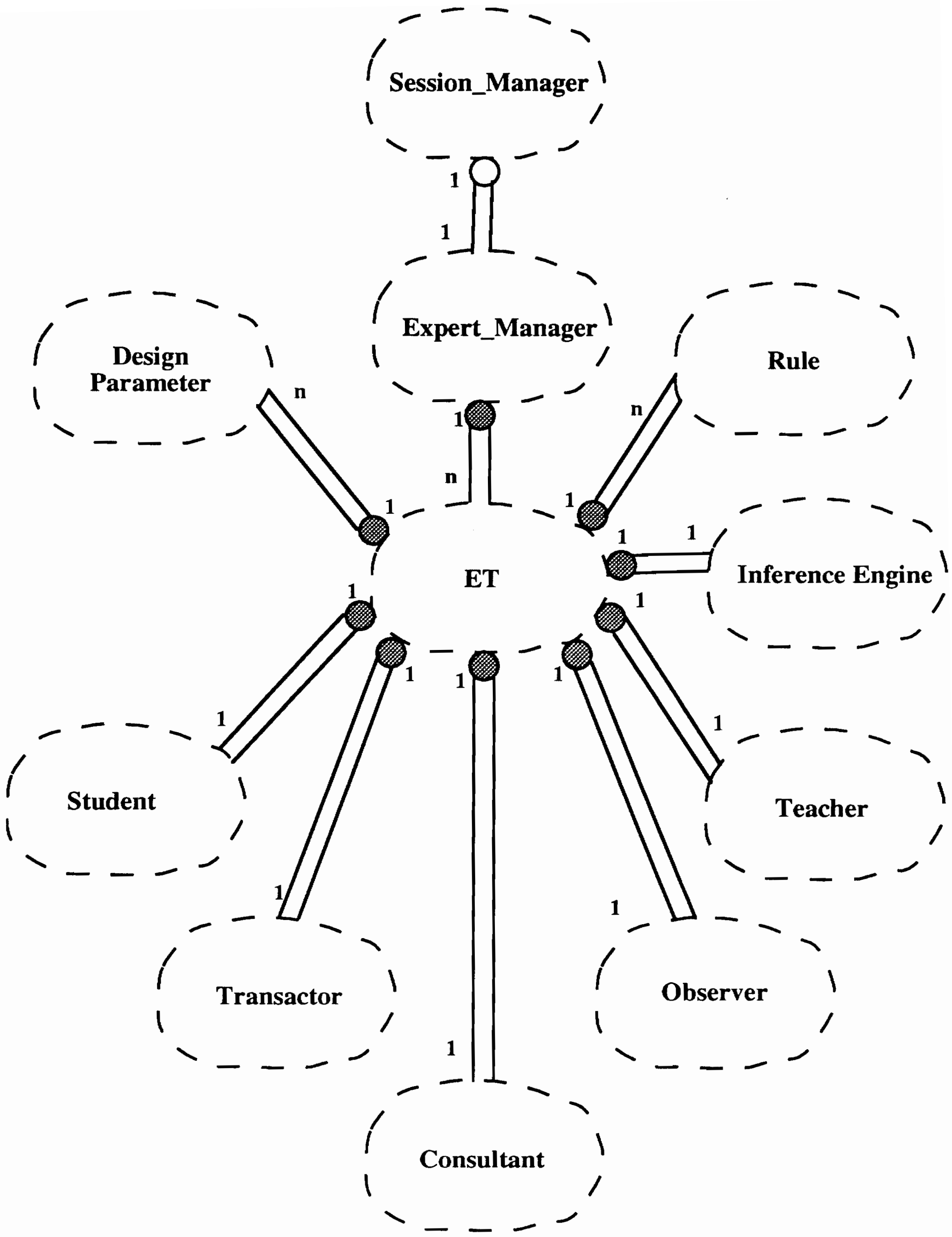
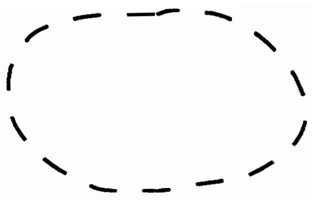


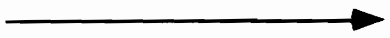
Figure 4 Class Diagram for the Framework

The class diagrams used in this thesis follow the standard representation of class diagrams in an object-oriented design that is explained in a great detail by Booch [Booc91]. Some of the main notations used are shown by figure 5. The blob represents a class in the design. This class would have data and methods but this data would not have any specific instance values. The arrow represents an inheritance. This means that the class towards which the tail of the arrow is pointing inherits all the data and the methods from the class at the head of the arrow. The other two notations represents a situation in which one class uses the data and the functions in the other class. In both these notations, the circle is placed at the class which is using the other class. The notation with the filled circle means that the class is using the other class, uses it for its implementation. That is the definition of the class would have one or more objects as a part of the data that is maintained. The notation with the hollow circle specifies that the class uses the other class for its interface.

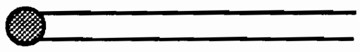
Figure 4 represents the class diagram of the main classes in the framework. The user interface classes are not shown in this figure. Figures 6 and 7 show the complete class diagram. A class has been designed for each of the objects that is a part of the expert system. As mentioned earlier, the ET is the central part of the expert system that will be developed using this framework. The ET class will contain data and methods for the functioning of an expert developed by instantiation of this class. The ET needs to operate in several different modes as specified by the requirements of the design. Each of these operating modes, namely, Consult, Transact, Observe, Teach and the Learn modes are implemented through classes with functions built into them that will make the operation of the ETs in these modes possible. The definitions of the Learn and the Teach classes are different from the definitions in ECE of CADMADE. These classes are a part of the ET and would also contain the functions necessary for the user interfaces for the



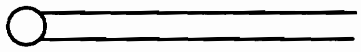
(a) Blob - Represents a Class



(b) Arrow - Represents Inheritance



(c) Uses for Implementation



(c) Uses for Interface

Figure 5 Notations Used in Object-Oriented Design

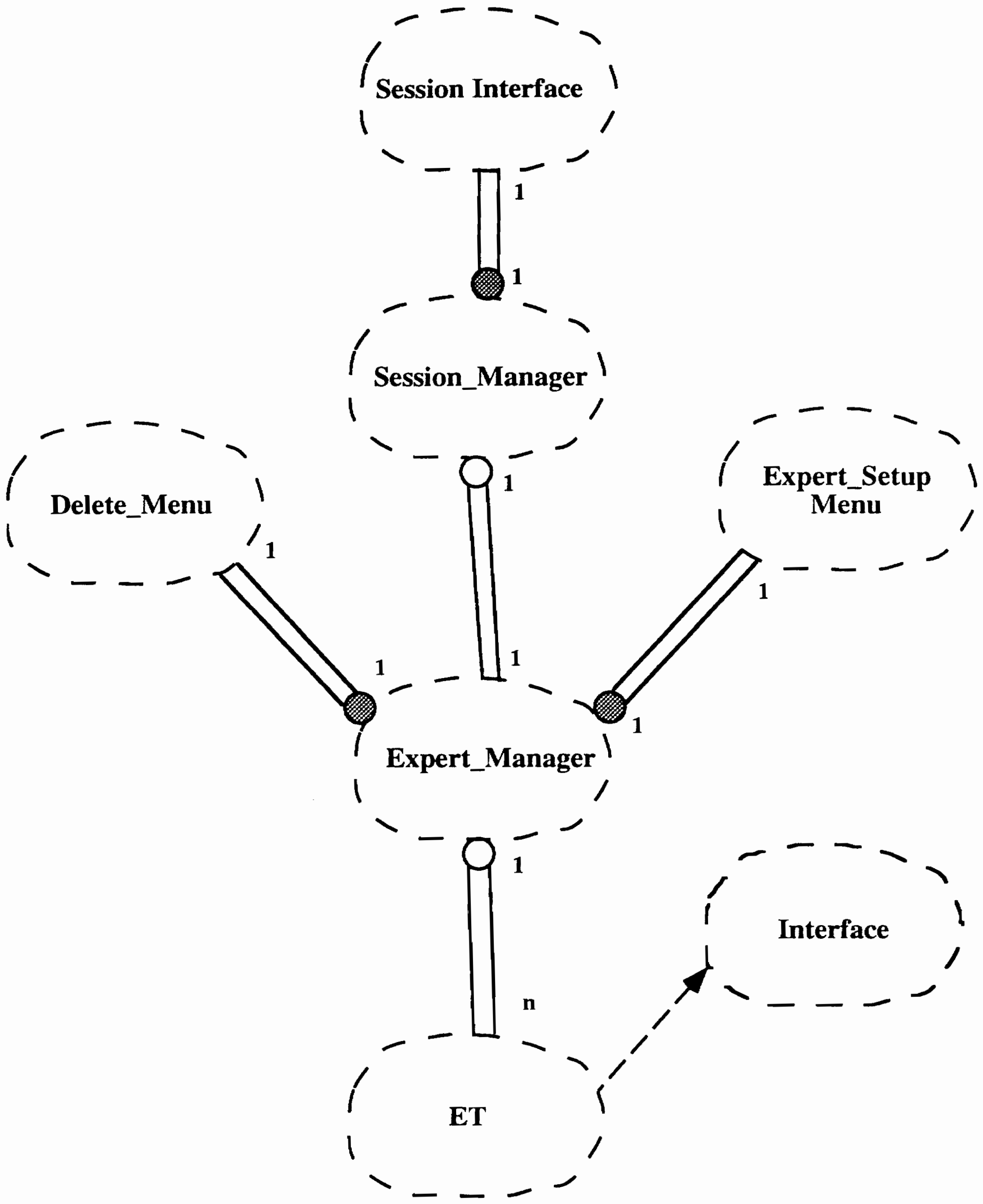


Figure 6 Class Diagram with User Interface Classes

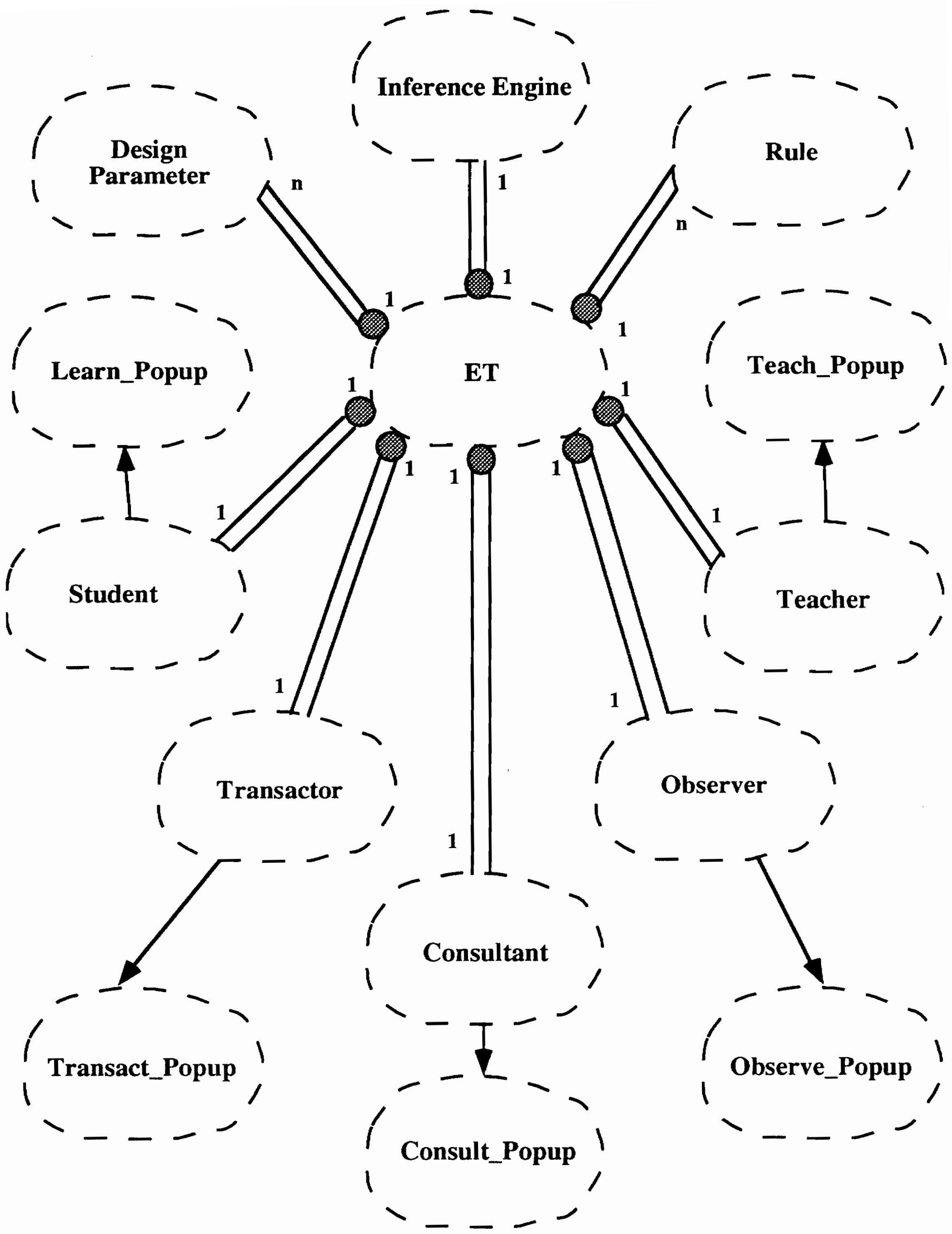


Figure 7 Class Diagram with User Interfaces - Continued

interaction between the user and the ET. The ET also uses a Design Parameter class. Objects (instances) of this class would be parameters used by the design system for the design of the component. The parameter class does not only represent a parameter data value but also has other methods for error checking, limit checking, etc. A Rule class is for creating objects of all the rules that an ET contains as a part of the knowledge. These rules could be equations, facts, design methodology, rules of thumb, etc. Objects of the rule class belong to the ET as a part of the knowledge of the ET.

The Inference Engine class is instantiated by the ET. This class contains all the methods for inferencing (forward and backward chaining), non-procedural parameter value determination, rule interpretation, etc. The inference engine uses the rules in the knowledge of the ET to which it belongs.

The framework also contains other classes which aid in the proper functioning of the expert system that would be built using these classes. These include the Session_Manager and the Expert_Manager classes. The Session_Manager class manages each session of the expert system. It should have functions built into it for the user interface such as the main window, dialog areas, main menus, etc. An Expert_Manager object is responsible for the managing of all the ETs that are created. It has functions for the creation and deletion of ETs, keeping track of all the ETs that are available and some of the interaction between the ETs and the user. A detailed description of the responsibilities of these classes and the functions to be built into these classes are available in the following chapters.

To create an Expert System, the programmer using this framework would create instances of the Session_Manager and Expert_Manager classes. The programmer would also need to create instances of the ET class to setup the various experts for the different domains of design. While creating the ET objects the programmer would need to specify the operating mode of the ET and the domain knowledge in the form of rules. Design Parameter objects are also created and are attached to various experts depending on the domain the parameter is likely to affect the most.

The user of the system can interact directly with the ET through their operating modes object. Each ET in a particular operating mode would have a menu item on the main window attached to it. Selecting these menu items would pop-up the various menus for the particular ET in the operating mode. The menu choices for the ETs would depend upon the operating mode of the ET. The user can query the ETs using these menus. The operating mode objects pass on the queries to the ET objects. The ETs make use of the knowledge available to them in the form of rules and searches through the knowledge for the information necessary to reply to the user's query. In this process the ET might also make use of the inference engine object. The reply to the users query is then written out to the user in the dialog area of the main window. Functions would be available to enable the user to pose a question to all the ETs that are active. This would be handled by the Expert_Manager which would pass on the query to all the ETs. Figures 6 and 7 show the complete class diagram for the framework.

The final design for the framework that is mentioned above has undergone a number of iterations. Through this iterative process the design for the framework has undergone a number of changes. Two of the earlier designs that were used for the framework earlier are shown in figures 8 and 9. Figure 8 shows the initial design for the framework. In this design of the framework, the user interface designed as a single class whose data and methods would be inherited by all the other classes that need to interact with the user. But the user interface function requirements for all the other classes is not common. For example, the ET needs to interact with the user through the operating mode classes. The

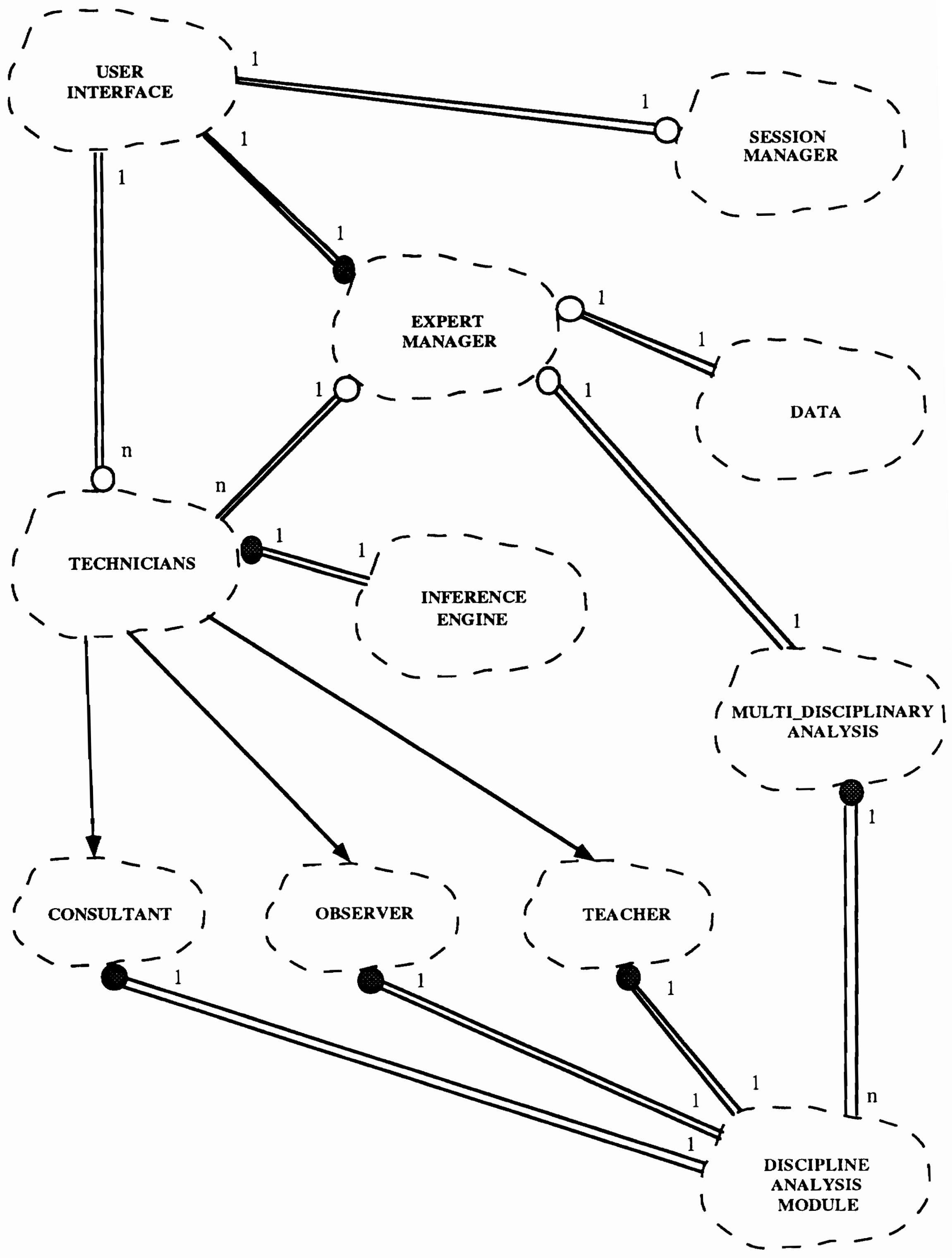


Figure 8 Class Diagram for the Initial Design of Framework

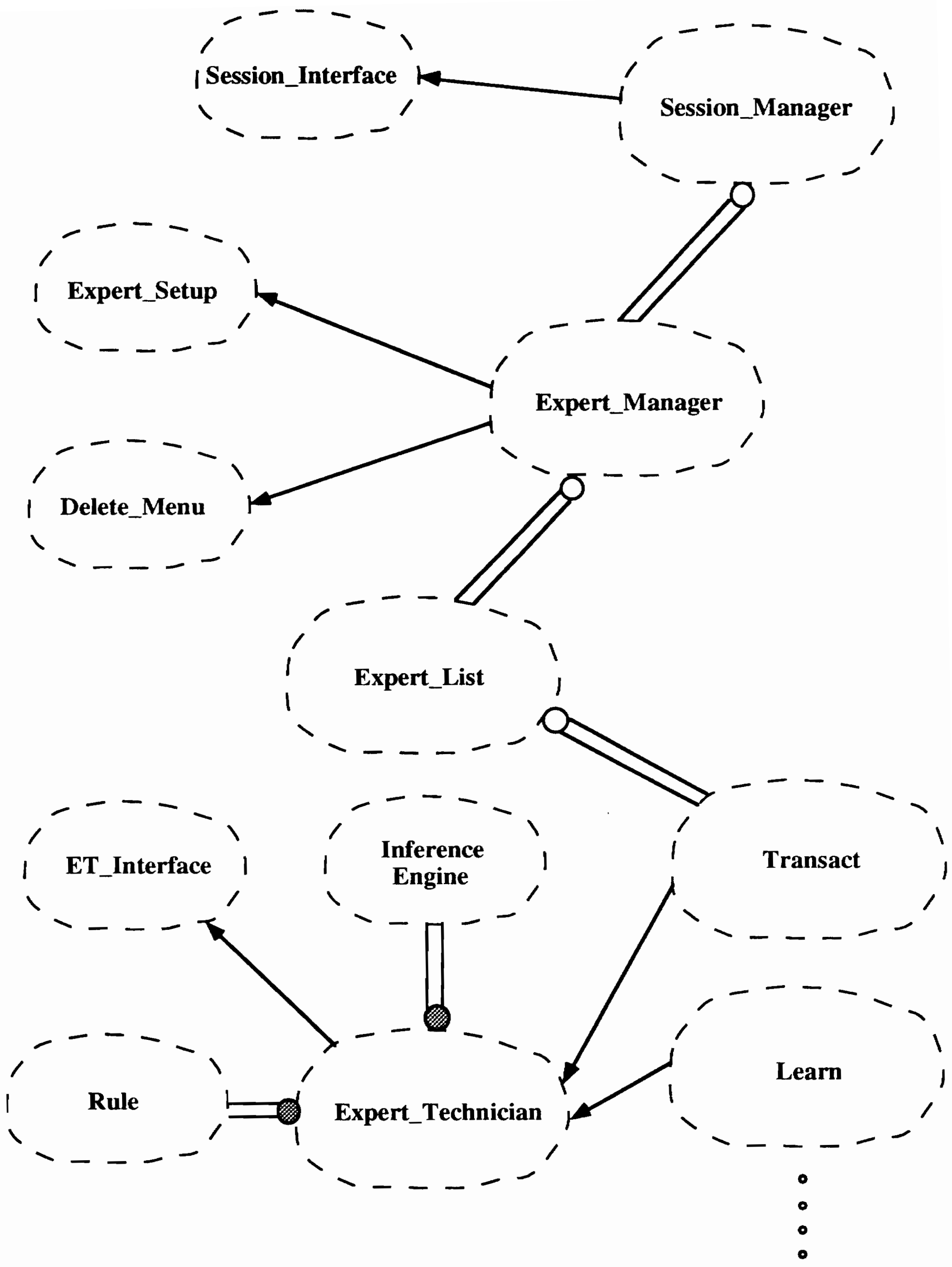


Figure 9 **Class Diagram of an Intermediate Design**

interaction of these operating modes of the ET with the user is different, thus, the user interfaces should be different. So it was decided that all the ETs should have different user interface functions built into them depending on the needs of the ET. In this case, the Session_Manager and the Expert_Manager are created by the programmer. This might lead to inconsistencies because the Expert_Manager takes care of all the ETs and the operating mode classes and creating two objects of this class would cause the splitting of information regarding the ETs between these two objects. In this design the Technician class is inherited by the operating mode classes. This would necessitate that the knowledge of an expert be stored in all the objects of the operating mode classes. This would mean that multiple copies of the knowledge would need to be created. A part of the knowledge of this class would be the design modules being used as a part of the CAD system. The operating mode classes would use the design modules for the calculation of parameters. The final design has a rule class wherein the equations used to calculate the parameters in the design module could be made an object of the rule class. This object would be used to determine the parameters of the design. The Data class in this design is used to store data pertaining to the domain of design. This could be handbook data such as material data, manufacturing data, etc.

