An Object-Oriented, Knowledge-Based, Non-Procedural Approach to Multi-Disciplinary, Parametric, Conceptual Design

by

Scott Reed Angster

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:

Dr. S. Jayaram, Chairman

Dr. A. Myklebust

Dr. W.H. Mason

July 19, 1993

Blacksburg, Virginia

c.2

LD
5655
V855
1993
A537
c.2

Abstract

The use of computers in the area of design and manufacturing is commonplace in industry. Many companies are turning to custom designed in-house software to surpass the competition. A growing number are developing knowledge-based expert systems to capture the knowledge of expertise of employees before they retire.

The use of traditional artificial intelligence languages can be cumbersome to engineers who are usually familiar with traditional languages such as FORTRAN and C. The use of expert systems shells can often hinder the customization of an expert system due to limitations of the shell. An alternate approach to these methods is the use of an object-oriented framework that facilitates the creation of customized expert systems. This framework, called the Expert Consultation Environment, alleviates the programming problems of expert system development and allows the engineer to concentrate on knowledge acquisition.

This thesis describes the design of the rule classes needed by the framework. These are the base Rule class, the Equation Rule class, the Control Rule class, the Constraint Rule class and the Heuristic Rule Class. Also presented, is the development of a methodology used in creating an expert system with the framework. A prototype expert system developed using the framework for parametric, multi-disciplinary, conceptual design of aircraft is described.

Acknowledgments

First, I would like to thank my wife, Kate, whose support, love and patience gave me the inspiration to make it through my graduate studies. I would like to thank my parents, Dr. Jay Reed Angster and Fay Angster who have guided and encouraged me throughout my life with all their love. I would also like to thank my sister, Sandra, for her support and encouragement during this time.

I would like to thank my advisor Dr. S. Jayaram for his support and guidance throughout my graduate studies. I also would like to thank Dr. A. Myklebust and Dr. W. H. Mason for serving on my graduate committee.

I would like to thank all my friends which I have made here during my graduate studies and undergraduate studies for the great times we had.

Table of Contents

1. Introduction....1

2. Literature Review....3

    Expert Systems and Aircraft Design....3

    Expert Systems and CAD....5

    Multi-Disciplinary Design....7

3. Problem Definition and Research Objectives....10

4. Object-Oriented Design....13

5. Expert Consultation Environment....19

    As used by the programmer....19

    As used by the designer....25

6. Class Descriptions....29

    Rule class....29

    Control rule class....29

    Equation rule class....35

    Constraint rule class....35

    Heuristic rule class....37

    Parameter class....38

    Other Modifications and Additions....41

7. Knowledge Representation and Acquisition....47

8. Creating the Persistent ETs and the Expert System....................................62
Parameters..62
Equation Rules...64
Control Rules..66
9. Inference Process...70
10. Prototype Development..79
Topic Selection...79
Control Rules and Equation Rules..81
Heuristic Rules and Constraint Rules..85
Problems Encountered...87
11. Implementation and Results..93
12. Summary and Conclusions...105
13. References...108
Appendix A - Graph Representations..111
Appendix B - Knowledge Files...137
Weight.kb...137
Geometry.kb...156
General.kb..162
Propulsion.kb..163
Trajectory.kb...164
Appendix C - Detailed Class Descriptions...165
Rule Class..165

Equation Rule Class..165
Control Rule Class...166
Parameter Class..167
Consult Class Modifications..169
Consult_Popup Class Modifications..170
ET Class Modifications...171
Vita...173

List of Figures

Figure 1 Notation Used in Object-Oriented Design....18

Figure 2 Class Diagram of the ET Class Relationships....21

Figure 3 Class Diagram of the Framework....23

Figure 4 Multi-Disciplinary Design Session....24

Figure 5 Representation of the Rule Class....30

Figure 6 Inheritance Relationships of the Rule Classes....32

Figure 7 Representation of the Control Rule Class....34

Figure 8 Representation of the Equation Rule Class....36

Figure 9 Representation of the Parameter Class....39

Figure 10 Example of a Simple Set of Dependent Equations....49

Figure 11 Example of Directed Graph Representation....50

Figure 12 Directed Graph Representation of the Simple Set....51

Figure 13 Example of a Complex Set of Dependent Equations....53

Figure 14 Directed Graph Representation of the Complex Set....54

Figure 15 Decomposed Set of Graphs for the Complex Set....55

Figure 16 Cylinder Example....56

Figure 17 Directed Graph Representation of the Cylinder Example....57

Figure 18 Labeled Directed Graph Representation of the Cylinder Example....59

Figure 19 Example of a Control Equation..60

Figure 20 Revised Graph Using Control Equation Numbers..................................61

Figure 21 Knowledge File Format for the Parameter Class and Example................63

Figure 22 Knowledge File Format for the Equation Rule Class and Example.........65

Figure 23 Knowledge File Format for the Control Rule Class and Example 67

Figure 24 Example Program for Creating the Expert System.................................69

Figure 25 Example of the Inference Process...75

Figure 26 Directed Graph Representation of Selected ACSYNT Code....................82

Figure 27 Decomposed Example of the ACSYNT Code..84

Figure 28 If-Then-Else Statement from ACSYNT Code..88

Figure 29 If-Then-Else Statement Converted to Knowledge File Format................89

Figure 30 Procedural Code Example and Converted File Format...........................90

Figure 31 Example of an Iteration Process in Knowledge File Format....................92

Figure 32 C++ Program for Creating the Prototype Expert System.........................89

Figure 33 Main Screen of the Prototype Expert System...96

Figure 34 "Effect" Menu...97

Figure 35 "Cause" Menu...98

Figure 36 Determining the Current Value of a Parameter......................................99

Figure 37 Observe Mode Menu...100

Figure 38 Entering a New Value for a Parameter...101

Figure 39 Changeable vs. Not Changeable Parameters...103

Figure 40 Creating and Deleting Experts...104

1. Introduction

The use of computers by engineers and designers has become commonplace in today's technological world. The use of the computer to perform high-level mathematical calculations began in the 1960's. From the use as drafting boards, to the optimization techniques used in the analysis of entire aircraft designs, the computer has made a powerful change in the design industry.

More and more companies are using sophisticated, customized software that has been developed in house to keep up with competition. It has been shown in a survey of 26 Fortune 500 that 88% of the companies have developed their own software for computer aided design and manufacturing [Penn91]. This software is being used for drafting, visualization, finite element analysis, dynamic analysis and numerical control.

The majority of these software systems for conceptual design were created using conventional procedural programming. This limits the expandability and flexibility of the program as new technology is developed. New programming techniques, such as object-oriented programming allow for easy modification as technology changes. This flexibility and expandability, alone, does not give a company the advantage over the competition.

It is through the use of expert systems that many companies are trying to get the edge on the competition. Expert systems allow companies to retain the expertise of retiring designers and train new designers at a faster pace. This results in a more rapid design process, saving millions of dollars in product development time.