Figure 9 shows an intermediate version of the design wherein the user interfaces are designed as separate classes and are inherited by the classes that need them. The interface methods in these classes are created to suit the needs of the classes that inherit them. A rule class and an inference engine class are used by the ET class for the implementation. The ET class is then inherited by the operating mode classes. This would create problems of maintaining several copies of the knowledge base in different operating mode classes. The Expert List class was designed to have information regarding the operating mode classes. This class would maintain a linked list of all the operating mode classes that are created. Later this class data and methods were merged with the Expert_Manager class since the manager should be keeping track of all the experts that are created.

7. Class Descriptions

This chapter and the following chapters explain in detail all the classes and the functions of these classes. The responsibilities of each class in the expert system that will be developed using this system are detailed along with the data and the functions that are present in these classes.

The Session_Manager class

This class manages the overall session of the expert system that will be developed using this framework. This class will be responsible for the starting of the expert system session and the setting up of the various user interfaces necessary for the functioning of the expert system. An object of this class is created by the programmer for each of the sessions of the expert system that needs to be invoked.

This class is also responsible for the creation of an object of the Expert_Manager class. This Expert_Manager class manages all the experts that are created in the expert system. If more than one session of the expert system is running at any given time, and if these sessions are working on the same design problem, the Expert_Manager object that is created in each of these Session_Manager objects needs to be the same. That is if more than one designer is working on the same problem, more than one session of the expert system needs to be invoked. For this reason the Session_Manager class has functions to get the Expert_Manager object from one session and store it as the Expert_Manager object in another session. This ensures that both the sessions will run using the experts with the same data and knowledge.

The user interface functions are built into another class, the Session_Interface class. The Session_Manager class inherits all the data and methods from this class. All through the design of this framework the user interface functions have been separated from the other functions as far as possible to keep the design simple, easy to understand and easily maintainable. This also allows for easy changes in the future where better user interface can be used to replace old ones.

Public Functions

Constructor:	The constructor for this class creates the Expert_Manager object and the Session_Interface object. All the user interfaces are setup in the constructor of the Session_Interface class.
Get_Manager:	This function will return the Expert_Manager object being used in the expert system session.
Add_Manager:	This function adds the Expert_Manager object that is passed in as the Expert_Manager object for the session.

Manager: This data member is an instance of the Expert_Manager class. This object is created in this class and passed out to other sessions if any.

The Session_Interface class

This class acts as a supporting class for the Session_Manager class in the sense that all the user interface functions necessary for the Session_Manager class are created by this class. This class is then inherited by the Session_Manager class. This class is responsible for opening the main window for the expert system session. It has methods to open the main window, put up menus necessary for the operation of the expert system and open up dialog windows necessary for the interaction between the user and the experts. The experts output their replies to the user in the dialog window. It also has methods for the creation of windows within the main window where the menu items for the experts will come up. It also creates an exit menu item on the main menu of the system, selecting which, the system exits from the expert system session.

Public Functions

Constructor: The constructor for this class creates the various user interface objects specified in the previous section. As mentioned earlier, an exit menu item is created on the main menu area. Selecting this menu item creates an object of the Exit_Dialog class. The Exit_Dialog class allows the user to exit the system.

The Exit_Dialog Class

An object of this class is created when the exit menu item on the main menu is selected. This class then has functions to close all the windows, delete all the existing objects in the system and ending the expert system session.

Public Functions

Constructor: The constructor deletes all the existing objects of the Expert_Manager class, the ET class and the Session_Manager class and closes the window of the application and exits the system.

The Expert_Manager Class

An object of this class is created by the Session_Manager class. This class is responsible for managing all the ETs that are created by the user. It keeps track of the number of ETs that are created and their modes of operation. This class includes functions for creating new ETs, deleting ETs and creating and deleting the operating mode behavior of each of these classes. Since the Expert_Manager keeps track of all the experts, it is necessary that for the sake of consistency the same Expert_Manager object is used wherever information regarding the ETs is necessary. This is the reason that while using multiple sessions of the expert system, the same Expert_Manager object is used in all the sessions and this object is passed to all the sessions in use.

The Expert_Manager keeps track of the ETs and their modes of operation by maintaining linked lists for each of these and is shown by figure 10. It maintains a linked list of the ETs which have been created. Whenever a new ET is created it is added to the linked list maintained by this object. The Expert_Manager also maintains a linked list for each of the operating modes of the ETs . Since each of the operating modes is a class of its own, when the ET is interacting with the user in a particular mode, it creates an object of that operating mode class in the ET object. A linked list of these objects are maintained in the Expert_Manager object. This linked list is maintained in the Expert_Manager object to keep track of the operating modes of each of the ETs and the number of each of these operating modes objects that are created.

The responses to the user from each of the ETs is sent to the Expert_Manager object. It is the responsibility of the Expert_Manager to present it to the user. This class has functions to write out to the dialog window. Since the responses from all the ETs are being output to the user through this function, a history of the expert system session could be maintained. Even the queries that are posed by the user to the different user are echoed back to the user by displaying it on the dialog window. Thus even the queries posed to the ETs can be saved to the file providing the designer with a complete history of the expert system session.

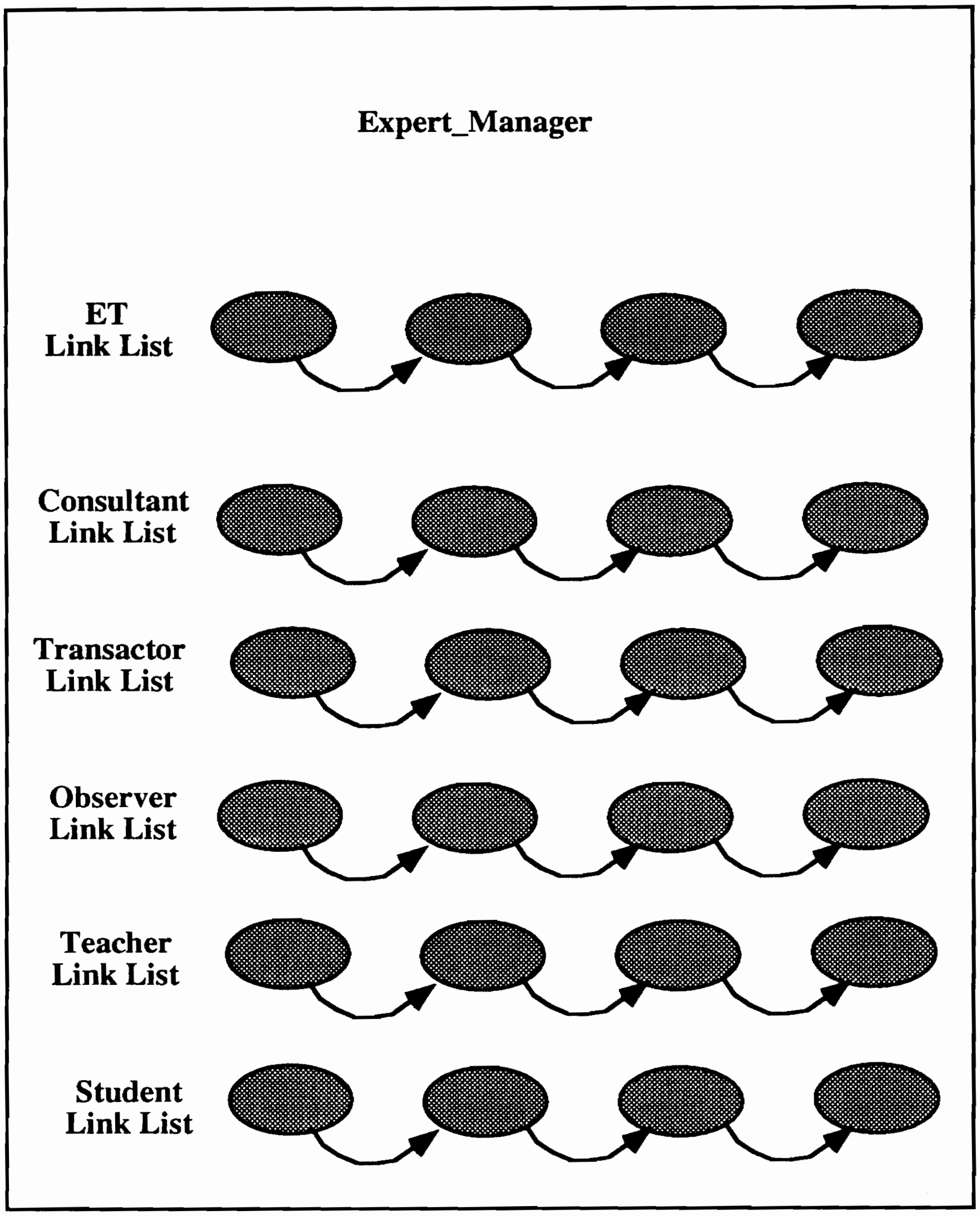


Figure 10 **Linked lists in the Expert_Manager Class**

Since the Expert_Manager is responsible for the creation and the deletion of the ETs and their operating modes, the menus necessary for these functions should be carried by the Expert_Manager. To keep the user interface functions separate from the other functions for the operation of the expert system, these are built into separate classes and the Expert_Manager class uses these classes. The Expert_Setup_Menu class is responsible for the user interface for the creation of the operating modes of the ETs. The Delete_Menu class is responsible for the creation of menus for the deletion of the expert modes.

Public Functions

Constructor:	This function initializes all the variables that keep track of the number of ETs and the operating modes that are created. It also initializes the various linked lists maintained in the object.
Write_To_User:	This function writes to the user the responses of the ETs to the user's query. It writes the string sent to it by the ETs to the dialog window.
Add_Consult:	Adds a Consultant to the linked list of Consultant class objects maintained by this class object.
Add_Observe:	Adds an Observer mode object to the linked list of Observer objects maintained by this class object.
Add_Transact:	Adds a transact class object to the linked list of Transactor mode objects maintained by this class object.

Add_Teach:	Adds a teach expert object to the linked list of Teacher mode objects maintained by this class object.
Add_Learn:	Adds a Student object to the linked list of Student mode objects maintained by this object.
Remove_Consult:	Removes the Consultant object from the linked list of Consultant mode objects maintained by this object.
Remove_Observe:	Removes the Observer object from the linked list maintained by this object.
Remove_Transact:	Removes the Transactor object from the linked list maintained by this object.
Remove_Teach:	Removes the Teacher mode object from the link list maintained by this object.
Remove_Learn:	Removes a Student mode object from the linked list maintained by this class object.
Add_ET:	Adds an ET object to the linked list of ET objects maintained by this object.
Get_First_ET:	Returns the first ET object in the linked list of ET objects.
P_Value_Changed:	This function is called by the Transactor and the Consultant ET when the value of a particular design parameter is changed. This function in turn accesses all the Observer objects from the linked

list that it maintains and sends a message to each one of them that the value of a parameter has changed.

Get_Value: Returns the value of a particular parameter. It searches through the linked list of ETs for the parameter. Once the parameter is located, it sends a message to the parameter to return its value and passes on the value to the ET that requested it.

The Expert_Setup Class

This class is responsible for the creation of the user interface to create an operating mode object in the ET. As mentioned earlier, the ETs operate in different modes. Each of these modes is a class and if the user needs a specific ET in a particular mode, an instance of that particular operating mode class is created in the ET. An object of the Expert_Setup_Menu class includes user interfaces for each of the ETs that are available. These include menus in which each of the inactive modes of the ET has a menu item. selecting the menu item would create an operating mode object in the ET. This class has methods which call the specific functions in the ET class to create an object of an operating mode class. This class is used by the Expert_Manager class.

Public Functions

Constructor: The constructor creates the menu items on the main menu, selecting which displays all the available ETs.

Update_Setup: This function updates the menu which displays all the ETs and the available modes whenever an operating mode class object is created in an ET. This is because this menu displays only those operating modes which have not yet been created in the ET.

The Delete_Menu class

This class is similar to the Expert_Setup_Menu class except that it is menu used for the deletion of a particular operating mode of the ET. This has functions to put up menu items for each of the ET and the modes in which the ET is presently operating. Selecting these menu items deletes the object of the operating mode class. This class is also used by the Expert_Manager class.

Public Functions

Constructor: This function puts up a delete menu item on the main menu of the application.

Update_Delete: This function updates the delete menu. Since only the current modes in which the ET is operating in is displayed in this menu, whenever an operating mode class object is created or deleted, this function is called to update the delete menu.

The Design_Parameter class

Since one of the objectives of this framework was to support parametric design, this class was designed. An instance of this class would represent a parameter that is being used for the design. The Parameter object not only represents the data value of the parameter but also has methods for constraint checking, error checking, etc. This object belongs to a specific expert (ET) depending on the domain this parameter is expected to affect the most. A linked list of Parameters is maintained in each of the ETs to keep track of all the parameters that belong to a certain ET. The Parameter object is created during the creation of the ET object by the programmer who specifies the default value for the parameter. This is done by reading in a stored object during the creation of the ET objects. The programmer is responsible for the creation of this stored object.

A parameter being used for design might be of any data type. For accommodating this flexibility has been provides to make an object of this class either to have a float or a string value. The programmer while creating the Parameter object specifies what type of a parameter it is. The class has functions to store both character and float values. It also has functions to return both type of values.

The Design_Parameter is more than just a design space. This class has been provided such that the user can create any type of parameter that is needed. For example this design parameter could be a float or a string. This also provides flexibility to the programmer developing the expert system to create any number of these in the ETs by just creating objects of this class. At a later stage when the expert system needs to be integrated into the overall CAD environment, functions could be built into this class to retrieve data from the database of the system or from an analysis routine that calculates the value of the parameter.

Public Functions:

Constructor:	This function creates an object of this class. While creating the object, the system has to know what kind of a parameter it is. A default value has also to be sent in to this constructor to be stored. Each parameter has a name and this string is passed into this constructor.
Get_Value:	This functions returns the current float value of the parameter.
Get_Char_Value:	This function returns the character value of the parameter, if the parameter happens to be of the type character.
Put_Value:	This function stores as the current float value the value that is passed into this function.
Put_Char_Value:	This function stores as the current character value, the string that is passed in.
Set_To_Default:	Sets the value of the parameter to its default value.

8. Expert Technician and the Operating Mode Classes

This chapter explains in detail the Expert Technician (ET) class, the operating mode classes, namely the Consultant class, the Transactor class, the Observer class, the Teacher class and the Student class and all the related classes.

The ET Class

This class represents the expert in a particular domain of the design. An instance of this class is self-contained with the knowledge and methods for a particular domain of design. The programmer creates an object of this class in the main program of the application. During creation, this object is supplied with the knowledge base and this knowledge base contains the rules in the form of equations, rules of thumb, design methodology, etc. pertaining to a specific domain of design.

One of the objectives of the design of the framework was to create these ETs such that their operation closely approximates the way experts interact with designers in a real-world situation. One method for accomplishing this was to have the ETs interact with the user in different modes referred to earlier as the operating modes. The different operating modes that are required were identified earlier [Jaya89, Jaya91, Jaya93]. These operating modes are the Consult mode, the Transact mode, the Observe mode, the Teach mode and the Learn mode. Of the five above mentioned modes the first three actually interact with the user and provide replies to queries, suggestions, warnings etc. The latter two have been provided to give the expert system more flexibility and adaptability.

These modes are different in the way they interact with the user and aid in the design process. While operating in the consult mode and the transact mode, the ET replies to the various queries from the user. The user poses the queries using the pop-up menus for the experts. The consult mode can be described as a multiple query - multiple reply mode in which the ET may reply the multiple queries with a single reply or a single query with a multiple reply. In the transact mode the session is more of a single query - single reply mode. Upon the receiving the query the ET searches through its knowledge for the rules that would provide a solution to the query. The search through the knowledge is carried out with the use of the inference engine. The way this search is carried out and replies are given to the user depends on the mode in which the ET is operating. The exact method of interaction between the ET operating in a particular mode is detailed in the class descriptions of these operating modes.

In the observe mode, the ET acts as an Observer to the design process and provides useful suggestions and warnings about the design process as and when necessary. The ET in this mode has access to all the design parameters pertaining to the particular domain of design and an independent analysis of the design is carried out by the ET. Based on the results of the independent analysis and the various rules and constraints available to the ET in the knowledge base, the ET arrives at independent conclusions about the design. Based on these conclusions the ET interrupts the design process with suggestions and warnings.

The teach mode and the learn mode are designed to provide more flexibility and adaptability to the system. In the teach mode the ET guides the user through the design process. This is done by making use of the rules in the knowledge-base and since these rules are a collection of design methodology, equations, rules of thumb, constraints etc., the system would be able to guide the user through the design process based on the requirements of the user for the design.

In the learn mode, the system learns from the ongoing design process. For example if the design process is being carried out by an experienced person in the particular field, the system would be able to query the user for the reasons for the decisions that were taken during the design. These decisions and reasons could be stored as a part of the knowledge and are used in later designs. In effect, what is being done here is that the knowledge of experienced designers could be stored as a part of the knowledge of the system, thus capturing the experience of these designers. There are two methods in which the systems acts in the Learn mode, the voluntary learn and the forced learning. The user decides on which of these modes the learn expert acts. During the involuntary learning process, the system tries and learns all the important decisions and queries the user for the reasons. This is a case of a classic AI machine learning process. In the forced learning process, the system lets the user input knowledge as rules directly into the system. This is done during the design process and the user need not exit the system to alter the knowledge base.

To integrate the expert system with the overall CAD system being developed, knowledge has to be received from the analysis modules of the CAD system to the expert system. This interface could be provided by declaring the rest of the CAD system as an expert, inheriting the methods and data of the ET class and redefining all the functions of the ET class to suit the analysis module. This has been represented as an Interface class in the class diagram in figure 6. The other ETs would be able to receive knowledge and information from this interface class ET through the Expert_Manager object. If the user wants to use just a function from the analysis module, this function could be declared as a rule.

The programmer using this framework creates the ET objects in the main program of the application. While creating this object, the knowledge and the parameters for the object are read in from an archived object which the programmer creates. As the rules are read in, objects of the rule class are created and stored as linked lists in the ET. Similarly, as the parameters are being read in, objects of the Parameter class are created and stored as a linked list in the ET. Thus each ET created has its own knowledge and data about the design that is being carried out. The rules give the ET expertise in a certain domain of design.

To operate in a particular mode, the ET creates an object of the operating modes class inside the ET object. A representation of the ET class is shown by figure 11. The object of the operating modes class interacts with the ET in the sense that the user queries the ET through these operating modes class. In effect only two of the operating modes class interact with the user - the consult class and the transact class in a manner that questions are posed to them and reply to these users queries. These class objects have pop-up menus that have choices for the user to select and to query the ET. This query is then passed on to the ET which uses the inference engine to search through and arrive at a decision regarding the query. This reply is then displayed to the user using the functions in the Expert_Manager object. When the user wants the ET to cease acting in a particular mode, the object of that particular mode maintained in the ET is deleted.

ET Class

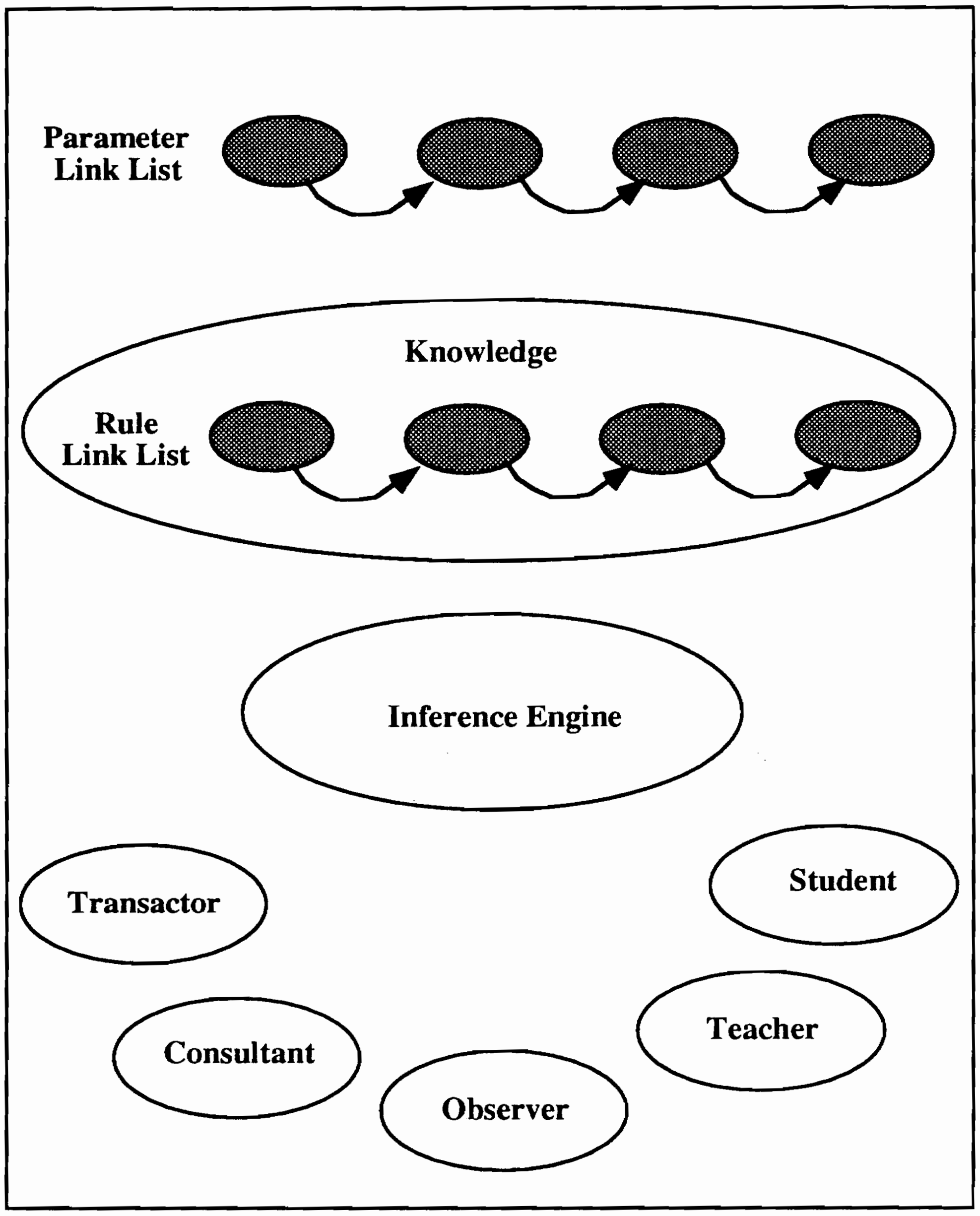


Figure 11 ET Class Representation

The ET class maintains a linked list of Parameters being used in the design process. These Parameters are those which tend to affect that particular domain in which the ET has the expertise in. These Parameters are created at the time the ET object is created by the programmer. There are functions built into this class that also allow the ET to add another Parameter to the linked list whenever necessary.

Public Functions:

Constructor:	The constructor for this object reads in the knowledge base that contains the rules and the Parameters. An object of the operating modes class is also created in this function for the mode through which the programmer wants the ET to operate initially. The default for this is the consult mode.
Create_Expert:	This function creates an object of the operating mode specified by the parameter passed into this function. An indicator for that mode is then set to “active” in the ET. This indicator specifies which of the modes the ET is presently acting in.
Delete_Expert:	This function deletes an object of the operating mode that the user wants the ET to stop operating in. The indicator is then set to “inactive” for that mode.
Get_Name:	Returns the name of the ET.
Get_ID:	Returns the identifying number of the ET.
Add_Parameter:	This function creates a Parameter object and adds it to the linked list

Add_Rule:	This function would be called by the Student object in the ET to add a rule to the knowledge base that has been "learned". An object of the rule class is created in this ET and added to the linked list of rules maintained by the ET.
Get_Value:	Returns the value of a parameter. This parameter might be a Parameter maintained by this ET object or any other ET object. It searches through its own Parameter linked list to find the parameter. If the parameter is found then the ET returns the value of the parameter. If it is not able to find the particular parameter in its own linked list, it sends a message to the Expert_Manager object for the value of the parameter. The Expert_Manager object in turn sends a message to other ETs for the value of the parameter.
What_Is:	This function is called by the Transactor and the Consultant when this menu item is picked from the pop-up menus for the operating mode. This function gets the current value of the parameter and displays it to the user.
Why:	This function is called by the Transactor and the Consultant when this menu item is selected from the pop-up menu. This function displays all the current values of the design parameters and sends a message to the inference engine to display all the rules that were used to arrive at the last decision of the ET.

How: This function would be called by the Transactor and the Consultant objects when the user wants to know how a particular reply was arrived at. The ET displays its name to let the user know where the reply came from.

Determine: This is a menu choice with the Transactor and the Consultant. This message is received when the user wants to calculate the value of a parameter. If this function is called from the Transactor, the value of the parameter is calculated using the rules and the inference engine and the ET lets the user know what the new value of the parameter is. If this function is called from the then calculates the values of all the other parameters affected by this change and displays to the user values of all these parameters.

Increase: This is a menu choice in the Transactor and the Consultant objects. Upon selection of this menu choice this function in the ET is called. This function increases the value of the parameter by the user specified amount. Then the values of the other parameters is calculated. In the transact mode, the user would then use the “Determine” menu choice to determine how this change has affected other parameters. In the consult mode, the ET would determine the values of all the other parameters and display this to the user.

Decrease:	This function is similar to the "Increase" function except that the parameter value is reduced by the user specified amount.
Validate:	This function would be called by the Observer object. This function sends a message to the inference engine object to check if any of the constraint rules are violated. If any of the rules are violated, the ET warns the user about the violation. Suggestions are provided by this function to avoid the violation of rules if possible.
Teach_User:	This function would be called by the Teacher object when it is set active by the user. This function would send a message to the inference engine object to start a forward chaining process to trace the parameters from the input parameters until the output parameters are reached. Appropriate equations and constraint rules are also output to the user to "teach" him the knowledge about the domain of design.

The Consultant Class

An object of the ET class is required to interact with the user in several modes. The Consultant class represents one of those operating modes. An object of this mode would aid the user to interact with the expert in the consult mode. The consult mode is a multiple query - multiple reply mode. In this mode of operation, the ET might choose to reply to a single query of the user with multiple replies or multiple queries of the user with a single reply. An object of this class belongs to an object of the ET class. When the ET wants to operate in this particular mode, an object of this class is created in the ET object. The user interacts with the Consultant object and the Consultant object passes on the queries to the ET which makes use of the inference engine object and the knowledge base to reply to the user's queries.

The Consultant object has methods to create the user interfaces which lets the user interact with the ET. The Consultant object, as it is created in the ET object, creates a menu item for itself and posts it on the main window of the expert system application. Selecting this menu item displays the pop-up menus for the ET operating in this particular mode. The user queries the ET using the menu choices on the pop-up menu. The consult object on receiving the queries through the pop-up menus, passes these queries to the ET object. Thus an object of this class would just facilitate the interaction of the ET with the user. When an object of this class is created within the ET, the object is added to a linked list of Consultant objects maintained in the Expert_Manager object.

The menu choices on the main pop-up menu for the Consultant ET are as follows :

What is: When the user selects this menu choice, another pull-down menu with a list of all the parameters belonging to that expert is displayed. If the user selects any of the parameters, a small description regarding the parameter along with the current value of the parameter is displayed on a pop-up window.

Why: This menu choice is for the user to find out how the system arrived at a particular reply. The Consultant object displays to the user the values of the various parameters and the rules that the system used to arrive at the decision. For this the inference engine would keep track of all the rules that were fired and would display to the user all these rules and the parameters used in these rules.

How: This menu choice would display to the user a list of the experts which were used to arrive at the reply to the query.

Increase: This menu choice is available to the user to determine the impact of increasing the value of a particular parameter on the other parameters or the design. Selecting this menu item would display a list of parameters that belong to a particular expert. Selecting any of the parameters would display a window with the current value of the parameter and a request for the user to enter the new value or a percentage increase.

Decrease: This menu choice is similar to the Increase menu choice except that the user will receive a reply on the impact of decreasing the value of the parameter on other parameters.

Determine: Selecting the Determine menu item would display a list of parameters belonging to this particular ET object. Selection of one or more of the parameters would send a message to the ET object to determine the value of the parameters. The value of the parameters are calculated anew from the rules that determine the values of the parameters. A message is sent to all the Parameter objects, whose values are determined from the parameter that had just been changed, that the value that they currently hold is invalid. The new calculated value of the parameter that was selected earlier is then displayed to the user. If the user wants to determine the values of all the parameters, a menu choice is available to the user which would allow him to do so. All the data and methods are inherited by this class for its user interfaces.

Public Functions

Constructor:	The constructor creates an object of this class. It also creates a menu item for the ET object operating in the consult mode in the main window. All the data and methods of the Consult_Popup class are inherited.
Get_Name:	This function returns the name of the expert that this consult mode belongs to.
What_Is:	This function is called when the user selects the "What is" menu item from the pop-up window of the Consultant object. This function sends a message to the ET object to display the value of the parameter selected.
Why:	This function is called when the "Why" menu item is selected from the pop-up menu of the ET. This functions sends a message to the ET object that this object belongs, to display to the user the values and the rules used to arrive at the last decision.
How:	This function is called when the user selects the "How" menu choice from the pop-up menu of the Consultant. This calls the How function of the ET object.
Determine:	This menu choice would be selected by the user to determine the values of one or more design parameters. This function is called when this menu choice is selected. This function sends a

message to the ET object that this Consultant belongs to determine the value of the parameter.

Increase: This function is called when the user needs to increase the value of a parameter. The value of the parameter would be increased by a percentage amount that is specified by the user. When the user selects the menu choice for this, this function displays a prompt window where the user inputs the percentage amount the parameter value needs to be increased by.

Decrease: This function is similar to the previous function except that the user would decrease the value of the parameter.

The Consult_Popup Class

An object of this class is inherited by the Consultant class. This class creates all the user interfaces necessary for the interaction between the user and the ET operating in the consult mode. This class has functions to create the pop-up menus. The pop-up menus contain the necessary menu choices for the user to query the ET.

Public Functions

Create_Popup:	Creates the main pop-up menus for the Consultant class objects.
Create_Pulldown:	This function creates the pull-down menus which are attached to the main pop-up menus. These pull-down menus would contain further menu choices and would depend on what the main pop-up menu choice was.
Create_Increase:	Creates a window where the user specifies the amount by which the value of the parameter has to be increased by.
Create_Decrease:	Creates a window where the user specifies the amount by which the value of the parameter has to be decreased by.

The Observer Class

An object of the Observer class is responsible for providing suggestions and warnings to the user during the design process. This object observes the design process being carried out and has access to all the Parameters and the rules. If the user exceeds any of the design constraints or violates any of the rules this object interrupts the user and warns him of the constraint violation. The way this is carried out is, whenever the user makes a change to any of the parameter values or executes a process that changes the values of the parameters, a message is sent to the Expert_Manager object that a parameter value has changed. The Expert_Manager in turn sends a message to all the Observers that a value of a parameter in a specific ET has changed. The ET object, through the inference engine checks the validity of the parameters or if any of the rules have been violated and warns the user about it.

This class inherits all the user interfaces functions from the Observe_Popup class. Selecting the menu item for the ET operating in this mode displays the pop-up menus for this ET. There are menu items available for either activating or deactivating the observe mode ET. Activating the ET would cause it to start "observing" the design process. It would then start interrupting the designer with warnings and suggestions. Deactivating the ET would stop it from operating in this mode. The Observer class object is not deleted but it stops providing warnings to the user.

Public Functions

Constructor:	The constructor for this class creates an object of this class and creates a menu item for itself on the main window.
Get_Name:	This function returns the name of the ET that this operating mode class object belongs to.
Set_Inactive:	This function is called when the user wants to set this object inactive. When this object is set inactive, it stops providing warnings and suggestions to the user.
Set_Active:	This function is called when the user wants the Observer to start providing suggestions and warnings.

Validate: When the Observer receives a message from the Expert_Manager object that the value of a parameter is changed, this function is called to make sure that none of the constraints are violated. This function sends a message to the inference engine class to validate the value of the changed parameter.

The Observe_Popup Class

This class is responsible for creating all the user interfaces necessary for the operation of the ET in the observe mode. It creates a pop-up menu for the ET which is displayed once the menu item for the ET on the main window is selected. This class creates all the menu items for various menu options which are displayed on the pop-up menu. This class is then inherited by the Observer class.

Public Functions

Create_Popup: This function creates the pop-up menus for the interaction between the user and the ET. This function creates the pop-up menu and creates menu items for activating or deactivating the observe mode ET on it.

The Transactor Class

This class is similar to the Consultant expert in the sense that it interacts with the user in replying to his queries. This class object will transact with the user in providing replies to his queries about the domain of the design that the ET has the expertise in. It is different from the consult mode in the sense that this is a single query - single reply mode. For each of the user's query, it accesses the knowledge base and replies with a single reply. When the ET receives the query, the knowledge base is accessed and searched for the reply. The ET does not reply to the user what further implications that reply will have on the other domains or parameters. Thus, it replies to the users queries with specific replies and does not provide further suggestions.

The user interacts with the Transactor ET using the pop-up menus for the Transactor class. This transact mode ET creates a menu item for itself on the main window of the application. Selecting this menu item would create pop-up menus for the ET. The ET has several menu items on the pop-up menus which the user would use to query the ET. These menu choices are also similar to the ones in the Consultant class. But replies to the users queries are different in this class. For example in the Consultant class if the user changes the value of one of the parameters, the values of all other design parameters that belong to the ET are also updated. But in this case only the value of the design parameter that was changed would be updated.

Public Functions

Constructor:	The constructor for this class creates a menu item for itself on the main window of the application. All user interface functions are inherited from the Transact_Popup class.
Get_Name:	This function returns the name of the ET that this operating mode belongs to.
What_Is:	This function is called when the user selects the "What is" menu item from the pop-up window of the Transactor object. This function sends a message to the ET object to display the value of the parameter selected.
Why:	This function is called when the "Why" menu item is selected from the pop-up menu of the ET. This functions sends a message to the ET object that this object belongs, to display to the user the values and the rules used to arrive at the last decision.
How:	This function is called when the user selects the "How" menu choice from the pop-up menu of the Consultant. This calls the How function of the ET object.
Determine:	This menu choice would be selected by the user to determine the value of a design parameters. This function is called when this menu choice is selected. This function sends a message to the

	ET object that this Transactor belongs, to determine the value of the parameter.
Increase:	This function is called when the user needs to increase the value of a parameter. The value of the parameter would be increased by a percentage amount specified by the user. When the user selects the menu choice for this, this function displays a prompt window where the user inputs the amount the parameters value needs to be increased by.
Decrease:	This function is similar to the previous function except that the user would decrease the value of the parameter.

The Transact_Popup Class

This class is responsible for the creation of the user interfaces for the Transactor class object. This class has functions to create the pop-up menus and the pull-down menus necessary for the interaction of the transact mode ET with the user. A pop-up menu is created which is displayed when the menu item for the ET is selected from the main window. This pop-up menu has several menu items which the user can choose to query the ET. Some of these menu items have pull-down menus which have further menu choices depending on the menu item on the main pop-up menu that was selected.

Public Functions

Create_Popup:	This function creates a pop-up menu for the ET operating in the transact mode. Menu items are created for the user to query the ET
Create_Pulldown:	Pull-down menus are created in this function. The pull-down menus will have further menu items depending on the menu choice on the main pop-up menu.
Create_Increase:	Creates a window where the user specifies the percentage amount by which the value of the parameter has to be increased.
Create_Decrease:	Creates a window where the user specifies the percentage amount by which the value of the parameter has to be decreased.

The Teacher Class

This class has been designed to lead the user of the expert system through a design process. This class has functions to access the knowledge base of the ET and the Parameters that belong to the ET. Using these the teach mode ET provides methods and rules to the user to manipulate the parameters to arrive at the final design solution. This class has been designed to use the knowledge that has been captured from experienced designers by the learn mode to teach novice users about the domain of design. Thus, this mode provides new users with the experience and knowledge to perform a design in a domain that they are not familiar with.

The teach mode ET starts with the initial parameters of the design. It displays to the user these parameters and the intermediate parameters that will be calculated from these parameters. For this it accesses the rules in the knowledge base of the ET. With the knowledge of the initial parameters and the rules the system would exactly know which of the intermediate parameters to calculate. Thus it proceeds through the hierarchy of parameters until the final parameters are reached. All through this process the ET lets the user know what parameters are needed to calculate the intermediate and final parameters. The equation rules, control rules and any other rules regarding the parameters in use or those being calculated are displayed to the user. Thus the ET shows the steps that are required for the calculation of the final design parameters from the starting parameters.

Public Functions

Constructor:	The constructor for this class creates a menu item for the ET on the main window of the application. Selection of this menu item would pop-up the menu for the ET. User interface functions are inherited from the Teach_Popup class.
Get_Name:	Returns the name of the ET that an object of this class would belong to.
Lead_Thru:	This function is called when the user wants the teach mode ET to lead through a design process. This function sends a message to the ET to use its inference engine to lead it through a design process.

Set_Active: This function would activate the Teacher object and lead the user through a design process.

Set_Inactive: This function is called when the user selects the menu choice to deactivate the Teacher object. This sets the Teacher object inactive without destroying the object itself.

The Teach_Popup Class

This class is responsible for the creation of the user interfaces for the ET operating in the teach mode. This class creates a pop-up menu for the teach mode object which is displayed upon the selection of the menu item for the ET object on the main screen. An ET operating in this mode would have menu items for activating or deactivating the teach mode ET. Activating the teach mode would prompt the user to specify the initial parameters of the design and would lead the user to the final design parameters. Deactivating the teach mode would make the stop the interaction with the user but would not delete this object.

Public Functions

Create_Popup: Creates a pop-up menu for the ET operating in this mode. Menu items are created for activating or deactivating the teach mode.

The Student Class

This class is responsible for capturing new knowledge regarding the domain of design from experienced designer and users of the expert system. The knowledge thus captured is stored in the knowledge base of the ETs and could be used for aiding the designers in further designs. There are two modes of learning that the Student mode ET operates in. The first is the voluntary learning process. In this case the ET queries the user of the system for the reasons for certain decisions that are taken during the design process. When the user replies to the ET, it stores these reasons in the form of rules in its knowledge. The ET would also be capable of learning from its own decisions regarding the design. This process would be coordinated by the Expert_Manager object. As mentioned earlier the ET operating in the Consultant or the Transactor mode are replying to the user queries constantly. The replies are made from the search through the knowledge base of the ET. During this process, the ET might come across relationships or control rules that are not in the knowledge base of the system but are obtained as a result of the manipulation of the parameters and inference of the knowledge. If these decisions are sent to the user and they are not in the knowledge base, the Expert_Manager object sends a message to the ET in the Student mode to add the reply as a rule in the knowledge base of the ET.

The second mode of learning is the forced learning. In this process, the user is allowed to enter rules directly into the knowledge base of the ET. When the Student ET is set to this mode, it pops up a window on the screen for the user to enter the rule. This is then added to the rule base of the system.

Public Functions

Constructor:	The constructor for this class creates an object and the user interfaces for the ET operating in this mode. The user interfaces for the ET are inherited from the Learn_Popup class.
Get_Name:	This function returns the name of the ET that it belongs to.
Force_Learn:	This function is called when the user selects the menu choice to forcibly make the Student "learn". This function displays a pop-up window for the user to enter the rule. This functions then sends the rule to the ET object that it belongs, to add to its knowledge.
Voluntary_Learn:	This function sets the mode of operation of this object to voluntarily learn from the design process that is being carried on. It pops up a window for the user to enter the reason for a decision that was taken regarding the design. When the user enters the reason, this object sends a message to the ET object to add this as a rule to its knowledge-base.

The Learn_Popup Class

This class creates the user interface functions necessary for the interaction of the user and the ET operating in this mode. A pop-up menu is created which is displayed when the menu item for the ET is selected on the screen. The menu items for the ET sets the Student mode either to the voluntary learn mode or the forced learn mode.

Public Functions

Create_Popup:	Creates the pop-up menus for the ET object operating in this mode. Menu items for setting the learn mode voluntary or forced are also created.
Create_Rule_Popup:	Creates a pop-up window for the user to enter the rule during the forced learning process.
Create_Voluntary_Popup:	Creates a pop-up for the user to enter a rule during a voluntary learning process.

9. Knowledge Base and Inference Engine

The Rule Class

This chapter discusses the Rule class and the Inference Engine classes. Both these classes belong to the ET and provide it with the knowledge and the methods to aid the designer in the design process. As mentioned earlier, the knowledge base of the expert would consist of rules in the form of parametric equations, control equations, constraints, heuristics and empirical relationships. All these different types of rules would be objects of the rule class. This chapter initially discusses the requirements for the rule class. The rule class should be flexible enough to accommodate all the above mentioned types of rules as an instance. The chapter further discusses the actual design of the rule class along with the functions that should be a part of the rule class design. The final section of the chapter discusses the requirements and the design of the inference engine class. An object of this class would search the knowledge of the expert using specific search strategies to reply to the users queries. Search strategies like forward and backward chaining would be built into this class for that purpose.

The ET object which represents the expert in the expert system should have knowledge regarding the domain of the design. The knowledge of the expert should specifically suit parametric design. Parametric design performed by current CAD systems are procedural in nature wherein the parameters are manipulated to obtain the final parameters of design. The design system starts with the initial parameters and uses these to calculate the intermediate parameters and from these intermediate parameters calculate the final design parameters. This process is procedural in the sense that the calculations of the parameters follows a certain order. The user enters the initial parameters and the system provides the user with the final design parameters. In this case the user has no control over the intermediate and the final parameters. The user would not be able to change the values of these and easily observe the changes brought about to the other parameters by this change.

Figure 12 shows the structure of the rule class. The base class for the rules would have the constructor and the common data and methods for the rule class. The other derived rule classes inherit the data and the methods from this Rule class. The derived classes are the Data_Rule class, the Parametric_Equation class, Control_Rule class, Heuristic_Rule class and the Constraint_Rule class. Any rule that needs to be entered into the knowledge-base of the ET would be made an instance of one of these rule classes. The programmer would specify at the time of creation of the ET class which rule class needs to be instanced to create a rule that would be stored in the knowledge-base of the ET. The Data_Rule class would contain rules in the form of tables of values, values from a graph, etc. Rules regarding the domain of design that obtained through experience or that have come about through empirical methods would be instances of this class. An example of this rule would be the relationship between parameters that specifies that if the value of parameter A increases, the value of parameter B decreases.

Calculation of the parameters are done by the parametric equations. Parametric equations are those equations that calculate the value of a parameter from the values of other parameters. The parametric equations that are used to calculate the values of the parameters are governed by the control rules. For example a control rule might use a particular equation to calculate the value of a parameter based on the value of another parameter. A typical control rule might be represented using an if-then-else statement in a procedural program. Constraints might be in the form of limitation to the maximum and the minimum value of a parameter based on constraints of design. For example the stress parameter in the design might have a maximum limiting value depending upon the material being used or the length of a structural member might have a maximum and a minimum value depending on the geometric constraints. Rules of thumb might also need to be entered as knowledge to the ET. These might be in the form of relationships between parameters or storing a particular value to a parameter depending upon the conditions for design.

As mentioned earlier, if the user wants to use a function of another module, for example, the analysis module, as a part of the knowledge of any ET, flexibility should be provided such that this function could be declared as a rule. This rule, shown as the Interface rule, would be similar to the equation rule in the sense that the inference engine would be able to fire this rule.

The rule class should be able to accommodate all the above mentioned forms of rules. Most of these rules would use the parameters for calculating or defining relationships. So, these rules should be able to access the parameters and change the values of the parameters. Rule objects would be used by the inference engine to arrive at decisions regarding the design. The rule objects should allow easy access to the inference engine. This rule class has been further designed and implemented by Scott Angster [Angs93].

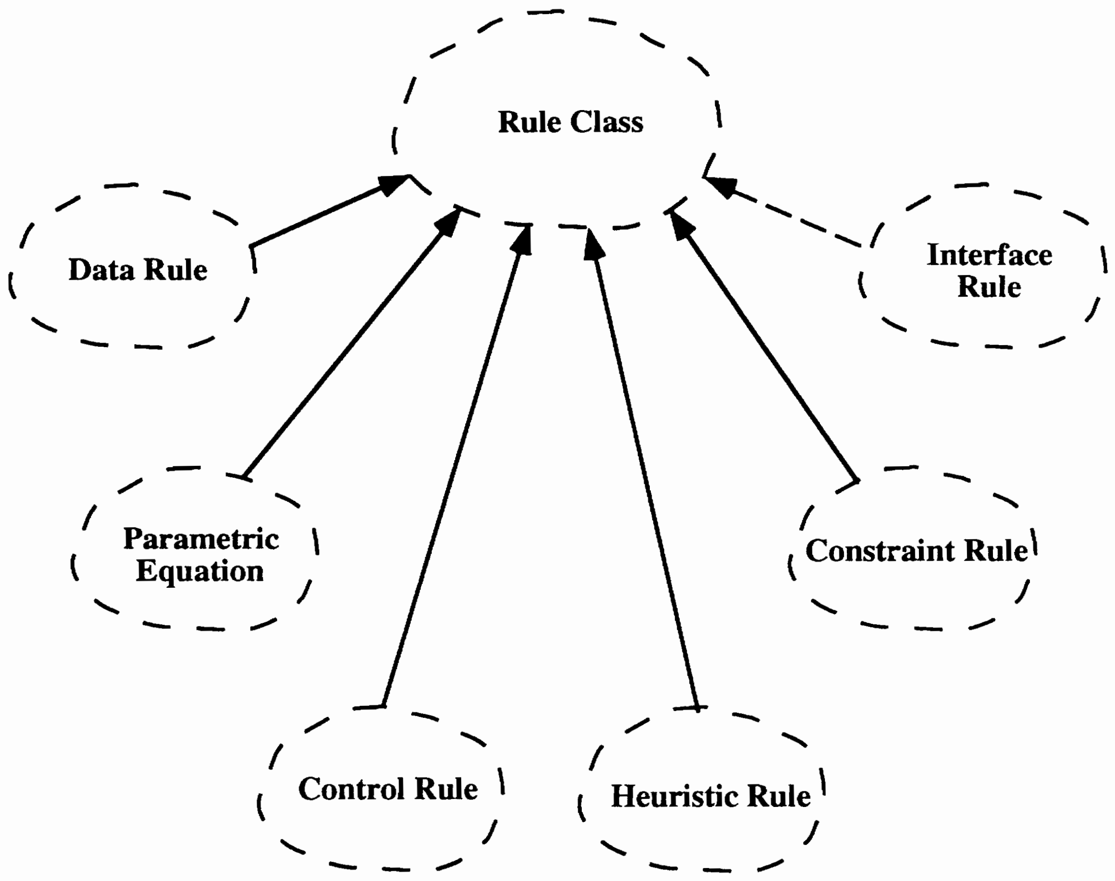


Figure 12 The Rule Class

Public Functions

Constructor: The constructor for this class would have methods to create an object of one of the rule classes mentioned above. This function would then add the rule object created to the linked list of rules maintained in the ET. This function would also prepare the rule for firing.