Many expert systems are developed from scratch, using languages such as Prolog or Lisp. Some are developed using expert systems shells such as Nexpert and IBM ESE. Both methods have their advantages and disadvantages. Those developed from scratch often take over a year to develop due to the complexity of the task in creating all the different aspects of an expert system. Engineers must have an expert knowledge in Lisp, Prolog, etc. to develop the expert system. Also, losing these engineers for such a long period of time can be detrimental. However, the finished product will be customized to the needs of the company. Those developed from expert system shells can be fully operational in much less time, perhaps a few months. These, however, will exhibit limitations that can not be overcome due to the limitations of the shell used.

This thesis discusses an object-oriented, knowledge-based, non-procedural approach to multi-disciplinary, parametric conceptual design.

2. Literature Review

Expert Systems and Aircraft Design

The complexity involved in the design of aircraft is growing at an enormous rate. New technology is being developed every day. To keep up with this growth of technology, expert systems are being coupled with the latest mathematical techniques. There have been several systems developed specifically for the design of aircraft.

Kroo and Takai [Kroo88] have developed an interactive framework for the analysis and optimization of aircraft. This system is based on a quasi-procedural concept which consists of procedurally written FORTRAN functions called in a non-procedural method. The idea behind this system is to eliminate the connections between functions that would be found in large procedural programs, but to maintain the structure of the analysis routines. The system works on a request-driven, backward chaining method. When the user makes a request for a variable, the system determines which functions are needed.

This quasi-procedural analysis method also contains a forward chaining expert system. This expert system warns the user on possible design errors and responds with

suggestions. The errors are displayed when the constraint error function is called by the analysis function in the case of constraints being violated or from the user calling the constraint error function directly to check for any violations prior to analysis. The addition of optimization techniques to this system is discussed in [Kroo92].

Mark Kolb [Kolb88] has developed a conceptual design aid using object-oriented programming and constraint propagation. This design aid was used in a program called Rubber Airplane, which is used for the conceptual design of aircraft. This system is written in LISP using an extension for object-oriented programming. Kolb uses a component modeling approach in his design aid system. Each component of the object being designed is described in terms of its attributes and constraints. The attributes are parametric quantities and the constraints are the mathematical relationships between the attributes. Constraints between components are modeled using design links. These allow the value of one component attribute to affect the value of other attributes of the same component or other components.

Bouchard, Kidwell and Rogan [Bouc88] discuss the use of artificial intelligence in the area of aeronautical design. A detailed description of parametric design and parametric design automation are given. The descriptions of the ideal automated design system and a realistic design system are given by the authors.

Bouchard [Bouc92] describes the Engineer's Scratch Pad (ESP) that was developed by Lockheed based on the ideas presented in [Bouc88]. ESP is designed to perform multi-disciplinary parametric trade studies on a customized application. The user is required to enter in equations for calculations and data-flow diagrams. Equations

and the parameters that are used in the equations are entered via text files called datasets. Datasets are read into the ESP and converted to the necessary form. Data-flow diagrams specify analysis procedures that link datasets, other data-flow diagrams or conventional programs together. Various optimization techniques are available in ESP. This system has been used in the design of individual components of vehicles as well as the entire vehicle conceptual design. ESP was written in LISP and has been developed over the last four years.

Tong, Powell and Goel [Tong92] present Engineous, a system that integrates expert systems with numerical methods for design and optimization of aerospace systems. Engineous contains many different optimization utilities that are used in the field of design, including mathematical-based and logic-based approaches. These utilities can be combined in any fashion by the user during the design session. Engineous can be coupled to existing analysis codes and perform optimization on a sequence of analysis codes. It written in Common Lisp, FORTRAN and C.

Expert Systems and CAD

The ability to capture the expertise of designers is valuable in all areas of design. Therefore, the use of expert systems in almost any area of mechanical design can be found in the literature.

Morse and Hendrickson [Mors91] have developed a framework for multi-disciplinary design utilizing an object-oriented blackboard structure. The GUIDE

model (globally unified interactive design and evaluation) was designed around two main sub models, a process model and a semantic model.

The process model is the problem solving framework that contains the interaction between the various domain experts. There are four main parts to the process architecture. These are the global solution database, containing the evolving design solution, the design operators, containing the design specialists and associated knowledge, the conflict resolver, containing methods for resolving conflicts among the design specialists, and the control elements, which are the monitors to invoke the conflict resolver and a control-scheduler which regulates the individual design operators.

The semantic model is a conceptual organization for the data needed to fulfill the requirements for the multi-disciplinary design. This model contains five classes of design information. These are the data attributes, data interdependencies, compatibility conditions, commitment level and evaluation metrics. Data attributes are the operational data needed for computation.

Rosenman, Gero, Hutchinson and Oxman [Rose86] discuss the expert system shell BUILD which was developed to be used in a wide variety of design applications. It is written in Quintus Prolog and incorporates both forward and backward chaining. BUILD allows for both design analysis, performance evaluation of a given system, and design synthesis for a desired performance.

Reinschmidt and Finn [Rein90b] discuss the integration of expert systems with computer aided design and computer databases. An explanation of expert systems, computer aided design systems and database management systems is given. The

integration of the three systems as they pertain to design in today's engineering market, including examples is also discussed. The future importance of knowledge base systems, in the field of engineering and design, is further discussed by Reinschmidt [Rein90a].

Ullman and Dietterich [Ullm87] discuss the implications of mechanical design methodology on knowledge-based systems and computer aided design. This discussion of the lack of understanding of mechanical design methodology is followed by an example of an ongoing research project in this area. Criteria needed for linking computer aided design and knowledge-based systems to follow a design methodology, as seen by the author, is also discussed.

Jayaram and Myklebust [Jaya89, Jaya90, Jaya93] suggested a high-level environment which will support the creation of customized expert systems for computer aided design and manufacturing applications.

Narayanan [Nara93] has developed a framework that facilitates the creation of customized expert systems. This framework is designed for use with parametric, multi-disciplinary design applications.

Multi-disciplinary Design

The increasing complexity in design has led to the need for a more effective use of design methodologies and techniques. The incorporation of multi-disciplinary design into the early stages of the design process has been seen as vital.

Elias [Elia86] describes the development and implementation of an aid in aircraft design called Paper Airplane. Paper Airplane is designed to take a set of input variables and functions and automatically generate design paths based on the requested variable output by the user. Parameters are given default values, operating units, and minimum and maximum values. Paper Airplane will notify the user when a parameter goes above or below the suggested limits. Paper Airplane is written in Lisp and does not support a graphical interface or optimization techniques. These modifications were added into the Rubber Airplane system discussed earlier.

Jenkinson and Simos [Jenk84] present a program designed to analyze General Aviation, Twin-Engined, Propeller-driven aircraft called GATEP. This program is designed to compare two different designs, perform sensitivity analysis, and assess the impact of future technology. GATEP uses a main controlling routine that calls the necessary functions to perform the requested analysis. Parametric studies can be analyzed by using the plotting capabilities of GATEP.