The Inference Engine Class

This class would be responsible for the chaining and searching strategies through the knowledge-base of the ET. The chaining process typically used in an inference engine would be the forward and the backward chaining. Forward chaining is a data-driven reasoning method while backward chaining is a goal-reasoning method. If a rule specifies IF A THEN B, the forward and the backward chaining processes differ in the method of firing this rule. In a forward chaining process the left-hand side of the equation, that is the data is matched with all the data of all the available rules and the rules are chosen to fired based on this data. Actions are taken consequent to the results of the rules that are fired. Thus in this case the data of A is selected from a previous rule and B is determined. In the case of backward chaining, B is the goal and A is the data that needs to be verified. The facts that establish A are searched for and fired. If such facts cannot be found in the rule base, then A is treated as a sub-goal and the chaining continues [Dym91]

The inference engine class designed would support the forward and backward chaining processes. This class would have functions that can perform both data-driven and goal driven searches. The search strategies and the firing of the rules is tailored for the specific needs of parametric design. An object of this class would belong to the ET object and the functions in these classes would be used for the determination of the final parameters of design. The inference engine object would directly act on the rules available in the knowledge-base of the of the ET. For this purpose the inference engine has access to the linked list of rule objects that is maintained by the ET. To determine the values of the parameters, this object would need values of some of the parameters that belong to the other ETs. For this purpose, the inference engine sends a message through the ET to the Expert_Manager to return the value of a parameter belonging to another ET.

Public Functions

Determine_Parameter: This function determines the value of a parameter. For this purpose, it uses the equations in the Parametric_Equation objects. The equation that is used is dictated by a Control_Rule wherever applicable.

Update_All: This function updates the values of all the parameters. In this case, the user defined values for the input parameters are used and the values of all the intermediate and final parameters are determined.

Validate:	This function would typically be used by the Observer object. This function is called if the Observer receives a message from the Expert_Manager that the value of a certain parameter has been changed. This function then checks to see if the value of that particular parameter violated any of the constraint rules. It then displays warnings about violations and suggests solutions to the violation.
Interpret_Heuristics:	This function would be used to parse a heuristic rule that is normally in the form of a string. To fire such a rule, it needs to be parsed which is done when this function is called.
Activate_Rules:	A control rule would normally determine the parametric equation that would be used for the calculation of a certain parameter. Based on the result of firing of the control rule, a particular equation that is used for the calculation of the parameter is activated. If there are other equations that determine the value of the same parameter, they are deactivated.
Deactivate_Rules:	This function is called to deactivate Parametric_Equation rules that would not be used for the calculation of a certain parameter under a certain condition dictated by a control rule.
Check_Control:	This function fires the control rules and determines which of the parametric equation rules need to be activated and deactivated.
Select_From_Data:	This function selects data from the data rules which normally holds data values from tables, graphs, charts, etc.

10. Example of an Expert System Session

The example considered for an expert system session is the parametric design of a two gear drive train. The drive train consists of two gears mounted on separate shafts with keys. One of the shafts is an input shaft and power is transmitted through the gears to the output shaft. For a parametric design of the system, an input parameter might be the power that needs to be transmitted. The design of this system involves determination of the various parameters that define this drive train. Some of the parameters for this system are shown in figure 13 and the rest of the parameters are listed below.

N1, N2 :	No of teeth on the input and output gears
KW1, KW2:	Width of keys
KL1, KL2:	Length of keys
KH1, KH2:	Height of the keys

The design process involved in this case could be a multi-disciplinary, parametric design. The different domains involved for this system are (a) kinematics which deals with the geometry and the angular velocities of the system and (b) strength which deals with the power transmitted and the strength considerations of keys, etc. To make the design complete from the point of view of manufacturing of the various components, a manufacturing domain, which deals with the issues of manufacturing is also introduced.

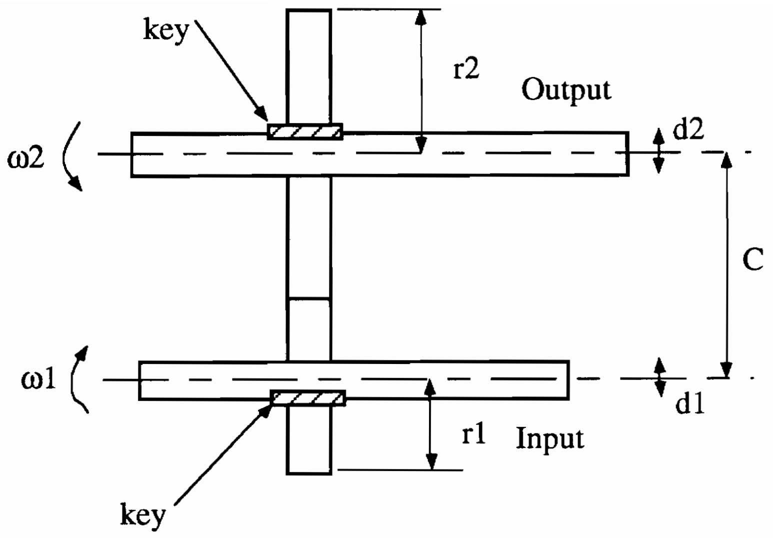


Figure 13 Sketch of the Gear Train Design

The various ETs that would be created for the expert system would be experts in the above mentioned domains. The programmer creating the expert system would provide the ET with the necessary knowledge and information regarding the domain of design that they are experts in.

The ET that deals with the kinematics domain, the kinematics expert, deals with the equations, constraints, etc. affecting the various kinematics parameters. This expert would "own" the following parameters that directly deal with the kinematics domain of the design. These parameters are the angular velocities, radii of the gears, number of teeth in each gear, ratios of angular velocities, center distance between the gear shafts and the module of the gear. The ET, as a part of its knowledge, would contain rules in the form of equations, constraints, rules of thumb, etc. which deal with the determination and manipulation of the parameters that belong to this ET.

The strength expert would deal with the rules regarding the strength considerations of design. Parameters that belong to this expert would primarily affect the design in this particular domain. The power transmitted would probably an input parameter in this domain. Depending on the power transmission through the drive train, the parameters that determine the size of the key and the key way are obtained. The reason for attaching the parameters such as the size of the keys and key ways to this expert is because these parameters would be directly affected by the maximum power transmission possible. The width of the gear would also determined by the power transmitted and thus would belong to this expert. This expert would have the rules to determine these parameters as a part of its knowledge base.

The manufacturing expert in this case would not own any of the design parameters. This is because none of parameters used in this domain need to be determined directly from the knowledge of this expert. This expert would contain rules that directly affect the manufacturability of the components designed. For example it would have rules that have the sizes of the cutters available to machine the key way on the shafts. If the calculation of the width of the key way yields a dimension that is impossible to manufacture, this expert would warn the user about it. This expert could also have rules that specify the number of teeth that could be manufactured on a blank of a particular diameter and the standard modules for the gears. Thus, if the user or the kinematics expert changes some of the values in the kinematics expert in a manner which is inconsistent from the manufacturing point of view, this expert would able to warn the user.

Thus, with all the parameters and rules input to the ETs of this system, a user of the expert system would be able to interact with the ETs and obtain replies to various queries. The user would be able to change the value of the parameters and observe the effect of these changes on other parameters of the system. The determination and the manipulation of the parameters would be performed by the rules that belong to the ETs.

The expert would be able to interact with the user in different modes of operation. The modes in which the ETs interact with the user is at the discretion of the user. The programmer of the expert system creates the ETs but it is the user who decides which of the modes of the experts the user wants to interact with. For example in this case, the user would most probably want the manufacturing expert to act in the observe mode since this ET does not "own" any parameters and does not have any rules to determine any of the parameters. This expert just has rules that would be constraints, rules of thumb and standard values. Thus the user would want this expert to observe the design and to provide suggestions and warnings regarding the parameters from a manufacturing point of view. This ET should be able to obtain the values of the parameters that belong to other ETs. These values would then be compared to the values and constraints available in the knowledge of this ET. Thus, based on this comparison, the ET would be able to provide suggestions and warnings to the user. For example, this ET would have a rule that has the dimensions of gears those are already being manufactured in the particular company. If the user comes up with a design of a gear whose dimensions are close to the gears those are already being manufactured, this expert would be able to suggest to the user to change the dimensions, if possible, to match the one being already manufactured. This expert would also have the standard sizes of the components and the standard sizes of machine tools that are available. For example if the strength ET determines the values of the parameters that specify the size of the keys, this expert would then output to the user the dimensions of the closest standard key that is available. For this expert would contain the values of standard dimensions of keys as rules in its knowledge-base. If the user comes up with a dimension of a component that is difficult to manufacture, this expert would warn the user about this.

If the kinematics expert is operating in the transact mode, this expert would provide a single reply to all the users queries. The user would be able to query this expert with the help of the pop-up menus that belong to this expert. This expert contains rules that deal directly with the kinematics domain of the design. For example one of the equations would be the equation relating the ratio of the number of teeth to the ratio of the input to output angular velocities. If the user changes the input angular velocity and wants to see how this has affected the number of teeth in the gears, the user would use the "Determine" menu of the ET. This would calculate the value of the parameter from the equation that is a part of its knowledge-base. Thus, this ET would be able to reply to all the queries of the user by searching through its knowledge. The user would be able to determine the values of the various parameters that belong to this ET and observe the changes to other ETs. The user would also be able to increase or decrease the value of a parameter and observe the changes to any other parameter to determine the relationship between the parameters. This is done using the equations in the knowledge of the ET. This would be a part of the chaining process of the rules in the ETs.

If the strength expert is operating in the consult mode, this expert would be able to provide multiple replies to a single query of the user. For example, if the user changes the power transmitted through the gear train, the expert would calculate all the parameters that are affected by this change. The expert would then let the expert know values of the parameters that were calculated and would let the user know the size of the keys needed to transmit the power input by the user. It would also let the user know the width of the gear that is necessary to transmit this power.

The parameters of design would belong to various experts depending on the domain of the design. Given below are the list of the parameters that belong to the various experts.

<u>Kinematics Expert:</u>

No of teeth in the gears:	N1, N2
Angular velocities:	$\omega 1$, $\omega 2$
Ratio of angular velocities	$\omega 1/\omega 2$
Radii of gears	r1, r2
Center distance between the shafts	C
Module	m

Strength Expert:

Length of keys:	kl1, kl2
Width of keys:	kw1, kw2
Height of keys:	kh1, kh2
Width of gears:	W
Power transmitted:	P
Diameter of the shafts:	d1, d2

The knowledge of an ET would contain among other things equations to calculate the values of some parameters. These equations are a part of the knowledge of a particular expert depending on the parameter that is being calculated. For example the equation to calculate the radius of the gear would be a part of the knowledge of the kinematics expert. The equations that are used in this for the calculation of the parameters are listed below. Some of the parameters could be calculated using several different equations.

Kinematics Expert:

N1:

$$N1 = \frac{\omega 2}{\omega 1} N2 \qquad - 1$$

$$N1 = \frac{2r1}{m} \qquad - 2$$

N2

$$N2 = \frac{\omega 1}{\omega 2} N1 \qquad - 3$$

$$N2 = \frac{2r2}{m} \qquad - 4$$

ω1: $\omega 1 = \frac{N2}{N1}\omega 2$ - 5

$\omega 1 = \frac{r2}{r1}\omega 2$ - 6

ω2: $\omega 2 = \frac{N1}{N2}\omega 1$ - 7

$\omega 2 = \frac{r1}{r2}\omega 1$ - 8

r1: $r1 = \frac{C}{\left(1 + \frac{\omega 1}{\omega 2}\right)}$ - 9

$r1 = \frac{\omega 2}{\omega 1} r2$ - 10

$r1 = C - r2$ - 11

$r1 = \frac{N1 * m}{2}$ - 12

r2: $r2 = \frac{C}{\left(1 + \frac{\omega 2}{\omega 1}\right)}$ - 13

$r2 = \frac{\omega 1}{\omega 2} r1$ - 14

$r2 = C - r1$ - 15

$r2 = \frac{N2 * m}{2}$ - 16

C: $C = r1 + r2$ - 17

A constraint equation for this expert could be the equation for the minimum number of teeth for meshing of the gears without interference. If the calculation of the number of teeth yields a value that is lesser than this value, this expert would warn the user about it.

N2min: $$N2_{\min} = \frac{34.2k^2 - N1}{2 * N1 - 34.2k}$$

Strength Expert:

kl: $kl = W$

kw: $kw = f(P, \omega, d)$

T: $T = f(P, d)$

W: $W = f(P)$

These equations would be a part of the knowledge of the expert system. The expert would use these equations for the calculation of the parameters and reply to the users queries. Some of the parameters have more than one equation that determine the value. In these cases the programmer should specify priorities for the order in which the equations will be calculated. For example for the calculation of the parameter r1, the equation with the highest priority would be equation 9 followed by equations 10, 11 and 12. The system would then try to calculate the value of parameter r1 from 9. If this is not possible in a case where the value of C is not specified and value of r2 is known, the system would use equation 10 to calculate the value of the parameter. Thus the determination of the value of the parameter would be in the order of the priority of the equations. This priority itself would be a rule for the expert.

If the user specifies the values of the ratio of the angular velocities and the center distance between the two shafts, the system would first calculate the values of r1 and r2 from equations 9 and 13. Knowing the values of r1 and r2 then the expert would calculate the values of w1 and w2. From these values, parameters N1 and N2 are determined. Thus, as the user inputs the values of the parameters, the expert would try to determine the values of all the other parameters that are owned by it. If the user now changes the value of one of the parameters, the values of the other parameters that are affected are calculated. For example, if the user now changes the value of C, the center distance, the expert would automatically update the values of the parameters r1 and r2 which are directly obtained from this parameter. Once the value of r1 and r2 are changed, the expert updates the values of w1 and w2. The values of N1 and N2 are also updated. Thus, changing one of the values of the parameters starts a process of updating values of other parameters affected by this change and is continued until the values of all the parameters are updated. Thus, the user would be able to observe the changes that are brought about by changing the values of parameters and would be able to determine the relationships between the parameters.

Once these parameters are calculated, the manufacturing expert would be able to suggest changes based on the standard gears available. Similarly with the strength expert, once the power transmitted and the width of the gear is known, the expert would be able to calculate the size of the keys needed. Based on this, the manufacturing expert would be able to suggest a key of standard size from the rules in the knowledge-base.

Figure 14 shows the flow of control in a sample session. The user changes the value of C in the kinematics ET. Based on this change the ET calculate the values of the other parameters affected by this change. For this purpose the ET uses the inference engine object. This inference engine object uses the various rules available in the knowledge-

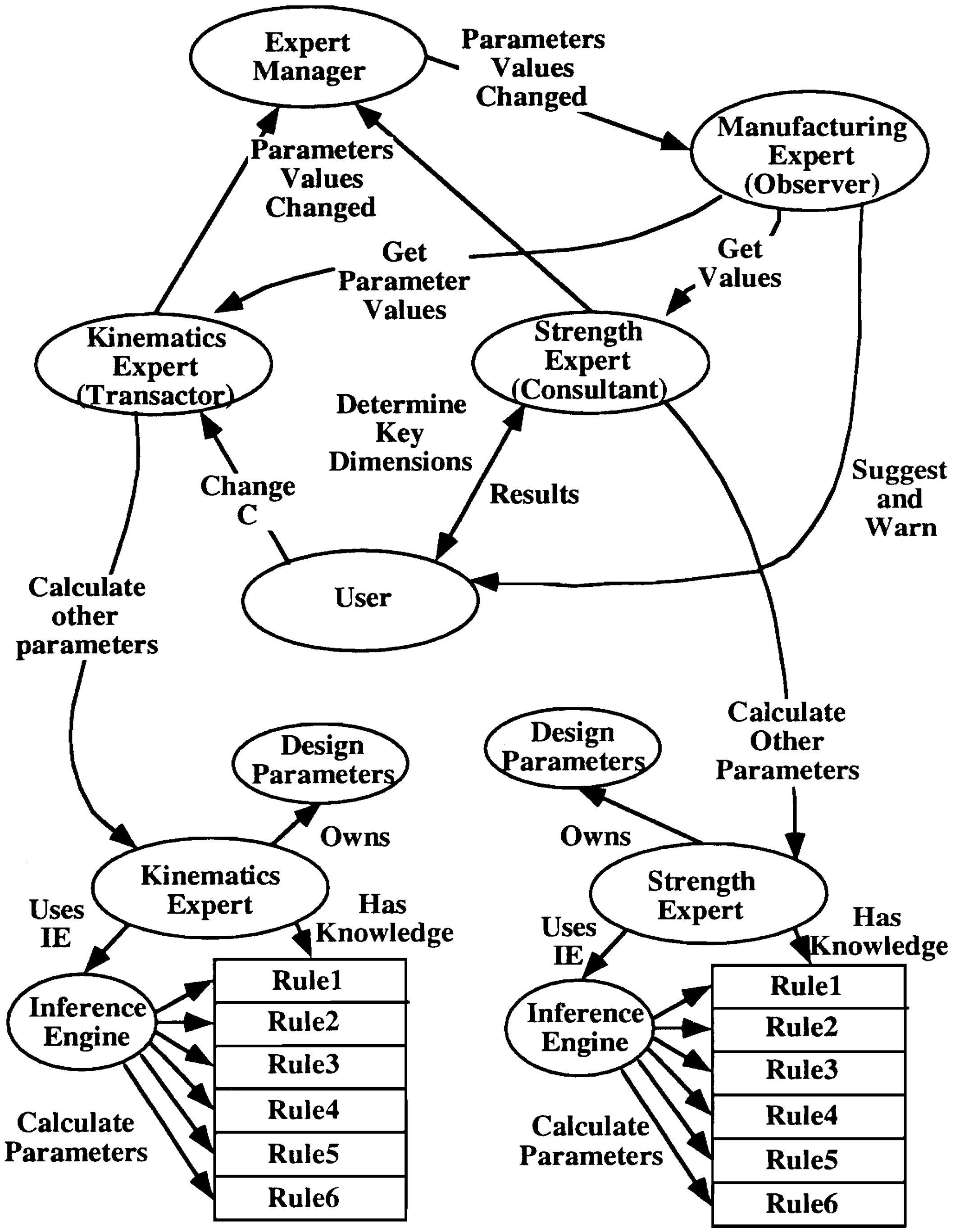


Figure 14 Flow Control for a Sample Session

base to calculate the values of all the other parameters. The kinematics ET then sends a message to the Expert_Manager object specifying that the values of its parameters have changed. Upon receiving this message, the Expert_Manager object sends a message to the manufacturing expert operating in the observe mode. This ET then accesses its rule base to see if any of the rules are violated to warn the user. For this purpose, it would need the values of the parameters in the other ET which it retrieves by sending a message to the appropriate ET. For example, the kinematics ET, based on the value of C input by the user, would have come up with a value for the module which is not standard. This ET would have the values of the standard values of the modules for gears. Based on this value, it will warn the user and suggest a new value for the module of the gear.

The strength ET in this example is operating in the consult mode. If the user changes the value of the power transmitted, the ET calculates the values of all the other parameters and outputs the values of these parameters to the user. The ET uses the inference engine object which accesses the rules in the knowledge-base for the calculation of the parameters. When these values are changed, the ET sends a message to the Expert_Manager object which in turn sends a message to the manufacturing ET object. The manufacturing ET, as earlier, checks to see if any of its rules are violated, which it then warns the user. Changing the value of the power transmitted would change the dimensions of the key being used for the design. The strength expert would send a message to the Expert_Manager that the dimensions of the keys have changed which in turn is sent to the manufacturing expert. This expert has dimensions of standard keys and suggests to the user that the dimensions of the keys might be changed to be consistent with the standard.

Thus the expert system that is developed from this framework would aid the designer during the design process by letting the designer observe the changes brought to the design by changing the value of a particular parameter. It would also provide the designer with suggestions and warnings regarding the design so that costly changes are avoided during the latter part of the design process. The expert system would also let the designer deal with experts in different domains simulating a real-world design process.

11. Implementation and Results

A prototype of the object-oriented framework designed was implemented using C++. C++ is a language that supports object-oriented programming. The platform used for the development of this framework was IBM RS/6000, using the AIX operating system. The C++ compiler used for the implementation was xlC. The user interfaces for the framework were implemented using OSF/Motif, an application programming environment using X Windows environment. The version of Motif used was release 1.1 and the X11 Release 5 version of the X windows was used.

C++ is a language gaining wide acceptance and popularity in the object-oriented programming world. It is a language that supports all the features of object-oriented design such as encapsulation, inheritance, polymorphism, etc. One of the main objectives of the thesis was to develop this framework using a language that is used and understood by programmers and engineers. This necessitated that the language used for the implementation should be a language that is in everyday use by the programmers of CAD systems. C++ has gained popularity among the programmers and is widely being used to develop CAD applications. The expert system developed from this framework would need to be integrated with the overall CAD system and an implementation of the framework using C++ would make the task of integration easier.

The user interfaces for the framework were implemented using OSF/Motif. Motif is based on X Window System. It is a graphical interface programming toolkit. It provides the user interface objects as widgets and provides these to the user through a library. This library includes several objects such as windows, push buttons, dialog boxes, lists, text windows, scrolled windows, etc. that can be used to develop a user interface for any application. A programmer using this toolkit for the creation of such user interfaces has to call the functions provided in the library to create these objects. Parameters are passed into these functions which affect the appearance and the location of these objects when they are displayed on the screen.

Motif was used for the implementation of this object-oriented framework because it provides a library of user interface objects which are easy to use and implement. Since it is based on X Window System, Motif is portable across all the platforms that run the X window system. The user interfaces have been designed to be as separate classes in this framework. Other classes requiring these interfaces inherit from these classes. This is to aid in the implementation and future expansion of the framework wherein only these classes need to altered or changed to accommodate any new graphical user interface toolkit.

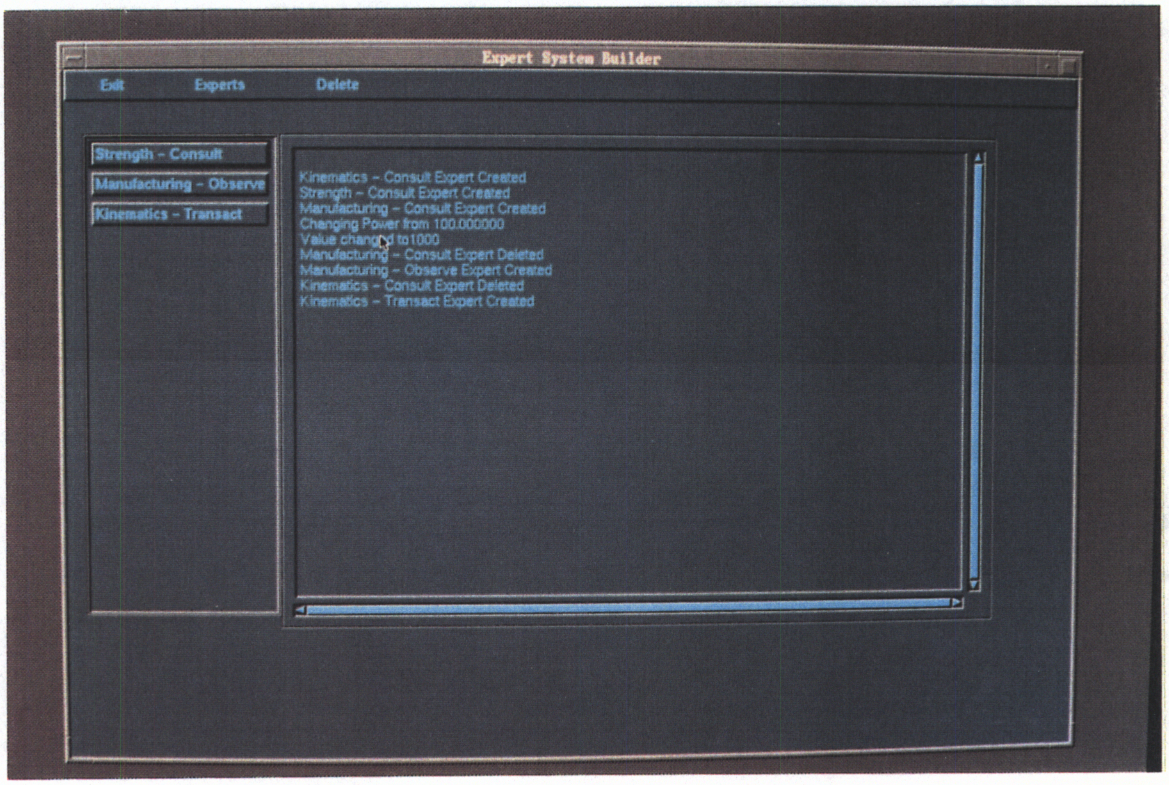
Since one of the objectives of this thesis was to only develop a prototype of the framework that was designed, all the classes that were designed were not implemented. The following paragraphs detail the classes that were implemented and to what extent these classes were implemented.

The classes that were fully implemented based on the design were the Session_Manager class, the Session_Interface class, the Expert_Manager class, the Delete_Menu class, the Expert_Setup class, the ET class, the Parameter class, the Transactor class, the Consultant class and the Observer class. The classes which were designed and were partially implemented include the Teacher class and the Student class. They have methods to create the class object but do not have any functions built into them for the designed interaction with the user. The user interface classes for these operating mode classes have been implemented. These classes typically contain the pop-up menus required for the interaction between the user and these operating mode objects. The Rule class was further designed and implemented by Angster [Angs93]. The Inference engine class was designed but was not implemented as a part of this thesis.

The implementation details are given below.

Session_Manager: As explained in the previous chapter this class creates an Expert_Manager object. Since this Expert_Manager object needs to be an external object, a new Expert_Manager object is created an set to be an external Expert_Manager object. All the variables that keep track of the ETs and the operating mode objects are initialized in this class. The functions for returning and accepting an Expert_Manager object have also been implemented into this class.

Session_Interface: This class has functions implemented which create all the user interface objects for the proper functioning of the expert system created from the framework. This class uses Motif function calls to create these objects. The Motif application is initialized and a main window is initially created. A menu bar is created as a child of the main window and the buttons for exit, setting up experts and deleting of experts are created in this class. A scroll-window where the push buttons for the ETs are placed is also created in this class. The dialog window where the replies of the experts are posted is also created in this class. Figure 15 shows the initial screen for the example problem described in the earlier chapter. Other objects such as frames for the



```
Kinematics - Consult Expert Created
Strength - Consult Expert Created
Manufacturing - Consult Expert Created
Changing Power from 100.000000
Value changed to1000
Manufacturing - Consult Expert Deleted
Manufacturing - Observe Expert Created
Kinematics - Consult Expert Deleted
Kinematics - Transact Expert Created
```

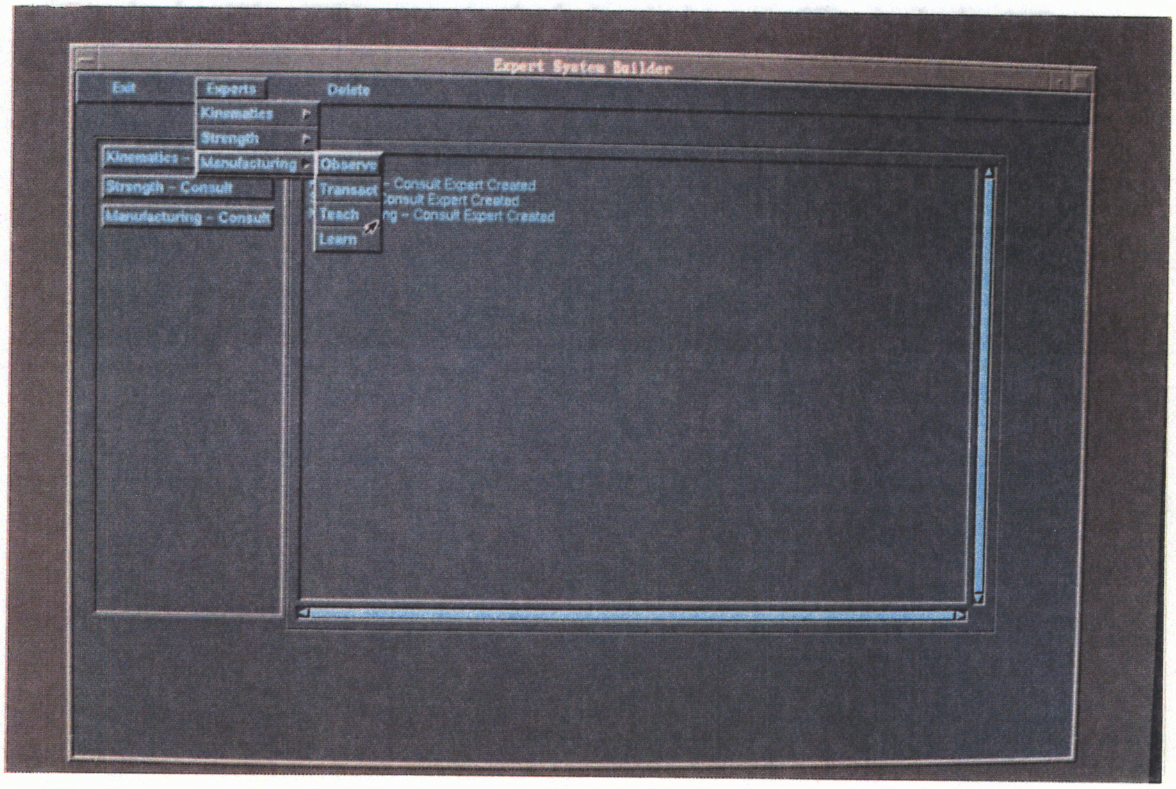
Figure 15 Main Screen of the Expert System

creation of other objects is also created in this class. Some of these user interface objects are defined as global objects and the reason for this is explained later in this chapter.

Expert_Manager: This class keeps track of all the ETs and their operating modes. All the functions described in the earlier chapter have been implemented for this class. Variables which keep track of the number of ETs and the operating modes have been implemented. This class maintains a linked list of all the ETs and the operating modes. A pointer to the first and last object in the linked list is always maintained by this class. Thus searches through the linked list use the first object pointer to start the search. Adding new objects to the linked list makes use of the pointer to the last object in the linked list.

Expert_Setup: This class displays the menus for the creation of a new operating mode for the ETs and is inherited by the Expert_Manager class. Motif function calls are made for the creation of a button on the menu bar in the main window for this purpose. This button has a pull-down menus attached to it. A push button is displayed on this pull-down menu for each existing ET. Each of these ET buttons have a second level of pull-down menus on which a push button is created for each of the operating mode of the ET. Push buttons are created only for those operating modes which are not yet active. Selecting these buttons sends a message to the Expert_Manager object to create the operating mode object in the ET. Figure 16 shows the menus for the creation of the operating modes of experts.

Delete_Menu: This class has functions which are similar to those in the Expert_Setup class except that the operating modes for which push buttons are created on the menus are those modes in which the ET is currently interacting with the user. Selecting these



- Consult Expert Created
onsult Expert Created
ng - Consult Expert Created

Figure 16 **Pull-down Menu for Creation of an Operating Mode of an Expert**

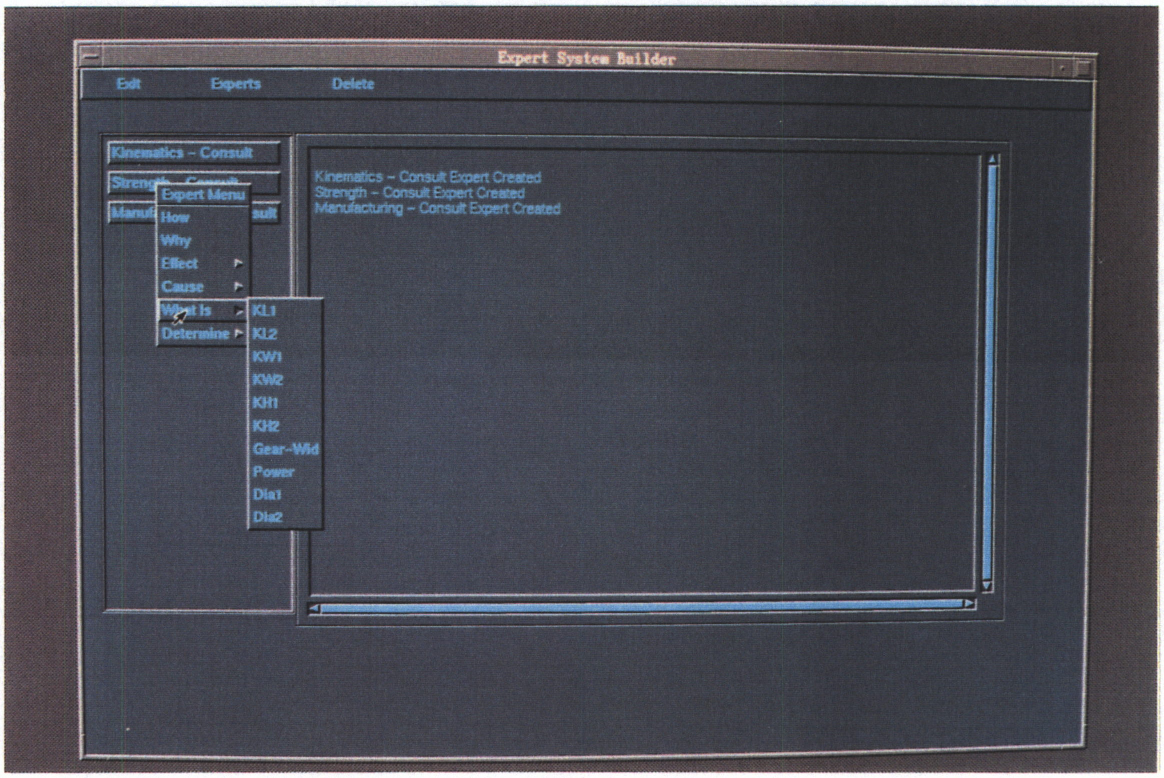
push buttons call functions in the Expert_Manager object to delete the operating mode object in the ET.

Parameter: This class was implemented completely as described in the earlier chapter. Figure 17 shows all the parameters that belong to the strength expert of the example problem described in the earlier chapter. Figure 18 shows all the parameters that belong to the kinematics expert in the example problem.

ET: This class represents the expert to be used in the expert system. An instance of this class is created by the programmer in the application. The expert is designed to interact with the user is different modes. Each of these modes is a class and the ET class has pointers to the different operating modes classes. When this class object is operating in a particular mode, a new object of the operating mode class is created and the pointer is set to that object. Each operating mode has a flag in the ET which specifies whether that mode is active or not. If the flag is set to active it means that the ET is operating in that particular mode. When the ET ceases to act in that particular mode, the operating mode object is deleted in this class and the flag is set to inactive.

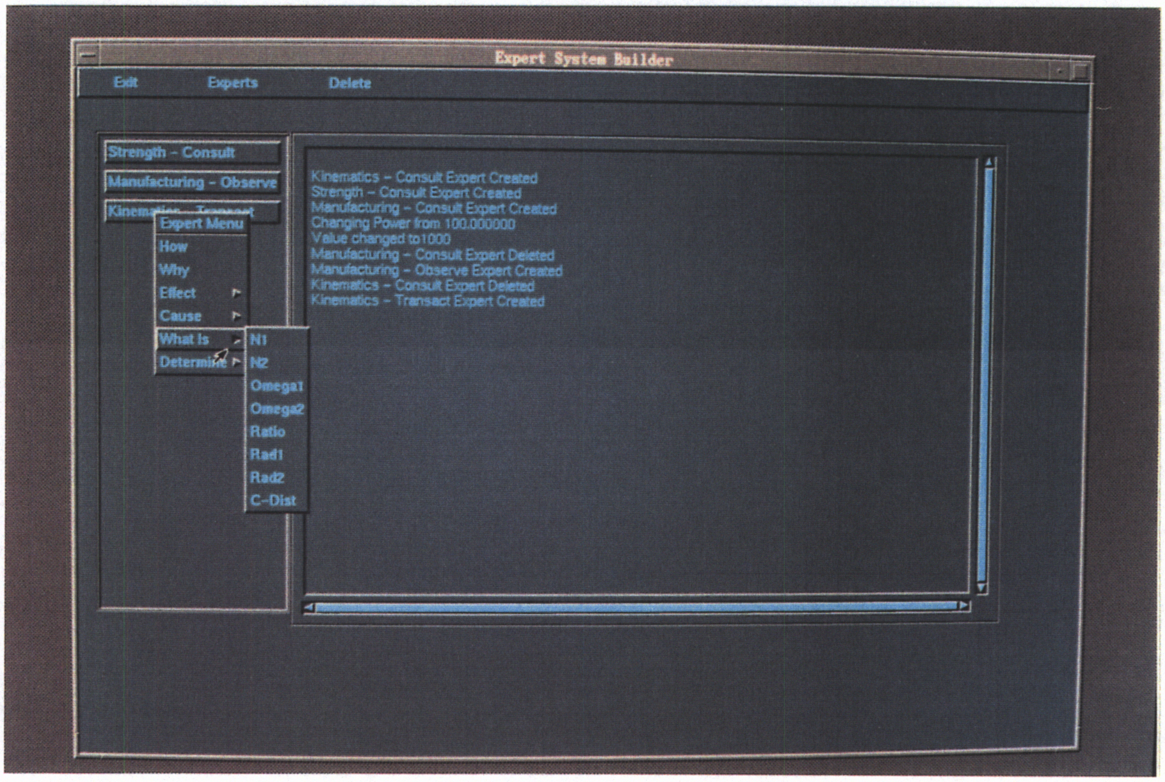
The knowledge base and the parameters that belong to an object of this class are defined during the creation of the object. The parameters and rules for the class may be stored in a file which is read in. As the parameters are being read into the ET object, objects of the parameter classes are created with the defined values. These parameters are stored in the form of a linked list. A pointer to the first parameter in the linked list is available to the ET class. Similarly as the rules are defined in objects of the rule classes are created. These rule objects make up the knowledge base of the ET.

Consultant: This class has not yet been fully implemented. Currently the constructor for this class just creates an object and the required user interfaces. The functions



Kinematics - Consult Expert Created
Strength - Consult Expert Created
Manufacturing - Consult Expert Created

Figure 17 **Parameters Belonging to the Strength Expert**



```
Kinematics - Consult Expert Created
Strength - Consult Expert Created
Manufacturing - Consult Expert Created
Changing Power from 100.000000
Value changed to1000
Manufacturing - Consult Expert Deleted
Manufacturing - Observe Expert Created
Kinematics - Consult Expert Deleted
Kinematics - Transact Expert Created
```

Figure 18 Parameters Belonging to the Kinematics Expert

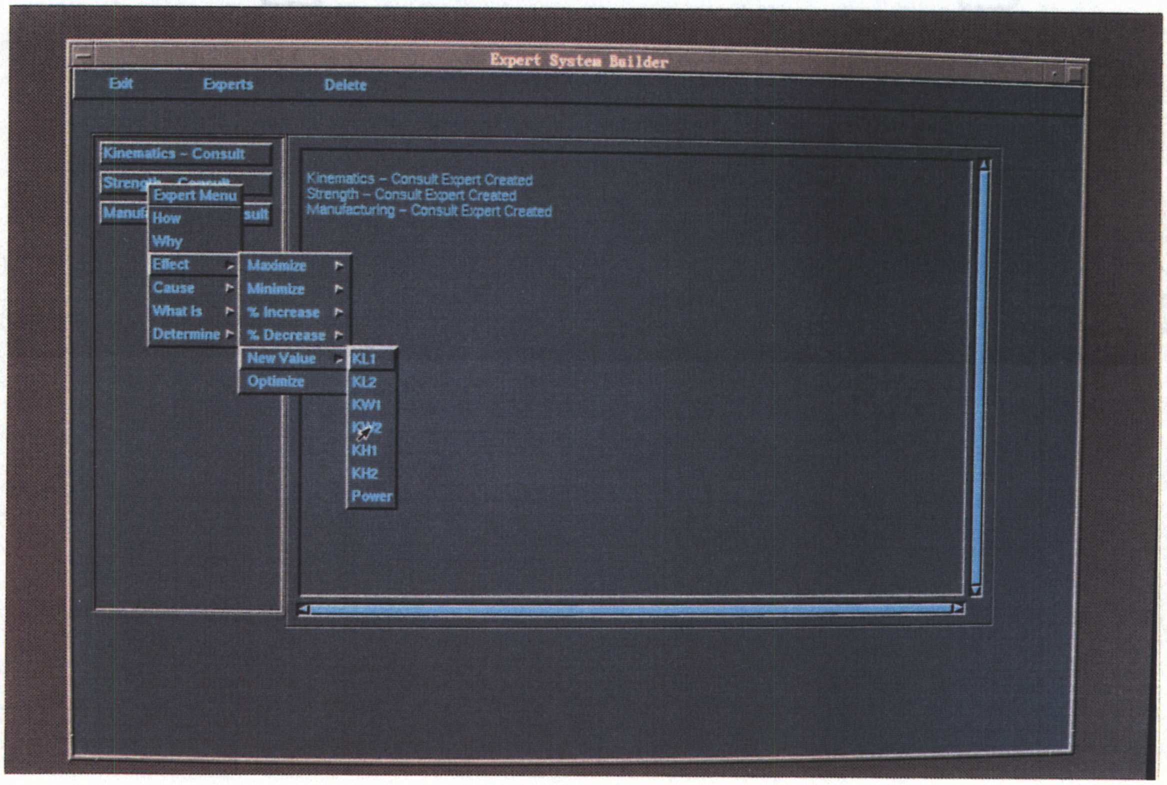
necessary for an object of this class to interact with the user have not been built into the class. This class carries pointers to the next and previous consult objects for the linked list of the consult object maintained in the Expert_Manager object. The Motif functions for this object are inherited from the Consult_Popup class.

Consult_Popup: This class has Motif functions for the pop-up menus that are displayed when the menus for the ET objects are selected. There are three levels of menus on the pop-ups for the experts in this mode. Figure 19 shows the three levels of menus for strength expert of the sample problem operating in the consult mode. The screen shown in this figure is displayed when the user wants to enter a new value of a parameter belonging to the strength expert. Figure 20 shows the pop-up menu which is displayed for the user to enter the new value of the parameter selected.

Observe: This class has also been partially implemented. The constructor for this class has been implemented and as with the other operating mode classes, the pop-up menus are inherited from the Observe_Popup class. The constructor of this class just creates an object of this class. The functions discussed in the earlier chapter have not been implemented.

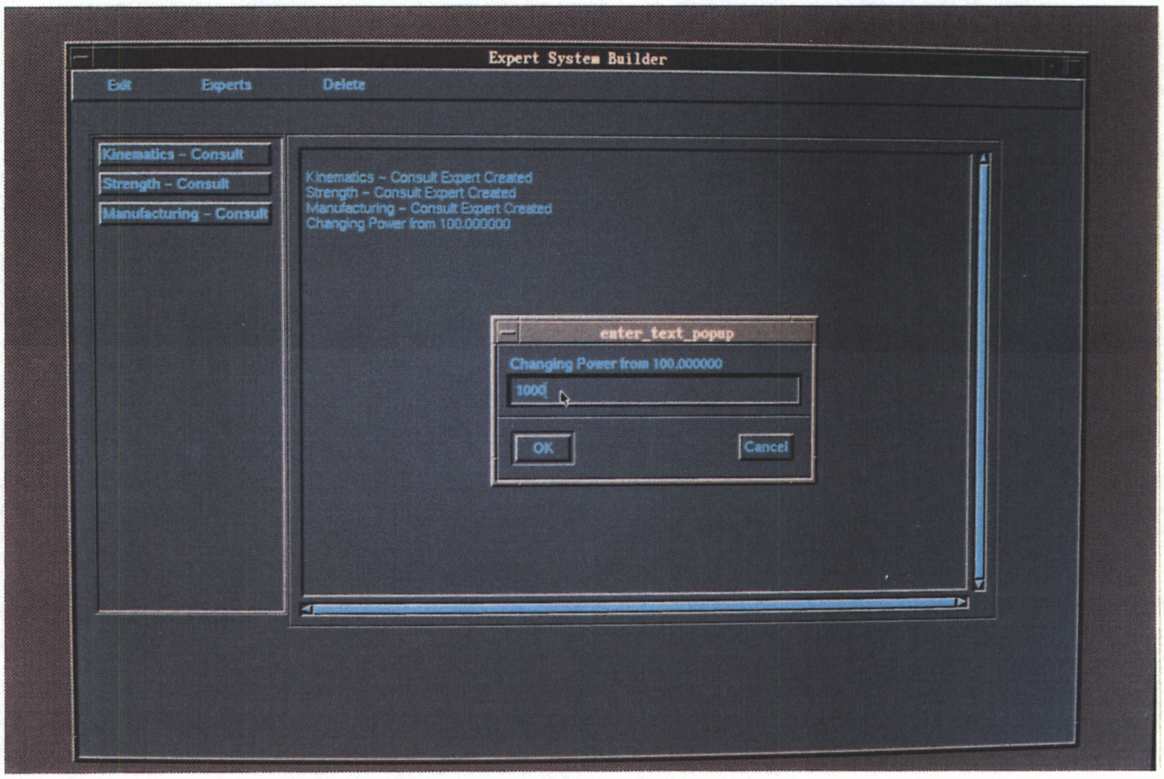
Observe_Popup: This class creates the pop-ups for the observe mode object. The pop-up for this class has buttons for setting the observe mode active or inactive. Figure 21 shows the menus for this ET.

Transact: The constructor of this class creates an object of this class. The functions designed for this class have not been implemented.