Wampler, Jayaram, Myklebust and Gelhausen [Wamp88a, Wamp88b, Jaya92] describe ACSYNT in a series of papers. ACSYNT or AirCraft SYNThesis is a parametric, multi-disciplinary design aid for the conceptual design of aircraft developed by NASA and Virginia Tech. ACSYNT allows the user to perform analysis operations such as weight and aerodynamic performance estimations, on parametrically designed aircraft models. Optimization can be performed on these models using COPES (COntol Program for Engineering Synthesis). COPES was written to take any analysis code and treat it like a subroutine. This enables the subroutines to be called any number of times

in any order to optimize a given set of design parameters for a given objective. COPES can perform a number of optimization techniques such as sensitivity analysis and approximation theory. ACSYNT is written in FORTRAN, C and C++.

Sobieszczanski-Sobieski and Tulinius [Sobi91] discuss the importance and complexity of multi-disciplinary design. Without multi-disciplinary design, key variables that can dramatically affect the performance of the aircraft, are often left untouched until late in the design process. The cost of redesigning, at such a late stage is too high. If these variables are analyzed in the early stages of conceptual design, the multi-disciplinary "chain of influences" can be calculated and optimized.

There are numerous papers available in the field of multi-disciplinary design. The few that are mentioned here pertained to the work presented in this thesis.

3. Problem Definition and Research Objectives

As important employees in the field of engineering and design retire, valuable knowledge and expertise is lost. In a competitive market of high technology, the loss of such expertise can destroy a company. Many companies are turning to expert systems to capture this knowledge. Through the use of expert systems, much of this expertise can be saved and utilized. However, the time-consuming task of developing an expert system from scratch can often discourage companies from using this valuable technology and the use of expert system shells often do not allow the flexibility for a truly customized expert system.

To develop an expert system from scratch, the knowledge of sophisticated languages, such as Prolog and LISP, and programming techniques, such as object-oriented decomposition and design, chaining algorithms and search methods are often required. Engineers are typically not familiar with these methods and languages. They are different from the traditional languages such as FORTRAN and C. Besides the task of learning all this, the programmer must worry about all parts of the expert system. These include the inference engine, the user interface, and the knowledge base. The sophisticated routines that the expert system must emulate, such as multi-disciplinary

design and design optimization, can be very demanding on a programmer using a new language. If an experienced programmer is used to develop the expert system, instead of an existing expert in the field, one problem, the language barrier, is eliminated. However, this often leads to another problem. It is often very difficult for a programmer to develop a customized expert system in a field they know little about. The ideas and methods that must be incorporated into the expert system may be difficult to convey. Therefore, companies may prefer to use existing employees to develop the expert system, but use expert system shells. These allow a programmer to use the shell interface to write the expert system. This eliminates the need for the programmer to learn any languages or techniques. However, the programmer is limited to the structure and interface provided by the expert system shell. The routines and methodologies used in the field of design are very complex and may not fit into an existing shell. This can cause compromises in the expert system due to the limitations of the expert system shell.

One alternative to these two methods is the use of a high-level environment which would allow the creation of customized experts systems in the field of parametric multi-disciplinary design using a language which is commonly used by engineers [Jaya89, Jaya90, Jaya93]. The development time of an expert system using such a framework would resemble that usually found using an expert system shell.

By using this framework, the programmer does not have to be concerned with the user interface or the inference engine. These are built into the framework. The user just has to understand the principles behind the different types of rules that are available in the framework and follow the format for creating the expert system. This allows the

programmer to concentrate on the knowledge acquisition part of developing the expert system and not the coding.

One such expert system development framework, using C++, allowing for fully customized expert systems such as those developed in artificial intelligence languages has been designed by Parasuram Narayanan [Nara93]. Narayanan's work involved the design and partial implementation of the classes needed for the framework. The key classes that were implemented during his work were the Session_Manager class, the Expert_Manager class, the ET class, the Consult class, and all the classes for the user interface. The design and implementation of these and other classes can be found in [Nara93].

The objectives of the research described in this thesis are as follows:

1. Demonstrate the use of this framework by developing a prototype expert system in the field of parametric, multi-disciplinary design of aircraft
2. Design and implement additional classes needed by the framework
3. Modify existing framework classes based on the creation of the prototype system
4. Define a methodology that can be used to develop an expert system using the framework

4. Object-Oriented Design

The complexity of the problems being solved by software has caused the complexity of the software to become unmanageable, unexpandable and inflexible. The need to simplify the problem as a set of smaller more manageable problems has resulted in the use of objected-oriented design methods.

To simplify a complex system, one must break the system down into smaller more manageable parts, or decompose the system. This has been performed for a long time using algorithmic decomposition, in which each module represented a step in the overall process. This, however, can lead to very unmanageable, inflexible and unexpandable code. By using object-oriented decomposition, this does not occur due to the inherent nature of the method. Instead of breaking the system down based on algorithms, object-oriented decomposition breaks the system down around key abstractions, or objects, in the domain. This type of decomposition is modeled around real-world situations in which each abstraction represents an object in the real world. The focus of object-oriented design is the creation of a system using object-oriented decomposition.

The building block of object-oriented design is the object. An object contains all the characteristics of itself that makes it unique from other objects. Objects also contains member functions that allow the object to perform actions on itself as well as the ability to send messages to other objects. These messages tell other objects to perform their associated member functions.

Booch [Booc91] describes an object model in object-oriented programming as containing four major elements that are essential to object-oriented design and three minor elements that are useful but not essential. The four required elements of an object model are abstraction, encapsulation, modularity, and hierarchy. The three minor elements of an object model are typing, concurrency and persistence.

Abstraction: Abstraction is the means in which we see a system and simplify the behaviors and properties of the system to a key few that we can comprehend. These behaviors and properties are those that each individual feels are the most important or significant. In terms of programming, the behaviors and properties are converted to member functions and information that distinguish that type of object from all others.

Encapsulation: Objects that use the functions of another object are called client objects. The client object only knows the behavior, or interface, of the object it is using, not the methods in which the behavior is performed. This property is know as encapsulation. This ability to hide the implementation of a behavior of an object is the property of object-oriented design that allows for easy modification of a program.

Modularity: As previously mentioned, object-oriented decomposition breaks down a large system into smaller systems called objects. This alone is known as

modularity. However, most object-oriented languages allow for an additional means of breaking down a large system at a higher level than the object called modules. The use of modules in object-oriented design allow the programmer to group together objects that work together to perform a larger more general behavior. Links between the modules are established so the modules can "talk" to each other.

Hierarchy: The last major element of object-oriented design is hierarchy. Hierarchy is the ranking of the abstractions. The most important type of hierarchy is inheritance. Inheritance allows a programmer to declare a class as a "kind of" another class. This will be discussed in more detail later.

Typing: Typing is used to formalize the characteristics and behaviors of the abstractions. The word "class" is often used in place of "type" in object-oriented programming. By using types, or classes, one can declare a set of objects to be of the same type. Therefore, they all exhibit the same behaviors, but still have a unique identity.

Concurrency: Concurrency is the property that allows each object that we define to act simultaneously and independently of each other. Each object has the ability to perform functions and exist while other objects are doing the same.