Kinematics - Consult Expert Created
Strength - Consult Expert Created
Manufacturing - Consult Expert Created

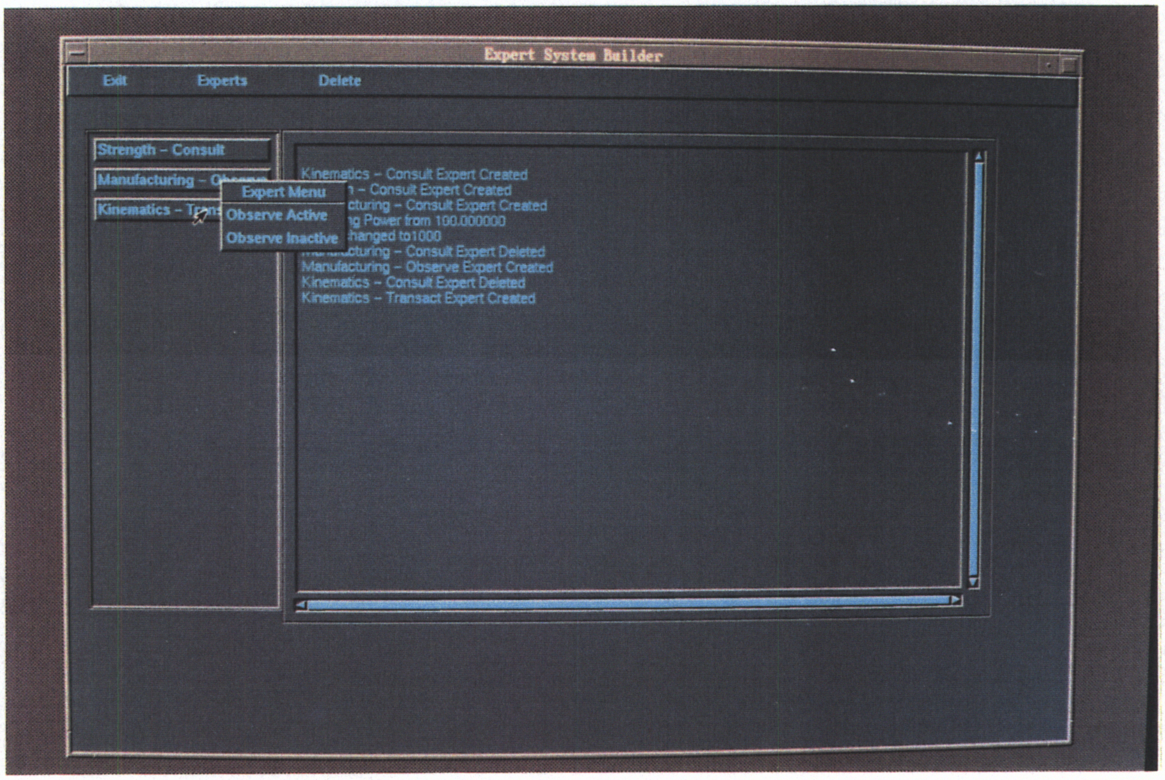
Figure 19 **Pulldown Menus for the Consult Operating Mode of an Expert**



Kinematics - Consult Expert Created
Strength - Consult Expert Created
Manufacturing - Consult Expert Created
Changing Power from 100.000000

Changing Power from 100.000000
1000

Figure 20 Pop-up Menu for Entering a New Value for a Parameter



```
Kinematics – Consult Expert Created
n – Consult Expert Created
cturing – Consult Expert Created
g Power from 100.000000
hanged to1000
cturing – Consult Expert Deleted
Manufacturing – Observe Expert Created
Kinematics – Consult Expert Deleted
Kinematics – Transact Expert Created
```

Figure 21 **Pop-up Menus for the Observe Mode of an Expert**

Transact_Popup: The pop-up menu for the Transact class is implemented in this class. It uses Motif functions for the implementation and the menu choices for this class are similar to the ones for the Consultant class.

Teacher: The constructor for this class has been implemented. The functions for this class have not been implemented.

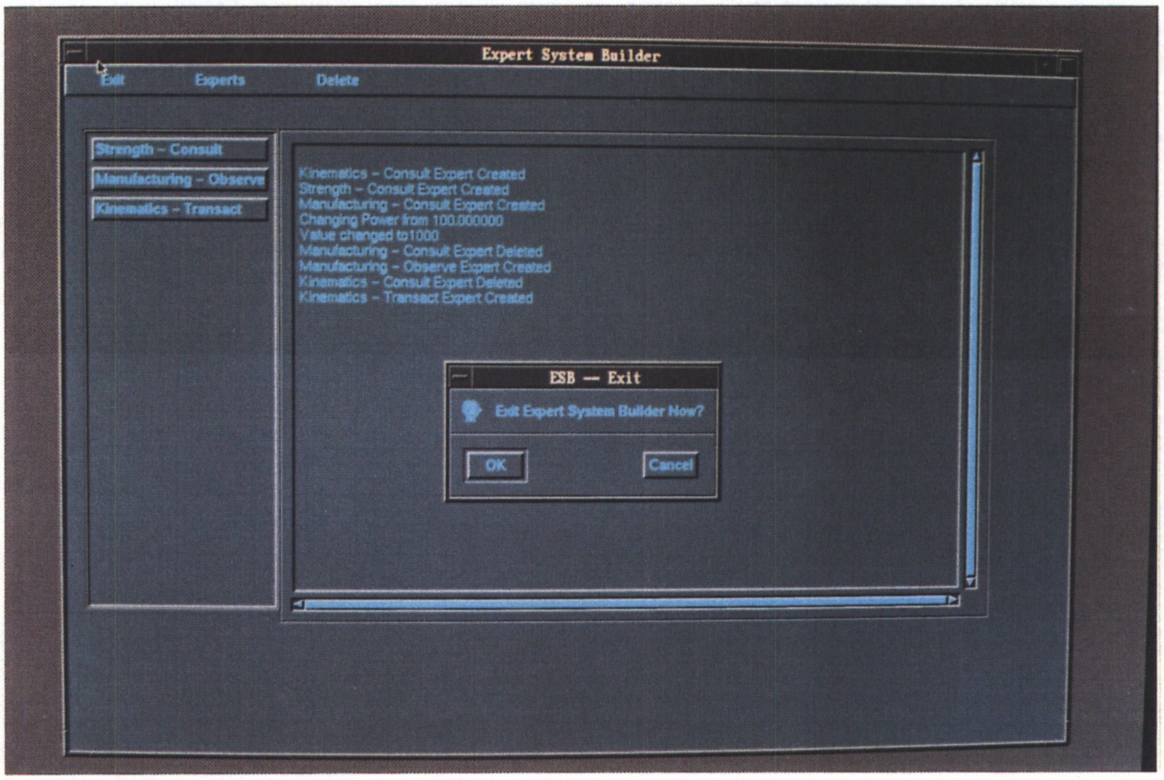
Teach_Popup: Pop-ups for the Teach class have been implemented in this class. The menus has buttons for setting an object of this class to be active or inactive.

Student: Constructor for this class has been implemented but the functions have not been built into this class.

Learn_Popup: The pop-up menus for the Student class have been implemented in this class. There are two buttons for the learn class on the pop-up for the class object one each for the forced learning and automatic learning.

Exit_Dialog: This class creates the menu for the user to confirm whether the user really wants to exit the session or not. This class has been implemented completely and deletes the objects of all the classes, closes the Motif application and exits the expert system session. Figure 22 shows the pop-up window for the confirmation from the user to exit the system.

During the implementation of these classes a number of problems were encountered. Even though implementation details were considered during the design stage of this framework, some problems could not be envisioned during this period. Due to these problems, the framework underwent minor design changes even during the implementation stage. As in any design, the iterative processes of design and implementation were carried out throughout this thesis. Even though the framework has



Kinematics - Consult Expert Created
Strength - Consult Expert Created
Manufacturing - Consult Expert Created
Changing Power from 100.000000
Value changed to1000
Manufacturing - Consult Expert Deleted
Manufacturing - Observe Expert Created
Kinematics - Consult Expert Deleted
Kinematics - Transact Expert Created

Figure 22 Pop-up for Confirmation from the User to Exit the Session

been designed taking into consideration future developments and expansion, some implementation details will necessitate changes to the design.

Most of the problems that were encountered during the implementation relates to the use of Motif as the user interface tool kit working within C++ classes. Even though Motif is an object-oriented toolkit, it has been implemented using C. C routines could be used as a subset of C++ classes, but the parameter passing for the function calls in Motif has been designed for C. The parameter passing in C is not as rigid as it is in C++. This created problems mainly in the callback functions of Motif. Callback functions are those functions which are called when an event is generated in the application program. Events might be generated when a push button is selected, an item from a list is selected or by any other action that necessitates the call of a function. These callbacks are attached to buttons or items when they are created so that when these are selected the callback functions are called.

While using Motif within C++ classes, these callback functions have to be declared as static functions returning the data type (void). A single instance of the static member exists independently of any object of the class. This makes it illegal for any non-static member to be present in the static member function in C++. This restriction poses problems in using variables or objects of other classes in these member functions. Variables and objects of other classes have to either be recreated in these functions or have to be declared as external in these static functions.

The Expert_Manager class object carries all the information regarding the various ETs and the operating mode objects. This manager object is constantly used in the call back functions of the Motif widgets to perform various tasks required by the user. To be used in the callback functions, the Expert_Manager object has to created as a new object in the callback functions or has to declared as an external object. The former is not feasible since we need to have only a single copy of the Expert_Manager object for the sake of consistency. This forced the declaration of the Expert_Manager object to be an external object in a number of other classes where it is being used in the callback functions. This satisfies the needs of Motif implemented within C++ classes and the requirement of this design to have a single copy of the Expert_Manager object for the sake of consistency. The definition of this Expert_Manager object is done in the Expert_Manager class and is declared as an external object in the other classes that need to use this object.

Motif follows the techniques of hierarchy in the creation of its objects. The main window is created first and all the objects in the main window are created as children of the main window. This hierarchy continues during the creation of other objects wherein parents have to be specified for the objects that are being created. Since these user interface objects are created in the various classes of the framework, using objects created in other classes as parents posed a problem of passing these objects around. This necessitated that a user interface object that is a parent of another object created in a different class be declared using global widgets. These widgets have been defined in a file Global.h file along with all the Motif include files.

Each ET created operates in different modes in interacting with the user. Each of the ET in a particular operating mode has a push button which appears on the window for the expert system. Selecting these buttons displays pop-up menus available for the ETs. The user interacts with the ETs using the choices on the pop-up menus available. It was desired that the pop-up menus appear on the buttons for the ETs. But during implementation it was observed that the pop-up menus appeared on the top left hand corner of the screen and changing the location of the pop-up menus by passing in location parameters only changed their location with reference to the whole screen of the terminal.

To rectify this an event handler was written using X windows calls which specifies the location of the cursor when the event is generated and positions the pop-up menus at the correct location.

The expert system that would be developed from this framework would support non-procedural multi-disciplinary analysis of designs. Figure 23 shows how the this framework could be used to create experts in different domains of an aircraft design system. The programmer developing the expert system would create experts in different domains of aircraft design. In the figure four domain experts are shown - economics, geometry, aerodynamics, and weights have been created by the programmer. A designer using the design system, in this case ACSYNT, a aircraft conceptual design system, would interact directly with the ETs. Thus these ETs would aid in the design process by providing replies to the users queries.

This framework would also support concurrent engineering techniques wherein experts involved at different stages in a product cycle get involved in the design process. This would aid the designer in taking into consideration the issues dealing with the different stages of the product cycle. An expert system that would be built for the CAD application could have experts created for the domains that deal with the product cycle. Figure 24 shows these experts in a session with the user. The expertise of these domains of the product cycle could be incorporated with the ETs and these ETs would aid the designer in the design process. As shown in the figure, these ETs could be experts in the domains of assembly, marketing, fabrication and maintenance. These expert would assist the designer to design the component by taking into consideration the whole product development cycle.

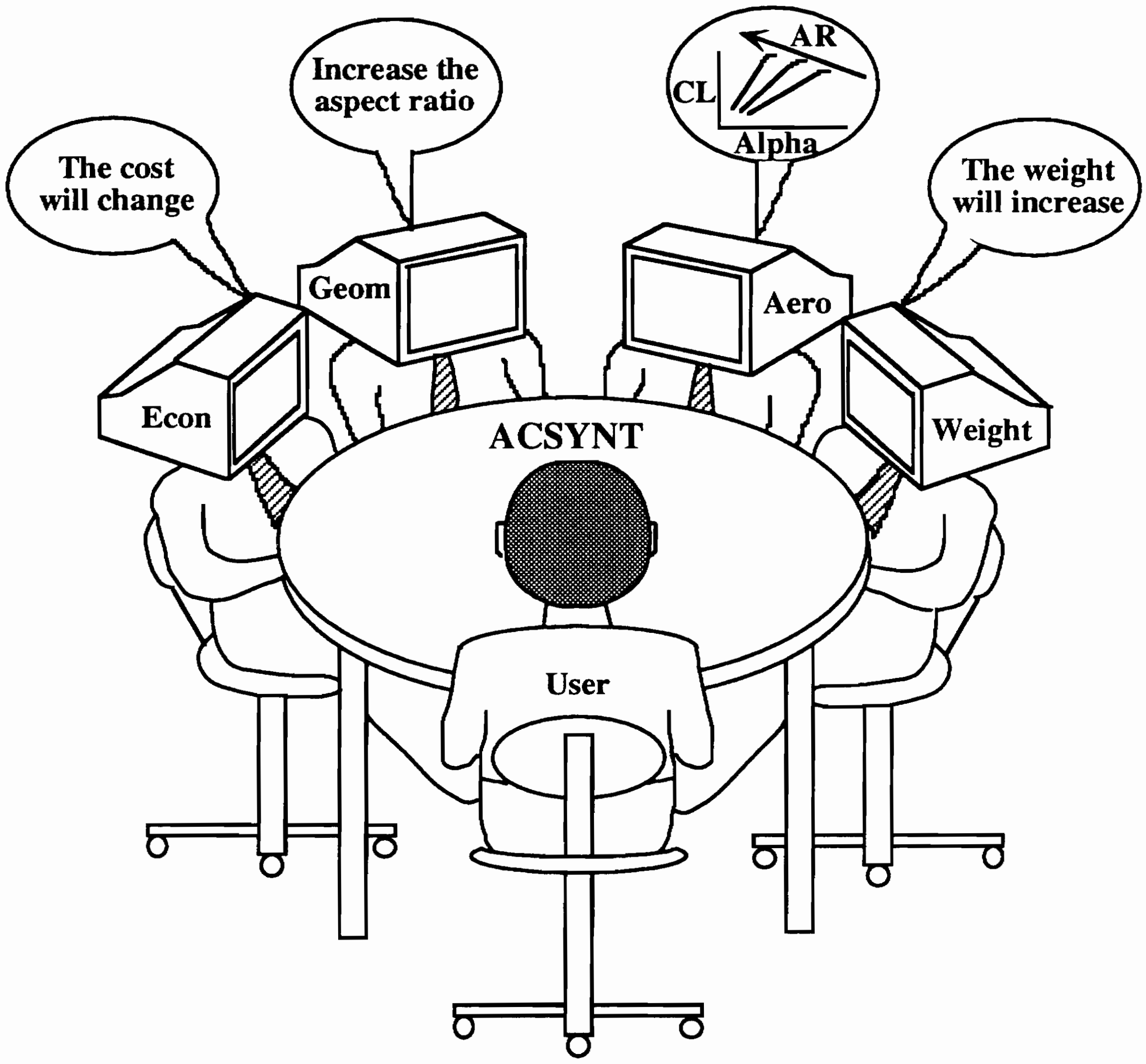


Figure 23 **Expert System Scenario for Multi-Disciplinary Design of Aircraft**

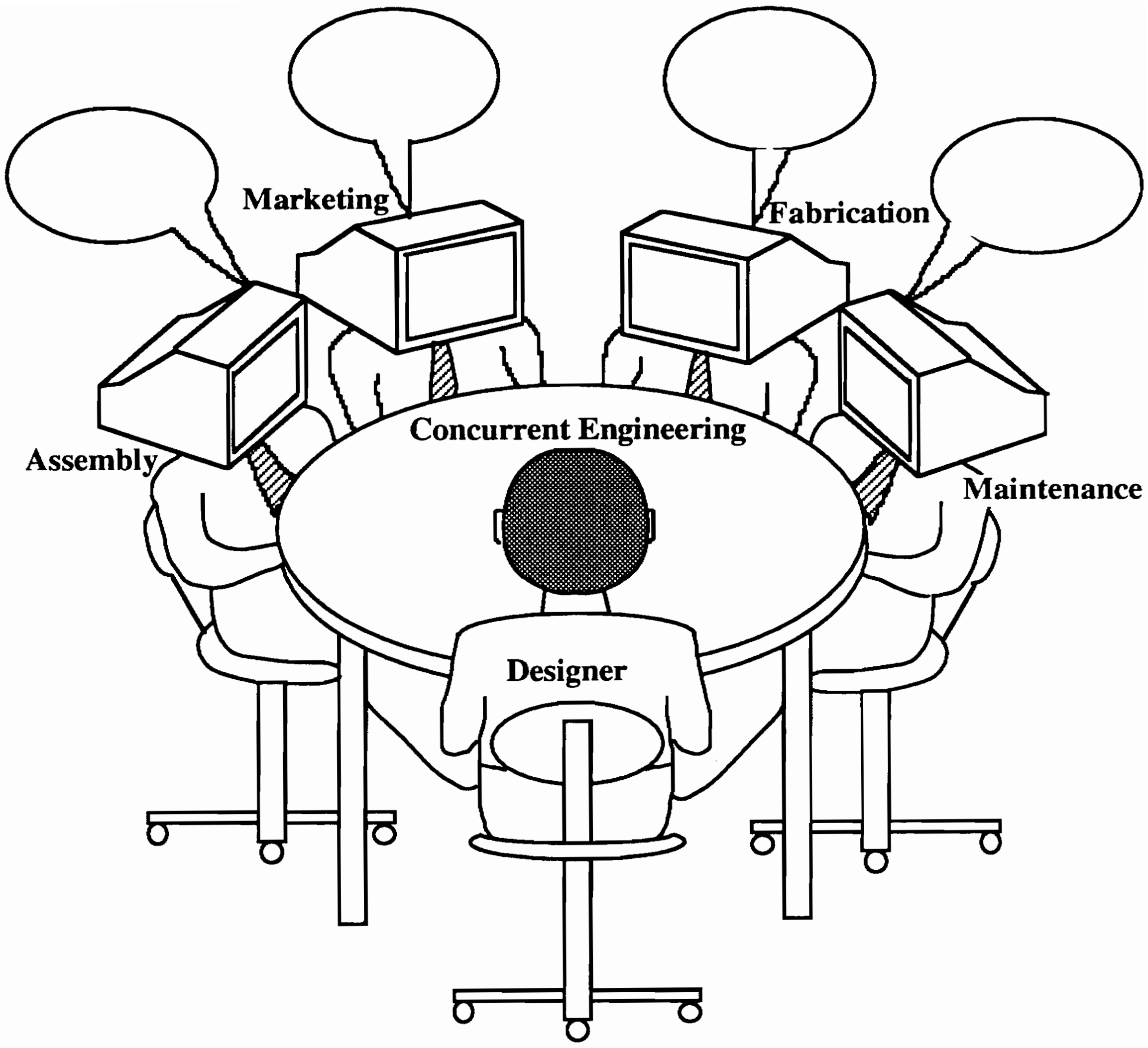


Figure 24 Expert System Scenario for Concurrent Engineering

12. Using the Framework

This chapter explains how this framework may be used to develop an expert system from the classes that have been created. This chapter further explains how the expert system developed from this framework may be used. There are two "users" that this chapter refers to. The first is the person who is involved in the development of the expert system from this framework and is referred to as the programmer throughout this chapter. The second is the final user of the expert system and is referred to as the user in this chapter.

The programmer using the framework makes uses of the classes implemented to create the expert system that will meet his specific needs. This chapter explains how the programmer needs to use the classes to create the expert system. Typically the programmer makes use of only three of the classes which were developed, to create the expert system. The three classes are the Session_Manager class, the Expert_Manager class and the ET class. The Expert_Manager class needs to be used by the programmer only if the programmer wishes to use more than a single session for the expert system. The number of sessions in use would be dictated by the number of users who want to use the expert system to work on the same problem at the same time.

To create an expert system from the framework the programmer needs to create a main program where the above mentioned classes will be used. Before using these classes the programmer has to include the files where these classes have been defined. The include files typically would be the *Session_Manager.h* and the *ET.h* files.

```
#include "Session_Manager.h"
#include "ET.h"
```

The Expert_Manager class definition has been included in the *Session_Manager.h* file and even if the programmer needs to use this class, the *Expert_Manager.h* file need not be included.

The programmer needs to create an object of the Session_Manager class in the main program. The syntax for this is

```
main () {
        ....................
        ...................
        Session_Manager ses_manager = new Session_Manager();
        ...................
        ....................
}
```

This creates a Session_Manager object called ses_manager. This Session_Manager object creates an Expert_Manager object. The Session_Manager object also initializes the Motif application for the user interfaces and creates all the user interface objects such as the main window, dialog boxes, etc. This object controls the expert system session that is created using the framework. A function of the Session_Manager class needs to be used in the main program. This function is handle_events. Typically this function call is the last line in the main program. Using the ses_manager object the syntax for this would be

```
        ses_manager.handle_events();
```

This function provides the loop back for the events generated in the application. After all the objects are created in the main program this function waits for the user to generate an event by selecting one of the push buttons on the screen. This makes sure that the main program is not exited until the exit button is selected on the main window of the application.

The programmer has to create all the experts needed for the expert system by creating objects of the ET class. This is done right after the Session_Manager object is created in the main program. The ET is the central part of the expert system created by using this framework. The programmer creates an ET for each of the domains of design. These ETs would have knowledge and methods regarding a certain domain of the design and would represent an expert in that domain.

The parameters that are passed in while the ET object is being created are the name of the ET and an operating mode in which the programmer requires that ET to interact with the user initially. If the operating mode is not specified, then the ET uses the default operating mode to interact with the user. Currently the default operating mode for the ET is the Consult mode.

```
ET aero("aero", "Observe");
ET geom("geom");
ET wht("weight", "Transact");
```

The above lines would create three experts. The first expert is an aerodynamics expert with the name aero. This would initially interact with the user in the observe mode. The geom expert that is created next would default to the Consult mode since an operating mode is not specified in the constructor of the ET class. Each ET has a name variable which is set to the name that is passed in. This might be different than the identifier for the object itself. Thus, putting all the above lines together to create a sample main program and is shown by figure 25.

These ETs that are created need to be fed with the knowledge. This knowledge would contain the rules and the parameters that belong to that ET. The knowledge for the ET is input through a stored object that the programmer creates. There is one stored object for each of the ET created. This object would contain all the parameters, their values and the rules that belong to each of the ET. This stored object is created while the ET object is being created by reading information from a file.

There is a separate stored object that needs to be created for each of the ET object containing the parameters and rules belonging to that expert. The name of the file should be the name of the ET with an extension kb. In the example shown above, the knowledge object for the aero expert would be aero.kb and the wht expert would be weight.kb. While the ET object is being created, the system looks for a stored object with the name of the expert and a kb extension. This stored object is read in and the parameters and the rules in the object are created as objects of the parameter and the rule classes.

The stored object may contain the parameters and rules in any order. The only restriction is that each parameter and rule must be entered in a new line in the file. The format for a parameter in the file would be

Parameter Aspect_ratio float 20.50

The first word in the line specifies that it is a parameter. The second word specifies the name of the parameter. The next word specifies the type of parameter. Two types of parameters are allowed namely float and string. The value of the parameter is specified

```
#include "Session_Manager.h"

#include "ET.h"


main() {


        Session_Manager ses_manager = new Session_Manager();

                        // Creates a new session manager object

                        // Session Manager creates a Expert Manager object


        ET aero("aero", "Observe");

                        // Creates an aerodynamics expert in observe mode


        ET geom("geom");

                        // Creates a geometry expert in the default consult mode


        ET wht("Weight", "Transact");

                        // Creates a weights expert in transact mode


        ses_manager.handle_events();

                        // Waits for an input from the menu


}
```

Figure 25 Sample Program to Create an Expert System from Framework

next. If the type of the parameter is string, the value could be an actual string. For example

Parameter Aircraft_Type string fighter

All the words in each line of the knowledge file are separated by a tab. As the parameters are being read in, instances of the parameter class are created in the ET. These parameters are then added to the linked list maintained in the ET object.

After the main program of the expert system is created, it should be compiled and the executable created. The user of the expert system would use the executable code to run a session of the expert system.

13. Conclusions

An object-oriented framework for the customized creation of expert systems for CAD was designed in this thesis. The requirements for the design of this framework were specified, the object-oriented design of the framework was completed and a prototype of the framework was created using C++. The central part of the expert system that will be created using this framework are the ETs which represent the experts in domains of design in real-world. These experts would be able to interact with the user as a Consultant, a Transactor, an Observer, a Teacher or a Student. The knowledge for the ETs in this expert system would be fed by the programmer during the creation of these ETs. The knowledge for the ETs are in the form of rules. These rules could be parametric equations, control equations, constraints, heuristic rules and data for the design parameters. These rules would operate on the design parameters of the design. The design parameters that are being used in the design belong to an ET depending on the domain of the parameter. Thus, the expert system developed from this framework would support parametric design.

CAD systems have traditionally used procedural methods to implement parametric design. Incorporating the procedural programming methods in an object-oriented environment has been a difficult task. Moreover, imbedding the procedures used in a traditional CAD applications as knowledge-based methods in an expert system proved to be difficult. The equations, constraints, control equations used in a procedural parametric design system can be embedded as rules in an expert system created using this framework.

The framework created would enhance the power of the CAD application and aid the users of the CAD system in designing better components faster. This framework would aid the programmers of CAD systems to build expert systems that are tailor-made for parametric conceptual design in a CAD application. A programmer using this framework would have tools to create expert systems that can be easily integrated with the overall CAD application and would be implemented in a language that is in everyday use by engineers.

14. References

[Angs93] Angster, S. R., "An Object Oriented, Knowledge-Based, Non-Procedural Approach to Multi-Disciplinary Parametric Conceptual Design", *M.S. Thesis*, Department of Mechanical Engineering, Virginia Polytechnic Institute and State University, July 19, 1993.

[Booc91] Booch, G., "Object-Oriented Design with Applications", *The Benjamin Cummings Publishing Company, Inc.*, CA., 1991.

[Bouc88] Bouchard, E. E., Kidwell, G. H. and Rogan, J. E., "The Application of Artificial Intelligence Technology to Aeronautical System Design", AIAA-88-4426, presented at the *AIAA/AHS/ASEE Aircraft Systems and Operations Meeting*, September 7-9, 1988, Atlanta, GA.

[Dym91] Dym, C. L., and Levitt, R. E., "Knowledge Based Systems In Engineering - 1 ed", *McGraw Hill Inc.*, 1991.

[Flem91] Fleming, S. and Myklebust, A., "Utilizing the graPHIGS API for CAD Application", *Proceedings of the Second International graPHIGS User's Group Conference and Workshop*, Blacksburg, Virginia, Oct 20-23, 1991, pp. 3-10.

[Flem92] Fleming, S. and Myklebust, A., "The Enhancement of PHIGS+ B-Spline Functionality for Geometric Modeling", presented at the *Fourth IFIP WG5.2 Workshop on Geometric Modeling in Computer Aided Design*, Rensselaerville, New York, Sept. 27 - Oct. 1, 1992.

[Fran90] Franke, D. W., “Imbedding Rule Inferencing in Applications”, *IEEE Expert , Intelligent Systems and Their Applications,* December 1990, pp 8-14.

[Gill92] Gillam, A., “A Knowledge-Based, Extensible Architecture for Space System Design”, AIAA-92- 1115, presented at the *1992 Aerospace Design Conference,* February 3-6, 1992, Irvine, CA.

[Grah93] Graham, I., “Migration Using SOMA: A semantically rich method of object oriented analysis”, *Journal of Object Oriented Programming*, Vol 5, No9, February 1993, pp 31-42.

[Ibra90] Ibrahim, M. H., and Woyak, S. W., “An Object-Oriented Environment for Multiple AI Paradigms”, *IEEE Conference on Tools for Artificial Intelligence*”, Herndon, VA, Nov 6-9, 1990.

[Jaya89] Jayaram, S., "CADMADE - An Approach Towards a Device-Independent Standard for CAD/CAM Software Development", *Ph.D. Dissertation*, Mechanical Engineering Department, Virginia Polytechnic Institute and State University, April 29, 1989.

[Jaya90] Jayaram, S., and Myklebust, A., "Towards a Standardized Environment for the Creation of Design and Manufacturing Software", *Proceedings of the International Conference on Engineering Design (ICED)*, Dubrovnik, Yugoslavia, August 28-31, 1990.

[Jaya93] Jayaram, S., and Myklebust, A., “Device Independent Programming Environments for CAD/CAM Software Creation”, *Computer Aided Design,* Volume25, No 2, February 1993, pp 94-105.

[Kidw87] Kidwell, G. H., "Workstations Take Over Conceptual Design", *Aerospace America*, January 1987.

[Kim86] Kim, S. H., and Suh, N. P., "On a Consultive Expert System for Design Axiomatics", *Journal of Robotics and Computer Integrated Manufacturing*, Vol 3, 1986.

[Kolb92] Kolb, M. A., "Constraint-Based Component Modeling for Knowledge-Based Design", AIAA-92-1192, presented at the *1992 Aerospace Design Conference*, February 3-6, 1992, Irvine, CA.

[Kolb88] Kolb, M. A., "A Flexible Computer Aid for Conceptual Design Based on Constraint Propagation and Component Modeling", *AIAA-88-4427.*

[Kroo88] Kroo, I. and Takai, M., "A Quasi-Procedural, Knowledge-Based System for Aircraft Design", AIAA-88-4428, presented at the *AIAA/AHS/ASEE Aircraft Design Systems and Operations Meeting*, September 7-9, 1988, Atlanta, GA.

[Kroo92] Kroo, I., "An Interactive System for Aircraft Design and Optimization", AIAA-92-1190, presented at the *1992 Aerospace Design Conference*, February 3-6, 1992, Irvine, CA.

[LinW93] Lin, W. and Myklebust, A., "A Constraint driven Solid Modeling Open Environment", accepted for presentation at the *Second ACM/IEEE Symposium on Solid Modeling and Applications*, Montreal, Canada, May 19-21, 1993.

[Lipp91] Lippman., S. B., "C++ Primer - 2 ed", *Addison-Wesley Publishing Company, Inc.*, 1991.

[Lu91] Lu, S. C-Y., and Wilhelm, R., "Automating Tolerance Synthesis: a Framework and Tools", *Journal of Manufacturing Systems*, Society of Manufacturing Engineers, Vol 10, No 4, 1991, pp 279-296.

[Nye90] Nye, A., and O'Reilly, T., "X Toolkit Intrinsics Programming Manual - 2 ed for X11 R4", *O'Reilly & Associates, Inc.*, 1990.

[Ohsu87] Ohsuga, S., "A Consideration to Intelligent CAD Systems", *Intelligent CAD - I,* North-Holland, 1989.

[OSF91] Open Software Foundation, "OSF/Motif Programmers Reference", *Prentice Hall,* NJ, 1991.

[Penn91] Pennington, S. L., "A Software Engineering Approach to the Integration of CAD/CAM Systems", *Ph.D. Dissertation*, Mechanical Engineering Department, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1991.

[Subr91] Subramanyam, S., and Lu, S. C-Y., "Computer Aided Simultaneous Engineering for Components Manufactured in Small and Medium Lot Sizes", *ASME Transactions, Journal of Engineering for Industry*, Vol 113, No 4, 1991, pp 450-464.

[Talu90] Talukdar, S. N., Sapossnek, M., Hou, L., Woodbury, R., Sedas, S., Saigal, S. and Jaeger, J., "Autonomous Critics", *Second National Symposium on Concurrent Engineering*, Concurrent Engineering Research Center, West Virginia University, Morgantown, WV, February 7-9, 1990.

[Tche91] Lu, S. C-Y., and Tcheng, D. K., "Building Layered Models to Support Engineering Decision Making: A Machine Learning Approach", *ASME Transactions, Journal of Engineering for Industry*, Vol 113, No 4, 1991, pp 1-9.

[Tong92] Tong, S. S., Powell, D. and Goel, S., "Integration of Artificial Intelligence and Numerical Optimization Techniques for the Design of Complex Aerospace Systems", AIAA-92-1189, presented at the *1992 Aerospace Design Conference*, February 3-6, 1992, Irvine, CA.

[Uhor93a] Uhorchak, S., "An Object-Oriented Class Library for Creating Engineering Graphs Using PHIGS", Presented at the *First Annual PHIGS User's Group Conference*, Orlando, Florida, March 21-24, 1993.

[Uhor93b] Uhorchak, S., "An Object-Oriented Class Library for the Creation of Engineering Graphs", *M.S. Thesis*, Department of Mechanical Engineering, Virginia Polytechnic Institute and State University, May 3, 1993.

[Ullm87] Ullman, D. G., and Dietterich, T. A., "Mechanical Design Methodology: *Implications on Future Developments of Computer-Aided Design and Knowledge Based Systems*", Engineering with Computers, Vol 2, 1987, pp 21-29.

[Winb90] Winbald, A. L., Edwards, S. D., and King, D. R., "Object Oriented Software", *Addison-Wesley Publishing Company, Inc.*, 1990.

[Woya93] Woyak, S., "A Motif-Like Object-Oriented Interface Framework Using PHIGS", Presented at the *First Annual PHIGS User's Group Conference*, Orlando, Florida, March 21-24, 1993.

[Yerr93] Yerramareddy, S., and Lu, S. C-Y, "Hierarchical and Interactive Decision Refinement Methodology for Engineering Design", *Knowledge Based Engineering Systems Research Laboratory*, University of Illinois at Urbana-Champaign.

Appendix A : Detailed Class Descriptions

Overview

Included in this appendix are the detailed descriptions of all the class members and functions that are used in all the classes in this framework. The classes that are included in this appendix are those that have been implemented and have been provided as a reference to those who wish to use the framework. It is assumed that those wishing to use this framework have knowledge of C++ and OSF/Motif. Some of the classes that were designed as a part of this thesis had not been implemented by the time this thesis was written and were later implemented by Scott Angster. For a detailed description of these classes refer to [Angs93].

Session_Manager Class

Class Description
This class creates the Expert_Manager object and creates various user interface objects for the interaction of the experts with the user.

Class Inheritance
Session_Interface class

Private Variables

Expert_Manager* **the_manager**	Pointer to the Expert_Manager object that is created

Public Functions

Session_Manager()

<u>Function Description</u>
Creates the Expert_Manager object and all the user interface objects through inheritance.

<u>Argument Description</u>
None.

Expert_Manager* **Get_Manager**()

<u>Function Description</u>
Returns a pointer to the expert manager object that is created in this class.

<u>Argument Description</u>
None.

void **Add_Expert_Manager** (Expert_Manager *manager)

<u>Function Description</u>
Adds the expert manager object that is passed in to the session that is created.

<u>Argument Description</u>

manager	Pointer to the expert manager object

Session_Interface Class

Class Description

This class initializes Motif application and creates all the main user interface objects like the menu bar, the main menu, dialog windows, scrolled windows, etc.

Class Inheritance

None.

Private Variables

Arg **al**[]	Argument list for creation of the Motif widgets.
int **ac**	Counter for the above argument list.
Widget **sw**	Scrolled window widget for dialog between user and the experts.
Widget **workArea**	Work area form widget for the main window.
Widget **bbframe**	Bulletin board widget frame widget for the dialog window.

Private Functions

Widget **Create_Menu_Bar** (Widget parent)

Function Description

Creates the main menu bar for the main menu of the expert system

Argument Description

parent	Parent for the widget, the main window widget is passed in.

static void **MenuCB** (Widget, XtPointer, XtPointer)

Function Description

Function provides a call back for the exit button on the main menu. Creates an object of the exit dialog class in the call back.

Argument Description

None.

Protected Variables

Widget **mainWindow**	The main window widget.
Widget **menuBar**	The main menu bar widget
Widget **frame**	Frame widget for the scrolled window
XtAppContext **app_context**	The application context for the initialization of Motif application

Public Functions

Session_Interface()

Function Description

Class constructor which creates all the widgets and initializes the Motif application

Argument Description

None.

void **handle_events**()

Function Description

This function provides a loop back for the application to wait for an event to be generated.

Argument Description

None.

Exit_Dialog Class

Class Description

This class object is created when the exit button on the main menu is selected. This class closes the Motif application and exits the program.

Class Inheritance

None.

Private Variables

Arg **al**[]	Motif argument
int **ac**	Argument counter for the above

Private Function

static void **DialogCB** (Widget, XtPointer client_Data, XtPointer)

Function Description

Call back function for the OK button on the exit dialog window. Closes the Motif application and exits the program.

Argument Description

client_Data	Button ID for the OK button

Public Functions

Exit_Dialog()

Function Description

This function creates a question box for confirmation on whether the user really wants to exit the system. If yes then the call back for the OK button is called.

Argument Description

None.

static void **show_exitdialog**()

Function Description

Manages the question box widget for the exit dialog.

Argument Description

None.

Expert_Manager Class

Class Description

This class manages all the experts that are created by the user. It keeps track of all the experts, processes requests from them and provides functions for them to interact with the user.

Class Inheritance

Expert_Setup, Delete_Menu

Private Variables

XFontStruct ***newfont**	Font structure for output font

XmFontList **newfontlist**	List of fonts
Arg **al**[]	Motif argument list
int **ac**	Argument counter for the list
Widget **windowtext**	Widget for the text window
char ***output_string**	Output string that is written to the text window
int **No_of_Consult_Experts_Created**	No of consult objects created in all the ETs
int **No_of_Observe_Experts_Created**	No of observe objects created in all the ETs
int **No_of_Transact_Experts_Created**	No of transact objects created in all the ETs
int **No_of_Teach_Experts_Created**	No of teach objects created in all the ETs
int **No_of_Learn_Experts_Created**	No of teach objects created in all the ETs
Consult* **first_consult**	Pointer to the first consult object in the linked list maintained by the manager
Observe* **first_observe**	Pointer to the first observe object in the linked list maintained by the manager
Transact ***first_transact**	Pointer to the first transact object in the linked list maintained by the manager
Teach ***first_teach**	Pointer to the first teach object in the linked list maintained by the manager
Learn ***first_learn**	Pointer to the first learn object in the linked list maintained by the manager
Consult* **curr_consult**	Pointer to the last consult object in the linked list maintained by the manager
Observe* **curr_observe**	Pointer to the last observe object in the linked list maintained by the manager
Transact ***curr_transact**	Pointer to the last transact object in the linked list maintained by the manager
Teach ***curr_teach**	Pointer to the last teach object in the linked list maintained by the manager
Learn ***curr_learn**	Pointer to the last learn object in the linked list maintained by the manager

ET ***first_ET**	Pointer to the first ET object in the linked list maintained by the manager
ET ***curr_ET**	Pointer to the last ET object in the linked list maintained by the manager
int **No_of_ETs**	No of ET objects created by the user

Public Functions

Expert_Manager()

Function Description
Sets the name of the expert and initializes all the variables to keep track of the experts that are created

Argument Description
None

void **Add_Consult** (Consult *cons)

Function Description
Adds the consult expert passed in to the linked list of consult objects maintained

Argument Description

cons	Pointer to the consult object that needs to be added to the linked list

void **Add_Observe** (Observe *obs)

Function Description
Adds the observe expert passed in to the linked list of observe objects maintained

Argument Description

obs	Pointer to the observe object that needs to be added to the linked list

void **Add_Transact** (Transact *trans)

Function Description
Adds the transact expert passed in to the linked list of transact objects maintained

Argument Description

trans	Pointer to the transact object that needs to be added to the linked list

void **Add_Teach** (Teach *teach)

Function Description
Adds the teach expert passed in to the linked list of teach objects maintained

Argument Description

teach	Pointer to the teach object that needs to be added to the linked list

void **Add_Learn** (Learn *learn)

Function Description
Adds the learn expert passed in to the linked list of learn objects maintained

Argument Description

learn	Pointer to the learn object that needs to be added to the linked list

void **Write_To_User** (char *msg)

Function Description
Writes the string that is passed in to the dialog window

Argument Description

msg	String that needs to be written to the user

void **Remove_Consult** (Consult *cons)

Function Description
Removes the consult expert passed in from the linked list

Argument Description

cons	Pointer to the consult expert needed to be removed from the linked list

void **Remove_Observe** (Observe *obs)

Function Description
Removes the observe expert passed in from the linked list

Argument Description

obs	Pointer to the observe expert needed to be removed from the linked list

void **Remove_Transact** (Transact *trans)

Function Description
Removes the transact expert passed in from the linked list

Argument Description

cons	Pointer to the transact expert needed to be removed from the linked list

void **Remove_Teach** (Teach *teach)

Function Description
Removes the teach expert passed in from the linked list

Argument Description

teach	Pointer to the teach expert needed to be removed from the linked list

void **Remove_Learn** (Teach *learn)

Function Description
Removes the learn expert passed in from the linked list

Argument Description

learn	Pointer to the learn expert needed to be removed from the linked list

int **Get_No_of_Consult**()

Function Description
Returns the number of consult experts created

Argument Description
None

int **Get_No_of_Observe**()

Function Description
Returns the number of observe experts created

Argument Description
None

int **Get_No_of_Transact**()

Function Description
Returns the number of transact experts created

Argument Description
None

int **Get_No_of_Teach**()

Function Description
Returns the number of teach experts created

Argument Description
None

int **Get_No_of_Learn**()

Function Description
Returns the number of learn experts created

Argument Description
None

void **Add_ET** (ET *et)

Function Description
Adds the ET object passed in to the linked list maintained

Argument Description

et	Pointer to the ET object that needs to be added to the linked list

int **Get_No_of_ETs**()

Function Description
Returns the number of ET objects that have been created by the user

Argument Description
None

ET ***Get_First_ET**()

Function Description
Returns a pointer to the first ET in the linked list of ETs

Argument Description
None

float **Get_Value** (char *param_name)

Function Description
Returns the value of the parameter, the name of the parameter is passed in

Argument Description

param_name	Pointer to the name of the parameter

char ***Get_Char_Value** (char *param_name)

Function Description

Returns the character value of the parameter, a pointer to whose name is passed in

Argument Description

param_name	A pointer to the name of the parameter

char ***Get_Type** (char *param_name)

Function Description

Returns "float" if the parameter is of the type float or "string" if the parameter is of the type string

Argument Description

param_name	A pointer to the name of the parameter whose type is needed

int **Is_Present** (char *param_name)

Function Description

Returns a 1 if the parameter whose name is passed in is a member of any of the ETs

Argument Description

param_name	A pointer to the name of the parameter

void **Put_Value** (char *param_name, float value)

Function Description

Sets a float value passed in to a parameter

Argument Description

param_name	Pointer to the name of the parameter
value	Value that needs to be set to the parameter

Void **Put_Char_Value** (char *param_name, char *value)

Function Description

Sets a string to a string parameter

Argument Description

param_name	Pointer to the name of a parameter
value	Pointer to the string that needs to stored

Expert_Setup Class

Class Description
Creates the user interfaces for the creation of a particular operating mode of the ET

Class Inheritance
None

Public Variables

Arg **al**[]	Motif Argument list
int **ac**	Counter for the Motif argument list
Widget ***setupchildren**	Button widgets for the operating modes of the ET
int **noofsetupchildren**	No of buttons for the operating modes

Public Functions

Expert_Setup()

Function Description
: Creates the user interface menu for the creation of the operating mode objects of the ETs

Argument Description
: None

void **Update_Setup**()

Function Description
: Updates the menus whenever a new ET or a new operating mode object of an ET is created or deleted

Argument Description
: None

static void **Create_Expert** (Widget, XtPointer client_Data, XtPointer)

Function Description
: Call back for the buttons to create the operating modes of the experts

Argument Description

client_Data	IDs of the ET and the operating mode

Delete_Menu Class

Class Description
Creates the user interfaces for the deletion of a particular operating mode of the ET

Class Inheritance
None

Public Variables

Arg **al**[]	Motif Argument list
int **ac**	Counter for the Motif argument list
Widget ***deletechildren**	Button widgets for the operating modes of the ET
int **noofdeletechildren**	No of buttons for the operating modes

Public Functions

Delete_Menu()

Function Description
Creates the user interface menu for the deletion of the operating mode objects of the ETs

Argument Description
None

void **Update_Delete**()

Function Description
Updates the menus whenever a new ET or a new operating mode object of an ET is created or deleted

Argument Description
None

static void **Delete_Expert** (Widget, XtPointer client_Data, XtPointer)

Function Description
Call back for the buttons to delete the operating modes of the experts

Argument Description

client_Data	IDs of the ET and the operating mode

Parameter Class

Class Description
Class represents a design parameter used in the design, could be of the type float or string.

Class Inheritance
None

Private Variables

float **curr_value**	The current float value of the parameter
float **changeable**	Represents whether the parameter value is user changeable or not. 1 = user changeable 0 = user cannot change the value of the parameter
char ***curr_char_value**	If the parameter is of the type string, the value of the parameter
char ***owner**	Pointer to the name of the ET that owns the parameter
float **minimum**	The minimum float value of the parameter
float **maximum**	the maximum float value of the parameter

Public Variables

char **type**[]	The type of the parameter, whether float or string
char **name**[]	The name of the parameter
Parameter ***next**	Pointer to the next parameter in the linked list of parameters that is maintained by the ET

Public Functions

Parameter (char *pname, char *ptype, float value, float changeable_value, float min_value, float max_value)

Function Description

Constructor for the float type parameter

Argument Description

pname	Pointer to the name of the parameter
ptype	Pointer to the type of the parameter
value	Float value of the parameter
changeable_value	User changeable flag
min_value	Minimum float value of the parameter
max_value`	Maximum value of the parameter

Parameter (char *pname, char *ptype, char* value, float changeable_value, float min_value, float max_value)

Function Description

Constructor for the string type parameter

Argument Description

pname	Pointer to the name of the parameter
ptype	Pointer to the type of the parameter
value	string value of the parameter
changeable_value	User changeable flag
min_value	Minimum float value of the parameter
max_value`	Maximum value of the parameter

char* **get_type**()

Function Description

Returns the type of the parameter

Argument Description

None

char* **get_name**()

Function Description
Returns the name of the parameter

Argument Description
None

float **get_type**()

Function Description
Returns the current float value of the parameter

Argument Description
None

float **get_changeable**()

Function Description
Returns 1 if the parameter value is user changeable and 0 if it is not

Argument Description
None

char* **get_char_value**()

Function Description
Returns the string value of the parameter

Argument Description
None

void **put_value** (float value)

Function Description
Stores the float value passed in as the value of the parameter

Argument Description

value	float value of the parameter that is passed in

void **put_value** (char *value)

Function Description
Stores the string value passed in as the value of the parameter

Argument Description

value	Pointer to a string that needs to be stored as the value of the parameter

float **get_max**()

Function Description
Returns the maximum value of the parameter

Argument Description
None

float **get_min**()

Function Description
Returns the minimum value of the parameter

Argument Description
None

ET Class

Class Description
Represents an expert in a particular domain of design

Class Inheritance
None

Private Variables

char ***name**	Name of the ET
int **No_of_Parameters**	No of parameters belonging to the ET
int **No_of_Changeable**	No of parameters that the user can change the values of
Consult ***cons**	Pointer to the consult operating mode object
Observe ***obs**	Pointer to the observe operating mode object
Transact ***trans**	Pointer to the transact operating mode object
Teach ***teach**	Pointer to the teach operating mode object
Learn ***learn**	Pointer to the learn operating mode object

Private Functions

void **Read_KB**()

Function Description
Reads in the knowledge of the ET, stored in the form of rules, parameters belonging to the ET, etc.