Persistence: The last element of object-oriented design is called persistence. Persistence is the concept that objects can exist even after the method that was used to create the object ceases to exist. This is found mostly in object-oriented databases.

The seven elements of object-oriented design can be combined and utilized using an object-oriented programming language such as C++. By using C++, or similar

languages, the power of object-oriented design can be used by the programmer to develop programs that are highly sophisticated, yet very manageable and expandable. The following is a description of additional terms and ideas associated with object-oriented programming.

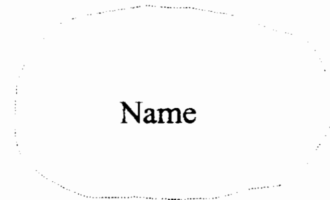
Classes: In C++, objects are defined as instances of a class. A class contains all the attributes and member functions associated with the abstraction that it represents.

Inheritance: Inheritance was alluded to previously in the section on hierarchy. Inheritance is a type of hierarchy used to rank or order classes. The top level class, or base class, contains certain characteristics. Classes that are declared as subclasses of the base class, inherit these same characteristics without the need to redefine them. Additional characteristics that are unique to the subclass can be added.

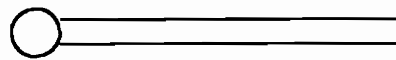
Polymorphism: Polymorphism is a term that is associated with the idea of encapsulation. Each class or object exhibits its own behavior. The implementation associated with the behavior is known only to the class itself. Therefore, the same behavior for two different classes may be carried out in two totally different ways. For instance, in an object-oriented CAD system, there may be class for circles, triangles and lines. Each class may have a function to draw themselves on the screen called Draw. If the user asks the system to draw a line, the system draws a line. Likewise, the system will draw a circle when given the command draw circle. At the programming level, the interface passes a message to the line class or circle class, to run the function Draw. The message to the two classes are the same, the function name is the same, but the result is different. This is polymorphism. The idea that a group of classes, generally a set of

subclasses of a base class, perform the same type of operation in different ways.

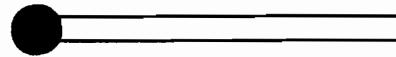
In Fig. 1, the Booch representation of a class diagram and the notations for the relationships between classes is presented [Booc91].



Class



Uses for interface



Uses for implementation

Inherits

Figure 1. Booch Notation for Object-Oriented Design

5. Expert Consultation Environment

As Used by the Programmer

An expert system development framework has been developed to facilitate the creation of an expert system for parametric design [Nara93]. It was designed around the suggested requirements for a Expert Consultation Environment (ECE) [Jaya89, Jaya90, Jaya93]. This ECE is part of an overall Application Programming Interface, CADMADE (Computer Aided Design and Manufacturing Application Development Environment), to support a programmer of CAD/CAM systems in the development of applications in the least amount of time. This section describes the use of the Expert Consultation Environment by a programmer to create an expert system that would be used by a designer or engineer.

The suggested ECE would aid a programmer in the development of an expert system by supplying the programmer with a set of high-level programming tools. These tools contain the inference engine, the knowledge base structure and methods to support forward and backward chaining within the knowledge base. The programmer, therfore, does not need to develop these aspects of the expert system, but only needs to supply the

knowledge base itself. The expert system development framework links the supplied knowledge base with the other aspects of expert systems mentioned earlier to develop a full functioning expert system.

The expert system development framework, allows the programmer to set up the expert system to function in several modes of operation. These modes are designed to make a design session using the ECE emulate the different types of interactions between a designer or engineer and an expert that could be present in a real-world design session. These modes are as follows:

- Transact
- Consult
- Observe
- Learn
- Teach

The Transact mode is designed as a single question - single answer mode. The Consult mode is a multiple question - multiple response mode. The Observe mode is designed as a warning and suggestion mode. The Learn mode is used to add knowledge to the knowledge base, whereas the Teach mode is used to let the expert system guide a user through the design process.

Since the idea of an expert system is to emulate a real world situation, working one on one with an expert, the framework is built around an Expert Technician (ET), as can be seen in Fig. 2. This ET is created by the programmer and contains all the parameters and domain knowledge that an expert in that field would know. Since in a

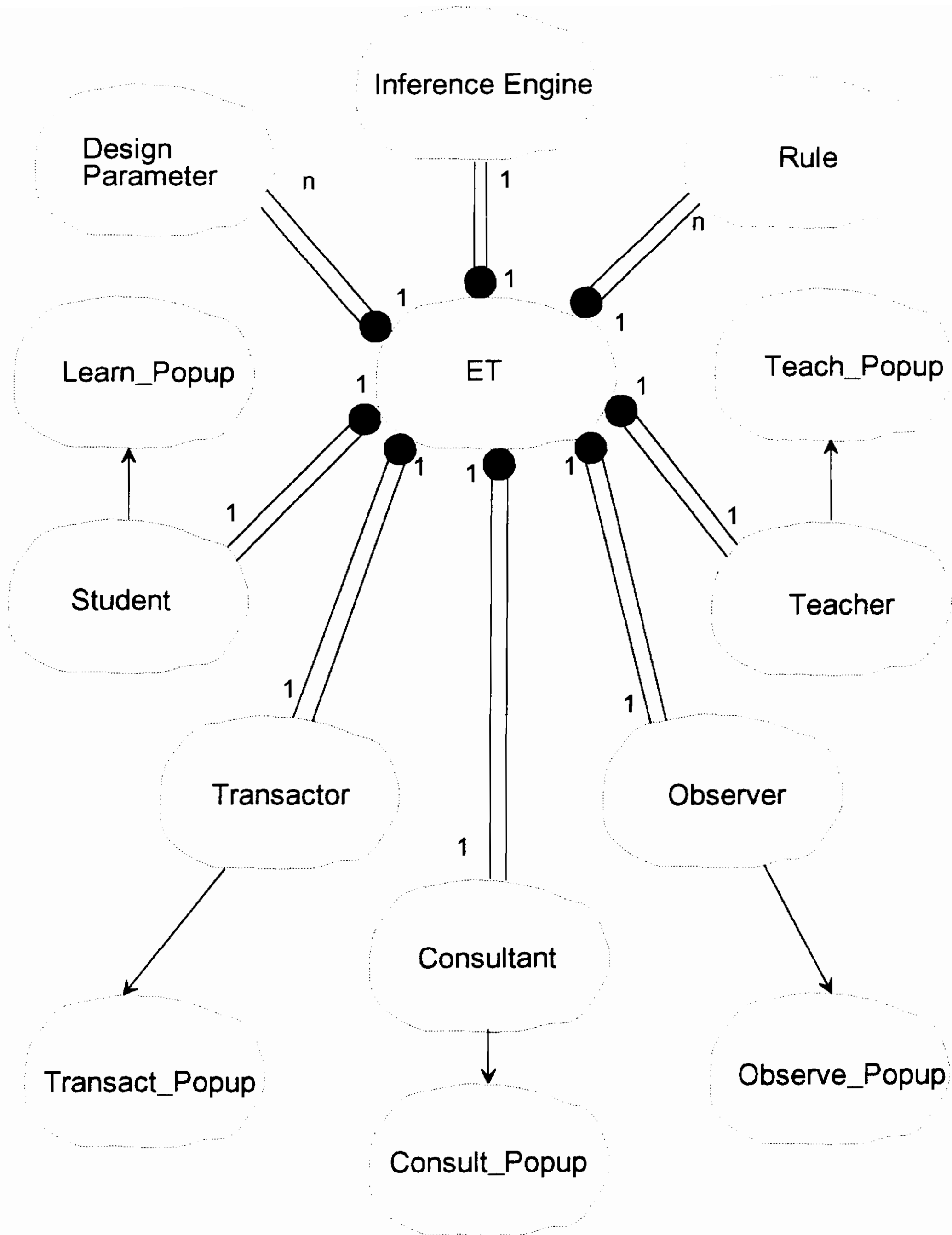


Figure 2. Class Diagram of the ET Class Relationships [Nara93]