Argument Description
None

Public Variables

Rule ***first_rule**	Pointer to the first rule in the linked list of rules maintained by the ET
int **No_of_Rules**	No of rules in the rule linked list
int **id**	ID of the ET
Parameter ***first_parameter**	Pointer to the first parameter in the linked list of parameters maintained by the ET
Parameter ***p_ll**	Pointer to a parameter in the parameter linked list
int **active_modes**[]	Each of the item in the array indicates whether a particular mode in the ET is active or not. 1 indicates the operating mode is active and 0 indicates the operating mode is inactive
ET ***next**	Pointer to the next ET in the linked list of ETs maintained by the Expert_Manager
Arg **al**[]	Motif argument list array
int **ac**	Counter for the above array
Widget **max_warning**	Widget for the warning box to warn the user that the value of a parameter is exceeding the maximum value
Widget **min_warning**	Widget for the warning box to warn the user that the value of the parameter is lower than the minimum value
XmString **warning_string**	String written to the user in the warning box
char ***string**	Warning string

Public Functions

ET (char *expertname, char *expertmode)

Function Description

Class constructor creates the expert object and sets one of the operating modes specified by the user active

Argument Description

expertname	Pointer to the name of the expert
expertmode	Pointer to the initial active mode of the ET

void **Create_Expert_Mode** (char *expertmode)

Function Description

Creates an operating mode object of the ET

Argument Description

expertmode	Pointer to the string specifying the operating mode of the ET that is to be created

void **Delete_Expert_Mode** (char *expertmode)

Function Description

Deletes an operating mode object of the ET

Argument Description

expertmode	Pointer to the string specifying the operating mode of the ET that is to be deleted

char ***Get_Name**()

Function Description

Returns the name of the ET

Argument Description

None

int **Get_No_Parameters**()

Function Description

Returns the number of parameters that belong to the ET

Argument Description

None

int **Get_No_Changeable**()

<u>Function Description</u>
Returns the number of parameters whose values could be changed by the user

<u>Argument Description</u>
None

int **Get_ID**()

<u>Function Description</u>
Returns the ID of the ET

<u>Argument Description</u>
None

void **Add_Parameter** (char *param_name, char *type, char* value,
char* user_changeable char* max_value, char* min_value)

<u>Function Description</u>
Creates a parameter object and adds it to the linked list

<u>Argument Description</u>

param_name	Pointer to the name of the parameter
type	Type of parameter - string or float
value	Value of the parameter
user_changeable	Pointer to string specifying whether parameter is user changeable or not 1 - user changeable 0 - not user changeable
min_value	Minimum value of the parameter
max_value	Maximum value of the parameter

void **Put_Value** (Parameter *parametr, float value)

<u>Function Description</u>
Stores the value of a parameter

<u>Argument Description</u>

parametr	Pointer to the parameter whose value has to be changed
value	Value that has to be stored

float **Get_Min_Value** (Parameter *parametr)

Function Description

Returns the minimum value of the parameter

Argument Description

parametr	Pointer to the parameter whose minimum value is required

float **Get_Max_Value** (Parameter *parametr)

Function Description

Returns the maximum value of the parameter

Argument Description

parametr	Pointer to the parameter whose maximum value is required

float **Get_Value** (char *param_name)

Function Description

Returns the float value of the parameter of type float

Argument Description

param_name	Pointer to the name of the parameter

char* **Get_Char_Value** (char* param_name)

Function Description

Returns the string value of the parameter of the type string

Argument Description

param_name	Pointer to the name of the parameter

int **Is_Present** (char* param_name)

Function Description

Returns a 1 if the parameter is present in a linked list maintained by any of the ETs else returns a 0

Argument Description

param_name	Pointer to the name of the parameter

int **Check_String** (char *str)

Function Description

Checks if the string is a mathematical function name or a regular string. If the string is a mathematical function name this function returns a 1 else a 0

Argument Description

str	Pointer to the string that needs to be checked

int **Format_Type** (char val[])

Function Description

Checks if the string passed in is a character string or a numerical string. If it is an alpha character string, this function returns a 1 and if the string is a numerical string a 0 is returned

Argument Description

val[]	String that needs to be checked

Consult Class

Class Description
This class represents the Consult operating mode of the ET

Class Inheritance
Consult_Popup

Private Variables

Arg **al**[]	Motif argument list
int **ac**	Counter for the above argument list
int **current_no**	ID of the expert
char ***name**	Pointer to the name of the ET
char * **param_name**	Temporary name of parameter
float **param_value**	Value of the above parameter

Private Functions

static void **Button_CallBack** (Widget, XtPointer call_data, XtPointer)

Function Description

Call back function for the ET button on the main scrolled window. Displays all the menus for the ET

Argument Description

call_data	ID for the ET

Public Variables

Widget **button**	Button widget for the ET on the main window
Consult* **previous**	Pointer to the previous Consult object in the doubly linked list of consult objects maintained by the Expert_Manager
Consult* **next**	Pointer to the next consult object on the doubly linked list of consult objects maintained by the Expert_Manager

Public Functions

Consult (char* expert_name)

Function Description
Constructor for the consult class

Argument Description

expert_name	Pointer to the name of the expert

~Consult()

Function Description
Destructor for the consult class

Argument Description
None

char ***Get_Name**()

Function Description
Returns the name of the expert

Argument Description
None

static void **Param_Whatis** (Widget, XtPointer call_data, XtPointer)

Function Description
Call back function for "What Is" button on the menu for the Consultant. Displays all the parameters in a pull-down menu and displays information on the parameter that is selected

Argument Description

call_data	ID for the parameter that is selected

static void **Param_Effect_Increase** (Widget, XtPointer call_data, XtPointer)

Function Description

Call back for the effect increase button.

Argument Description

call_data	ID of the ET and parameter selected

static void **Param_Effect_Descrease** (Widget, XtPointer call_data, XtPointer)

Function Description

Call back for the effect decrease button.

Argument Description

call_data	ID of the ET and parameter selected

static void **Param_Effect_Maximize** (Widget, XtPointer call_data, XtPointer)

Function Description

Call back for the effect maximize button.

Argument Description

call_data	ID of the ET and parameter selected

static void **Param_Effect_Minimize** (Widget, XtPointer call_data, XtPointer)

Function Description

Call back for the effect minimize button.

Argument Description

call_data	ID of the ET and parameter selected

static void **Param_Effect_New_Value** (Widget, XtPointer call_data, XtPointer)

Function Description

Call back for the effect new value button.

Argument Description

call_data	ID of the ET and parameter selected

static void **Receive_prompt_value** (Widget, XtPointer Client_Data,
XtPointer sel_struct)

Function Description

Receives the new value of the parameter entered by the user in the prompt window

Argument Description

Client_Data	Pointer to the consult object that calls the function
sel_struct	Pointer to the structure with the new value entered

static void **Receive_prompt_increase** (Widget, XtPointer Client_Data, XtPointer sel_struct)

Function Description

Receives the percentage increase of the parameter entered by the user in the prompt window

Argument Description

Client_Data	Pointer to the consult object that calls the function
sel_struct	Pointer to the structure with the percentage increase entered

static void **Receive_prompt_value** (Widget, XtPointer Client_Data, XtPointer sel_struct)

Function Description

Receives the percentage decrease of the parameter entered by the user in the prompt window

Argument Description

Client_Data	Pointer to the consult object that calls the function
sel_struct	Pointer to the structure with the percentage decrease entered

Consult_Popup Class

Class Description

This class contains all the user interface functions for the consult class

Class Inheritance

None

Private Variables

Arg **al**[]	Motif argument list

int **ac**	Counter for the above list
Widget **popupbutton**	Button widget for the pop-up buttons
XmString **label**	Label string for the buttons
Widget **popup_pulldown_pane**[]	Main pull-down panes for the menus
Widget **param_pulldown_pane**[]	Pull-down panes where the parameters are listed
Widget **changeable_pulldown_pane**[]	Pull-down panes where the user changeable parameters are listed
Widget **enter_text**	Widget for the prompt window
char ***name**	Pointer to the name of the ET
int **ETID**	ID of the ET
int **no_parameter**	No of parameters owned by the ET
int **no_changeable**	No of changeable parameters owned by the ET
char ****p_list**	List of parameter names for display on the menus
char ****user_list**	List of user changeable parameter names
Widget **popupwindow**	Pop-up window for the parameters
int **no_on_list**	No of parameters in the list of parameters
int **no_of_changeable_panes**	No of pull-down panes on which the user changeable parameters are listed

Private Functions

void **Get_Param_list**()

Function Description
Gets the list of parameters belonging to this ET

Argument Description
None

void **Get_Changeable_List**()

Function Description
Gets the list of user changeable parameters belonging to this ET

Argument Description
None

void **Create_Parameter_Pulldown**()

Function Description
Creates pull-down panes with all the parameters as separate push buttons

Argument Description
None

int **Create_Changeable_Pulldown**()

Function Description
Creates pull-down panes with all the user changeable parameters as push buttons. Returns the no. of panes needed to display all the parameters

Argument Description
None

void **Create_Determine_Pulldown**()

Function Description
Creates pull-down panes for the "Determine" menu choice and displays all the user changeable parameters on the pull-down pane

Argument Description
None

Protected Variables

float **entered_number**	The value entered by the user in the prompt window
float **new_value**	Value entered by the user to change the current value of a parameter or the value changed as a result of percentage increase or decrease
float **percentage**	Percentage increase or decrease entered by the user

struct **effect_callback_struct{**	Structure used to pass information regarding the parameter to the call back function
int **ETID**	ID of the parameter
int **parameter_no**	ID of the parameter
Consult_Popup ***this_pointer**	Pointer to the object calling the call back function
} ***params**	

Protected Function

Widget **Prompt_user** (char* msg, char* msg2, char *reason)

<u>Function Description</u>

Prompts the user for a new value of the parameter or the percentage increase or decrease of the parameter

<u>Argument Description</u>

msg	Message on the prompt window
msg2	Message to enter string
reason	Whether a new value or percentage increase or decrease

Public Functions

Consult_Popup (char* ETname)

<u>Function Description</u>

Constructor for the Consult_Popup class

<u>Argument Description</u>

ETname	Pointer to the name of the ET

void **Create_Consult_Popup**()

<u>Function Description</u>

Creates the main pop-up menus for the consult mode expert

<u>Argument Description</u>

None

void **Create_Consult_Popup_Pulldown**()

<u>Function Description</u>
Creates the pulldown menus for the pop-up buttons in the main menu of the consult mode ET

<u>Argument Description</u>
None

Observe Class

Class Description
This class represents the observe mode expert.

Class Inheritance
Observe_Popup

Private Variables

Arg **al**[]	Motif argument list
int **ac**	Counter for the above list
int **current_no**	ID for the ET
char ***name**	Pointer to the name of the ET

Private Function

static void **Button_CallBack** (Widget, XtPointer call_data, XtPointer)

<u>Function Description</u>
Call back function for the ET button on the main menu. Displays the pull-down menu for the ET

<u>Argument Description</u>

call_data	ID for the ET and the expert mode

Public Variables

Widget **button**	Button widget for the observe mode ET on the main menu

Observe ***previous**	Pointer to the previous observe mode expert in the doubly linked list maintained by the Expert_Manager
Observe ***next**	Pointer to the next observe mode expert in the doubly linked list maintained by the Expert_Manager

Public Functions

Observe (char *ETname)

Function Description

Constructor for the observe operating mode

Argument Description

ETname	Pointer to the name of the ET

~Observe()

Function Description

Destructor for the class

Argument Description

None

char* **Get_Name**()

Function Description

Returns the name of the ET that this observe mode object belongs to

Argument Description

None

Observe_Popup Class

Class Description

This class has all the user interface functions for the observe operating mode ET

Class Inheritance

None

Private Variables

Arg **al**[]	Motif argument list

int **ac**	Counter for the above argument
Widget **popupbutton**	Button widget for the pop-up pane menu items
XmString **label**	Label for the menu items
Widget **popup_pulldown_pane**	Pop-up pull-down pane for the menu items for this operating mode

Public Functions

Observe_Popup()

Function Description

Constructor for the class. Creates all the menus for the observe mode expert

Argument Description

None

void **Create_Observe_Popup()**

Function Description

Creates the pop-up panes and the main menu for the observe mode ET

Argument Description

None

Transact Class

Class Description

This class represents the transact operating mode of the ET

Class Inheritance

Transact_Popup

Private Variables

Arg **al**[]	Motif argument list
int **ac**	Counter for the above argument list
int **current_no**	ID of the expert

char ***name**	Pointer to the name of the ET
char * **param_name**	Temporary name of parameter
float **param_value**	Value of the above parameter

Private Functions

static void **Button_CallBack** (Widget, XtPointer call_data, XtPointer)

Function Description

Call back function for the ET button on the main scrolled window. Displays all the menus for the transact mode ET

Argument Description

call_data	ID for the ET

Public Variables

Widget **button**	Button widget for the ET on the main window
Transact* **previous**	Pointer to the previous transact object in the doubly linked list of transact objects maintained by the Expert_Manager
Transact* **next**	Pointer to the next transact object in the doubly linked list of transact objects maintained by the Expert_Manager

Public Functions

Transact (char* expert_name)

Function Description

Constructor for the transact class

Argument Description

expert_name	Pointer to the name of the expert

~Transact()

Function Description

Destructor for the transact class

Argument Description
None

char ***Get_Name**()

Function Description
Returns the name of the expert

Argument Description
None

static void **Param_Whatis** (Widget, XtPointer call_data, XtPointer)

Function Description
Call back function for "What Is" button on the menu for the Transactor. Displays all the parameters in a pull-down menu and displays information on the parameter that is selected

Argument Description

call_data	ID for the parameter that is selected

static void **Param_Effect_Increase** (Widget, XtPointer call_data, XtPointer)

Function Description
Call back for the effect increase button.

Argument Description

call_data	ID of the ET and parameter selected

static void **Param_Effect_Descrease** (Widget, XtPointer call_data, XtPointer)

Function Description
Call back for the effect decrease button.

Argument Description

call_data	ID of the ET and parameter selected

static void **Param_Effect_Maximize** (Widget, XtPointer call_data, XtPointer)

Function Description
Call back for the effect maximize button.

Argument Description

call_data	ID of the ET and parameter selected

static void **Param_Effect_Minimize** (Widget, XtPointer call_data, XtPointer)

Function Description
Call back for the effect minimize button.

Argument Description

call_data	ID of the ET and parameter selected

static void **Param_Effect_New_Value** (Widget, XtPointer call_data, XtPointer)

Function Description

Call back for the effect new value button.

Argument Description

call_data	ID of the ET and parameter selected

static void **Receive_prompt_value** (Widget, XtPointer Client_Data, XtPointer sel_struct)

Function Description

Receives the new value of the parameter entered by the user in the prompt window

Argument Description

Client_Data	Pointer to the transact object that calls the function
sel_struct	Pointer to the structure with the new value entered

static void **Receive_prompt_increase** (Widget, XtPointer Client_Data, XtPointer sel_struct)

Function Description

Receives the percentage increase of the parameter entered by the user in the prompt window

Argument Description

Client_Data	Pointer to the transact object that calls the function
sel_struct	Pointer to the structure with the percentage increase entered

static void **Receive_prompt_value** (Widget, XtPointer Client_Data, XtPointer sel_struct)

Function Description

Receives the percentage decrease of the parameter entered by the user in the prompt window

Argument Description

Client_Data	Pointer to the transact object that calls the function

sel_struct	Pointer to the structure with the percentage decrease entered

Transact_Popup Class

Class Description

This class contains all the user interface functions for the transact class

Class Inheritance

None

Private Variables

Arg **al**[]	Motif argument list
int **ac**	Counter for the above list
Widget **popupbutton**	Button widget for the pop-up buttons
XmString **label**	Label string for the buttons
Widget **popup_pulldown_pane**[]	Main pull-down panes for the menus
Widget **param_pulldown_pane**[]	Pull-down panes where the parameters are listed
Widget **changeable_pulldown_pane**[]	Pull-down panes where the user changeable parameters are listed
Widget **enter_text**	Widget for the prompt window
char ***name**	Pointer to the name of the ET
int **ETID**	ID of the ET
int **no_parameter**	No of parameters owned by the ET
int **no_changeable**	No of changeable parameters owned by the ET
char ****p_list**	List of parameter names for display on the menus
char ****user_list**	List of user changeable parameter names
Widget **popupwindow**	Pop-up window for the parameters

int **no_on_list**	No of parameters in the list of parameters
int **no_of_changeable_panes**	No of pull-down panes on which the user changeable parameters are listed

Private Functions

void **Get_Param_List**()

Function Description
Gets the list of parameters belonging to this ET

Argument Description
None

void **Get_Changeable_List**()

Function Description
Gets the list of user changeable parameters belonging to this ET

Argument Description
None

void **Create_Parameter_Pulldown**()

Function Description
Creates pull-down panes with all the parameters as separate push buttons

Argument Description
None

int **Create_Changeable_Pulldown**()

Function Description
Creates pull-down panes with all the user changeable parameters as push buttons. Returns the no. of panes needed to display all the parameters

Argument Description
None

void **Create_Determine_Pulldown**()

Function Description
Creates pull-down panes for the "Determine" menu choice and displays all the user changeable parameters on the pull-down pane

Argument Description
None

Protected Variables

float **entered_number**	The value entered by the user in the prompt window
float **new_value**	Value entered by the user to change the current value of a parameter or the value changed as a result of percentage increase or decrease
float **percentage**	Percentage increase or decrease entered by the user
struct **effect_callback_struct{**	Structure used to pass information regarding the parameter to the call back function
int **ETID**	ID of the parameter
int **parameter_no**	ID of the parameter
Transact_Popup ***this_pointer**	Pointer to the object calling the call back function
} ***params**	

Protected Function

Widget **Prompt_user** (char* msg, char* msg2, char *reason)

Function Description
Prompts the user for a new value of the parameter or the percentage increase or decrease of the parameter

Argument Description

msg	Message on the prompt window
msg2	Message to enter string
reason	Whether a new value or percentage increase or decrease

Public Functions

Transact_Popup (char* ETname)

Function Description
Constructor for the Transact_Popup class

Argument Description

ETname	Pointer to the name of the ET

void **Create_Transact_Popup**()

Function Description
Creates the main pop-up menus for the transact mode expert

Argument Description
None

void **Create_Transact_Popup_Pulldown**()

Function Description
Creates the pulldown menus for the pop-up buttons in the main menu of the transact mode ET

Argument Description
None

Teach Class

Class Description
This class represents the teach mode expert.

Class Inheritance
Teach_Popup

Private Variables

Arg **al**[]	Motif argument list
int **ac**	Counter for the above list
int **current_no**	ID for the ET
char ***name**	Pointer to the name of the ET

Private Function

static void **Button_CallBack** (Widget, XtPointer call_data, XtPointer)

Function Description
: Call back function for the ET button on the main menu. Displays the pull-down menu for the teach mode ET

Argument Description

call_data	ID for the ET and the expert mode

Public Variables

Widget **button**	Button widget for the teach mode ET on the main menu
Teach ***previous**	Pointer to the previous teach mode expert in the doubly linked list maintained by the Expert_Manager
Teach ***next**	Pointer to the next teach mode expert in the doubly linked list maintained by the Expert_Manager

Public Functions

Teach (char *ETname)

Function Description
: Constructor for the teach operating mode

Argument Description

ETname	Pointer to the name of the ET

~Teach()

Function Description
: Destructor for the class

Argument Description
: None

char* **Get_Name**()

Function Description
: Returns the name of the ET that this teach mode object belongs to

Argument Description
: None

Teach_Popup Class

Class Description

This class has all the user interface functions for the teach operating mode ET

Class Inheritance

None

Private Variables

Arg **al**[]	Motif argument list
int **ac**	Counter for the above argument
Widget **popupbutton**	Button widget for the pop-up pane menu items
XmString **label**	Label for the menu items
Widget **popup_pulldown_pane**	Pop-up pull-down pane for the menu items for this operating mode

Public Functions

Teach_Popup()

Function Description

Constructor for the class. Creates all the menus for the teach mode expert

Argument Description

None

void **Create_Teach_Popup**()

Function Description

Creates the pop-up panes and the main menu for the teach mode ET

Argument Description

None

Student Class

Class Description

This class represents the Student mode expert.

Class Inheritance

Learn_Popup

Private Variables

Arg **al[]**	Motif argument list
int **ac**	Counter for the above list
int **current_no**	ID for the ET
char ***name**	Pointer to the name of the ET

Private Function

static void **Button_CallBack** (Widget, XtPointer call_data, XtPointer)

<u>Function Description</u>

Call back function for the ET button on the main menu. Displays the pull-down menu for the learn mode ET

<u>Argument Description</u>

call_data	ID for the ET and the expert mode

Public Variables

Widget **button**	Button widget for the teach mode ET on the main menu
Learn ***previous**	Pointer to the previous Student mode expert in the doubly linked list maintained by the Expert_Manager
Learn ***next**	Pointer to the next Student mode expert in the doubly linked list maintained by the Expert_Manager

Public Functions

Learn(char *ETname)

Function Description
Constructor for the Student operating mode

Argument Description

ETname	Pointer to the name of the ET

~Learn()

Function Description
Destructor for the class

Argument Description
None

char* **Get_Name**()

Function Description
Returns the name of the ET that this Student mode object belongs to

Argument Description
None

Learn_Popup Class

Class Description
This class has all the user interface functions for the Student operating mode ET

Class Inheritance
None

Private Variables

Arg **al**[]	Motif argument list
int **ac**	Counter for the above argument
Widget **popupbutton**	Button widget for the pop-up pane menu items
XmString **label**	Label for the menu items

Widget **popup_pulldown_pane**	Pop-up pull-down pane for the menu items for this operating mode

Public Functions

Learn_Popup()

Function Description

Constructor for the class. Creates all the menus for the Student mode expert

Argument Description

None

void **Create_Learn_Popup**()

Function Description

Creates the pop-up panes and the main menu for the Student mode ET

Argument Description

None

Rule Class

Class Description

This class contains the knowledge of the experts in the form of rules. These rules are maintained as linked lists inside this class. This class has been designed in detail and implemented by Scott Angster [Angs93]

Appendix B :Example Problem Listing

Overview

The list of parameters input as knowledge to the experts created are shown in the following pages. The rules in the knowledge of the experts are not shown as the rule class was not developed as a part of this thesis. The different types of rule classes were developed later by Scott Angster and examples of these class objects are listed by him [Angs93]. Listed in here is also the main program used to create the experts for the example problem.

Knowledge for the Kinematics Expert

Filename	Kinematics.kb

Knowledge :

Parameter	N1	float	0
Parameter	N2	float	0
Parameter	Omega1	float	0
Parameter	Omega2	float	0
Parameter	Ratio	float	1
Parameter	Rad1	float	10
Parameter	Rad2	float	10
Parameter	C-Dist	float	20
Parameter	Module	float	0

Knowledge for the Strength Expert

Filename			Strength.kb
Knowledge			
Parameter	KL1	float	10
Parameter	KL2	float	10
Parameter	KW1	float	1
Parameter	KW2	float	1
Parameter	KH1	float	1
Parameter	KH2	float	1
Parameter	Gear-Wid	float	10
Parameter	Power	float	100
Parameter	Dia1	float	5
Parameter	Dia2	float	5

Main Program for the Example Problem

Filename main.C

```
#include "Session_Manager.h"
#include "ET.h"


main()
{

    Session_Manager *ses_manager = new Session_Manager();

    ET kinematics("Kinematics", "Transact");

    ET strength( "Strength" )

    ET manufacturing( "Manufacturing", "Observe");

    ses_manager->handle_events();

}
```

Vita

Parasuram Narayanan was born on the 8th of February, 1967 in Tirupattur, Tamil Nadu, India. He grew up in Madras, India where he completed his high school. He completed his undergraduate studies in Mechanical Engineering from the University of Roorkee, India. After graduation he worked for the Earthmoving Equipment Division of M/S Larsen & Toubro, Bangalore, India as an engineer trainee for a year. He then decided to pursue graduate studies in Mechanical Engineering at the Virginia Polytechnic Institute and State University where he specialized in Computer Aided Design. After graduation he is interested in pursuing a career in CAD especially in the areas of geometric modeling, user interfaces and graphics

Parasuram

Parasuram A. Narayanan