real world situation, the expert would know how to use the knowledge and add to it, the ET utilizes several other classes to perform the needed operations. These include the modes of operation classes and the inference engine class.

The object-oriented design of the rest of the framework is partially shown in Fig. 3. Multiple ETs can be created with each containing its own knowledge. These are all managed by an instance of an Expert Manager class which handles the interaction between the experts. All communication between experts is handled through this class. The Expert Manager class uses an instance of the Session Manager class to keep track of multiple sessions of the framework that can be running at the same time. The Session Manager inherits the Session Interface class. This contains all of the applications for the user interface. Therfore, by creating multiple experts, using them in various modes of operation, the framework gives the programmer the ability to set up a design session that emulates the real world. This is depicted in Fig. 4.

The only other responsibility left for the programmer is the creation of an instance of the Session Manager class. This class manages the entire expert system that is developed using the tools of the framework. The Session Manager is responsible for the creation of an instance of the Expert Manager class that oversees the interaction between the experts, as well as using the functions of the Session Interface class to generate the user interface.

As can be seen, the high-level tools provided by the framework take the burden of developing the inference engine and the user interface off the programmer. This allows the programmer to concentrate on the knowledge acquisition part of setting up an expert

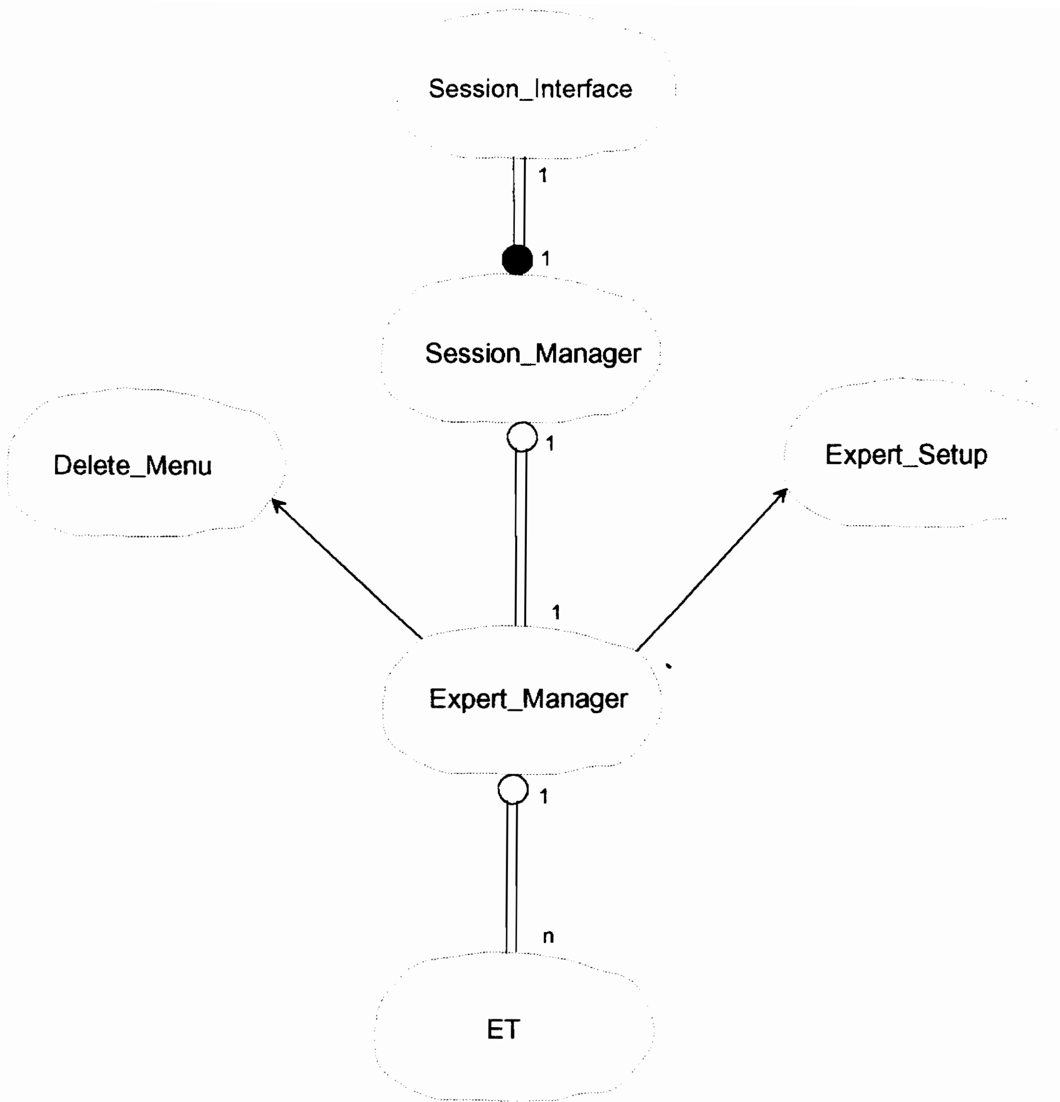


Figure 3. Class Diagram of the Framework [Nara93]

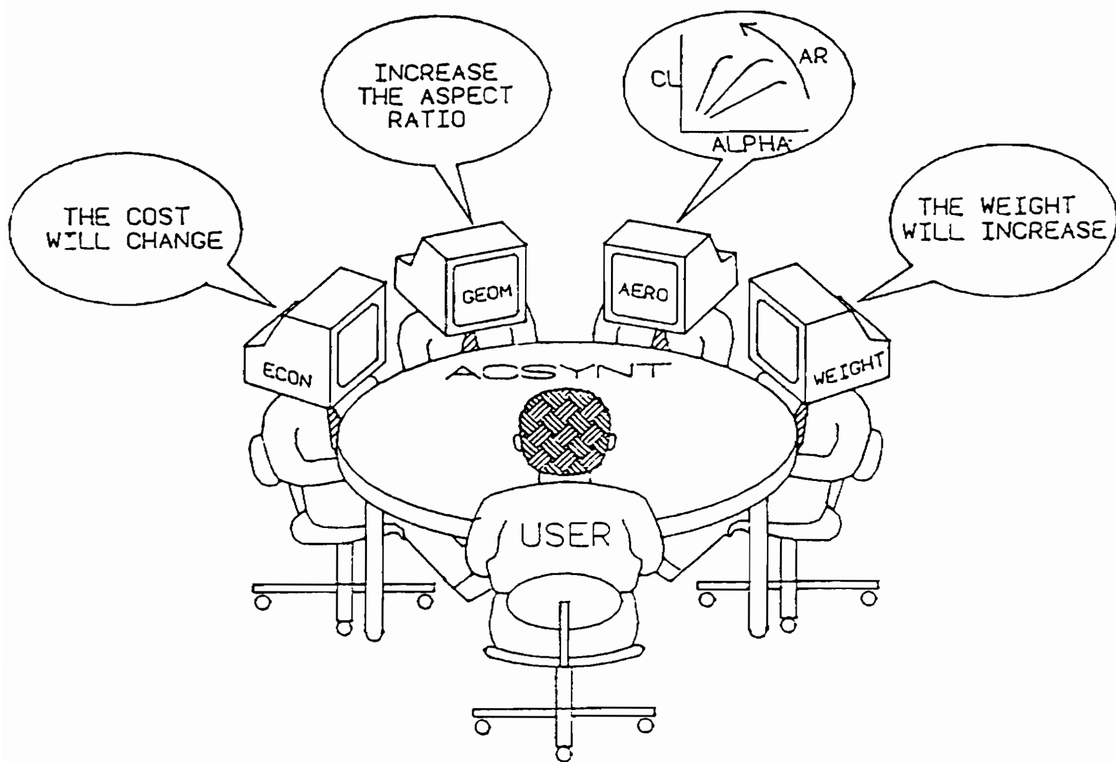


Figure 4. Multi-Disciplinary Design Session

system. Refer to Narayanan [Nara93] for a detailed description of the expert system development framework.

As Used by the Designer

Once the programmer has designed an expert system with the ECE, a designer can use the expert system for conceptual design. This section describes how a designer could utilize the created expert system.

As mentioned earlier, there are several modes of operation that can be used by the designer to exploit the power of the expert system. These are the consult mode, the transact mode, the observe mode, the teach mode and the learn mode.

The consult mode is designed as a multiple question - multiple response mode. The user poses questions to the expert system in areas such as the effects of changing parameter values or what will cause a parameter value to increase. This mode of operation is very interactive between the user and the expert system. The consult mode will allow the designer to find out everything that changes due to a change in parameter value. This tracking of a domino effect, can take place across the various ETs that exist. Thus, multi-disciplinary effects can be watched. In the consult mode, the user can perform the following list of actions:

What is: Tells the user what the current value of a selected parameter is.

Why: This selection responds with the rule path

that was generated in the calculation of a selected parameter.

How: This choice displays which experts were used in the reply to the query

Effect: This lets the user see the results of changing a variable. When selected, another menu pops up with what type of actions can be performed on a parameter. These include maximize, minimize, % increase, % decrease, new value and optimize.

Cause: This lets the user ask the expert system what will cause a given effect on a selected parameter. The menu that pops up includes increase and decrease.

Determine: Recalculates the value of a parameter. A list of parameters is provided that the user can select from.

The transact mode is very similar to the consult mode. However, it is designed to be only a single question and single response mode. It will not track the chaining effect of changing a parameter value. It only will tell the designer the immediate effects. The list of choices available in the transact mode are the same as those in the consult mode.

The observe mode is used as a watch and advise mode. The main use of this mode will be constraint checking and design guidance. As the designer is changing values of parameters, the observe mode is watching to make sure that no violations have occurred. These violations can be in the area of code constraints, design constraints or design violations. Design violations are mistakes that previous designers have made that do not necessarily break any mathematical rules but are entered as heuristic knowledge for other designers to watch out for. If a violation has occurred, the designer is notified, and alternative solutions will be proposed.

The learn mode is designed to capture the knowledge of the designer. There are several ways in which this can be performed. These include forcing knowledge into the knowledge base by entering rules to be added to those that exist, modifying existing rules that are already in the knowledge base or posing questions to the designer during the design process.

Once an expert technician has been created, the addition of rules to its knowledge is essential. The easiest way of doing this is via a user interface to allow the designer to "fill in the blanks" with the information that is required for each of the different kinds of rules. The ET will then take the entered information and convert it to the correct format for the specified kind of rule. This eliminates the need for the designer to understand the structure of the knowledge base rules as required by the programmer.

The need may arise for the information contained in a rule to be changed. An equation used may no longer be valid, or the value used in a constraint rule may need to be modified. This will also be performed using a user interface to separate the designer

from the programming task.

The last method for entering knowledge is performed during the design process. As a designer is making changes, the expert system poses questions, such as "Why was this done?", for the designer to answer. This could be answered by typing the response in, or an alternate solution of voice entry could be used. The reason for voice entry, is to make the process less obtrusive to the designer. The design session that is performed using this means of knowledge entry may be a mock design, purposely used for knowledge extraction, or an actual design.

The teach mode is designed to act as a teacher for the novice designer or student. This mode would allow a designer or student to step through a design process while being guided by the expert system.

All the features and modes described above have not been fully implemented. Please refer to Narayanan [Nara93] for a detailed list of implemented features.

6. Class Descriptions

For a programmer using the expert system development framework, there are several classes that are of key importance. These include the Rule class, the Control Rule class, the Equation Rule class, the Constraint Rule class, the Heuristic Rule class and the Parameter class. These classes are used to hold all of the knowledge that each ET owns. In the sections that follow, a description of the design of each of the Rule classes and the modifications to the Parameter class, described by [Nara93], is given. In this design, the details of the data members and member functions are defined.

Rule Class

The information contained in the base Rule class is essential to the operation of the expert system and can be seen in Fig. 5. However, it does not contain any specific rule information. It is inherited by all the other rule classes. It contains the flag for active versus inactive rules, data needed by the specific rule types and a pointer to the next rule in the linked list of rules. There is only one function that is associated with the Rule class and it is to return the active/inactive state of the rule.

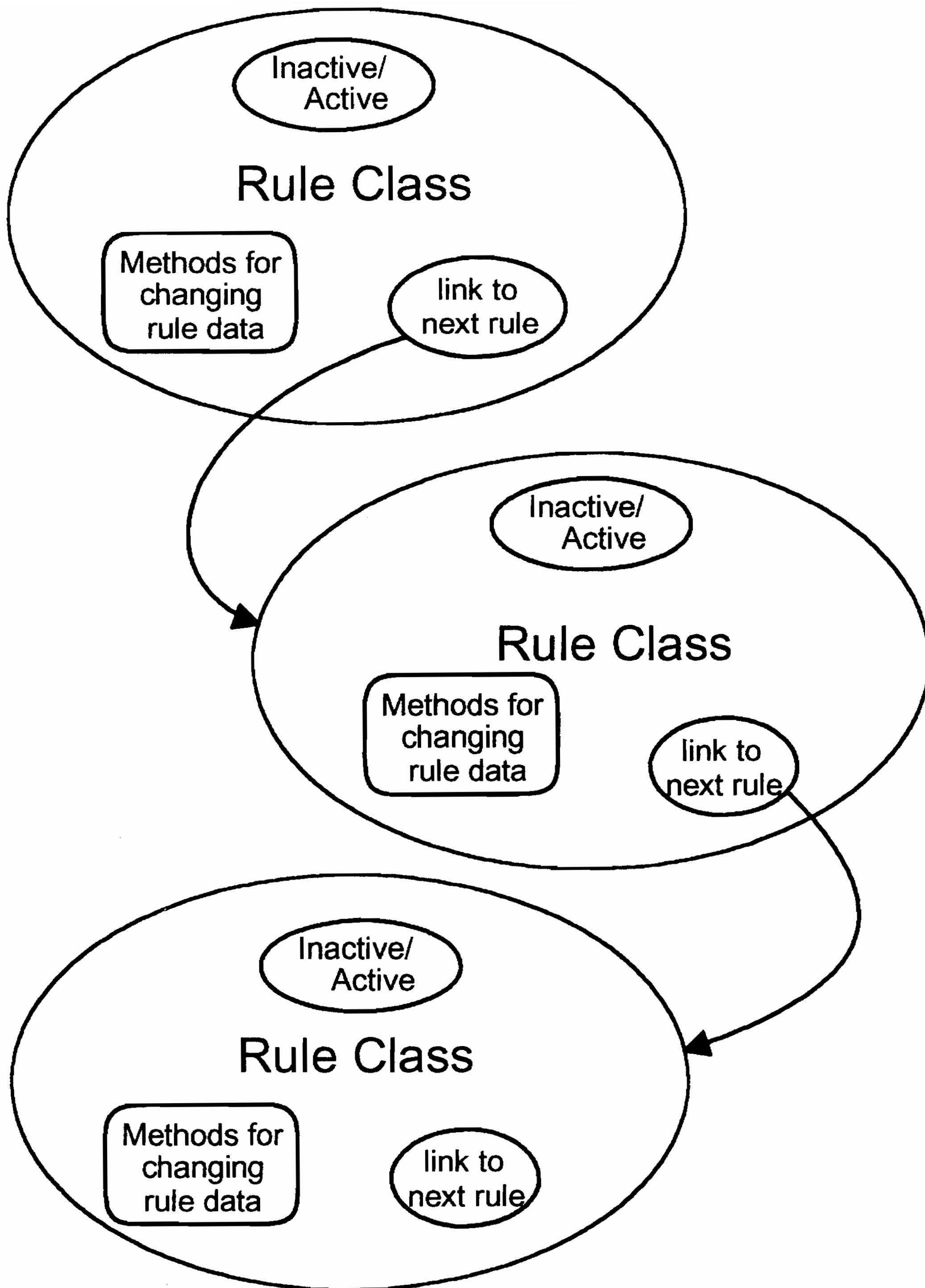


Figure 5. Representation of the Rule Class

The flag for the active/inactivate state of a rule will be used by the inference engine to check which rule should be used for a given variable or action. It is very common for more than one rule pertain to the same parameter. In this case there will be a governing control rule than will decide which rule should be used and activate it by setting this flag to the active state. This will be discussed in more detail under the topic Control Rule Class.

The Rule class contains various data needed by the specific rule types. These are the Action structure and the Equation structure. These structures allow the rule types to be very flexible.

The Action structure is used by the Equation Rule class and the Control Rule class. These data members are used for specifying what type of action is to be performed and what rule or function it should be performed on. The types of actions that can be performed are Activate_Rule, Deactivate_Rule, Fire_Rule and Call_Function.

The Equation structure is used by the Equation Rule class. It contains the data needed by the inference engine concerning equation rules. These include the parameter the equation will solve for, the equation to be used, a list of parameters that are used in the equation and the number of parameters in the list.

As mentioned earlier, this base Rule class is inherited by several other classes. This can be seen in Fig. 6. The classes that inherit the Rule class are the Control Rule class, the Equation Rule class, the Constraint Rule class and the Heuristic Rule class.

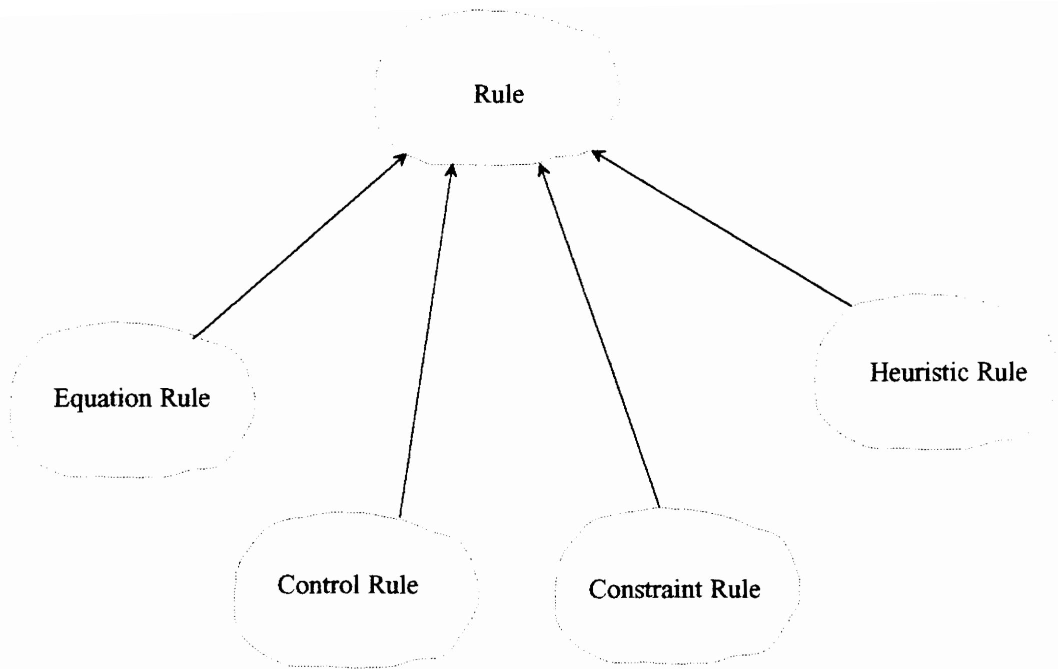


Figure 6. Inheritance Relationships of the Rule Classes

Control Rule Class

The Control Rule class is used as a decision making tool for multiple rules which gonern the same parameter. Which rule is used will be based on the values of other key variables. This is the job of the Control rule. It analyzes the situation and performs one or more "actions" based on the input. This action is contained in an Action structure.

There are several types of actions that can be performed in a Control Equation Rule. These are Activate_Rule, Deactivate_Rule, Call_Function and Fire_Rule. It is up to the Control Equation to activate the right rule and deactivate the others. This is performed with the Activate_Rule command and the Deactivate_Rule command. Often, the user may want to see the result of a change of a parameter in terms of a graph instead of numerical values. This will be performed using the Call_Function command. This command can also be used for invoking special pieces of FORTRAN or C code. The last command, Fire_Rule, gives the user the ability to override the automatic chaining that occurs in the inference engine. The user may want to change a value based on some input or perform a set of actions in a specific order (procedural coding). This is done using the Fire_Rule command.

This rule class is set up in an "If Then" type of format. The control equation is a Boolean equation and can only take on the value of true or false. If it is true, the inference process will perform a certain action, if it is false it will perform a different action. The Control Rule class contains data members, as shown in Fig. 7, for the following: a name for the rule to be used by other rules, the Boolean equation and two

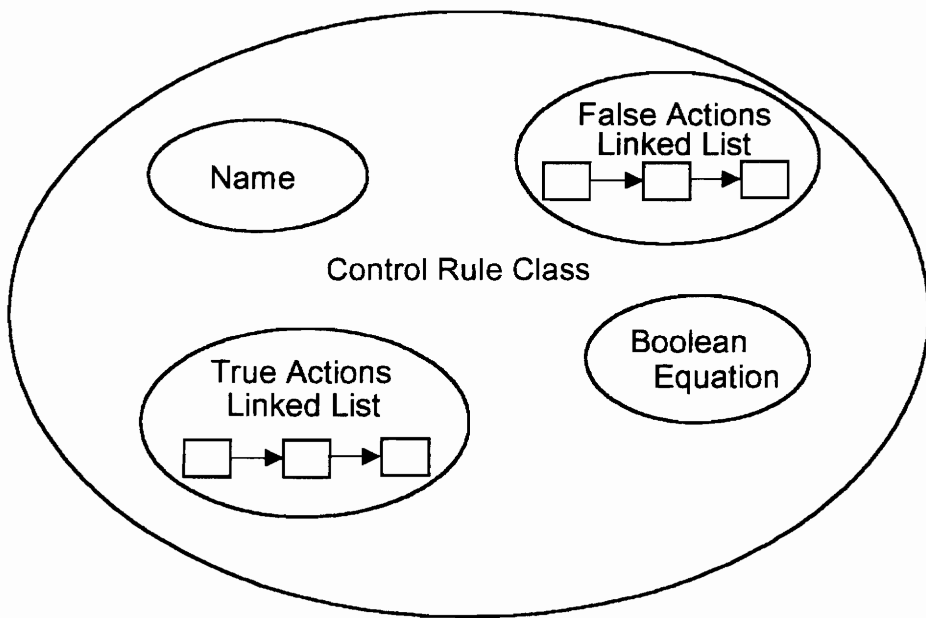


Figure 7. Representation of the Control Rule Class

independent linked lists of Action structures, one linked list of actions for the true response and one for the false response.

Equation Rule Class

The Equation Rule class is set up to solve for a given parameter or a set of parameters using algebraic equations. When the inference process needs to calculate the value for a parameter it will fire the active rule for that parameter. Once fired, the rule calculates the value and sends the value back to the parameter for use by the inference process.

The Equation Rule class consists of a name for the rule, for reference by other rules, and a linked list of equations that will be evaluated when the rule is fired. There is also a linked list of Actions to be performed when the rule is fired. This is shown in Fig. 8.

Constraint Rule Class

The Constraint Rule class is used to set up constraints in the design. During the process of design, the value of a parameter is often limited to a specific range based on another parameter's value or the type of design being performed. These are not constraints on the design due to the limitiations of the program. These constraints are entered by the programmer to aid the designer under certain design conditions. It is the

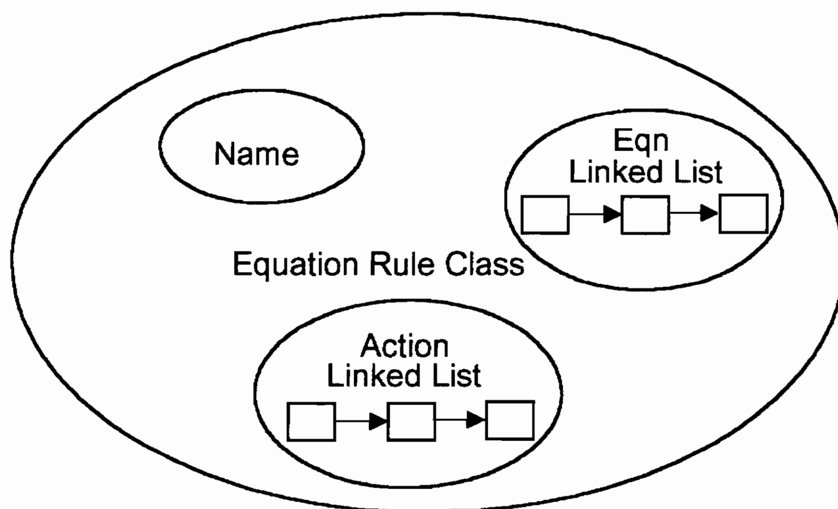


Figure 8. Representation of the Equation Rule Class

objective of the Constraint Rule class to monitor these constraints set up by the programmer, and warn the user if any are being violated.

The class contains the name of the parameter which the constraint pertains to as well as the constraint value itself. The name of other variables and corresponding values that are checked to invoke a constraint are contained in a linked list.

Heuristic Rule Class

Heuristic rules are rules of thumb that are used in a design of a system. They are usually developed over a period of time and are used by experts in design without thinking about it. This is the most difficult type of knowledge to capture. However, it is probably the most important. It is this knowledge that is lost when a senior engineer or designer retires or leaves a company.

Heuristic rules often answer questions that begin with the phrase, "What if." These types of questions are frequently asked by a designer in the conceptual design stages. The result of changing the value of a parameter is often known by an experienced designer without the need to calculate the chaining effect. Therefore, the purpose of the heuristic rule in this framework, is to answer the "what if" questions without recalculating. Heuristic rules will have priority over equation rules that solve for the same parameter. If there are no heuristic rules that can answer a "What if" question, a forward chaining process will be invoked recalculate the parameters affected by the changed parameter. By having a heuristic rule set up, time will be saved since this

calculation will not have to be performed.

Other types of rules that can be encoded using the Heuristic Rule class are past mistakes and company specific rules. By storing past mistakes, new designers can avoid making costly errors that have already occurred. Often, companies have their own rules and knowledge with respect to cost and time estimations, availability of resources, existing manufacturing facilities, etc., that new designers are not familiar with. These can also be stored as heuristic rules.

Parameter Class

The Parameter Class, depicted in Fig. 9, is used to declare the design parameters that will be used by the knowledge base. This class contains all data that is needed in the evaluation of a design as well as the methods that are used to access the data stored as this type of class. The data stored includes the name of the parameter, the type of parameter, such as float or string, the current value of the parameter, which expert owns the parameter, whether the parameter is user changeable or internal and the minimum and maximum values of the parameter based on code limitations. Since each expert will contain a linked list of all of the parameters that it owns, the parameter class also contains a pointer that points to another instance of the parameter class.

As mentioned previously, functions belong to the parameter class to access the data contained in the class. The public functions created by Narayanan:

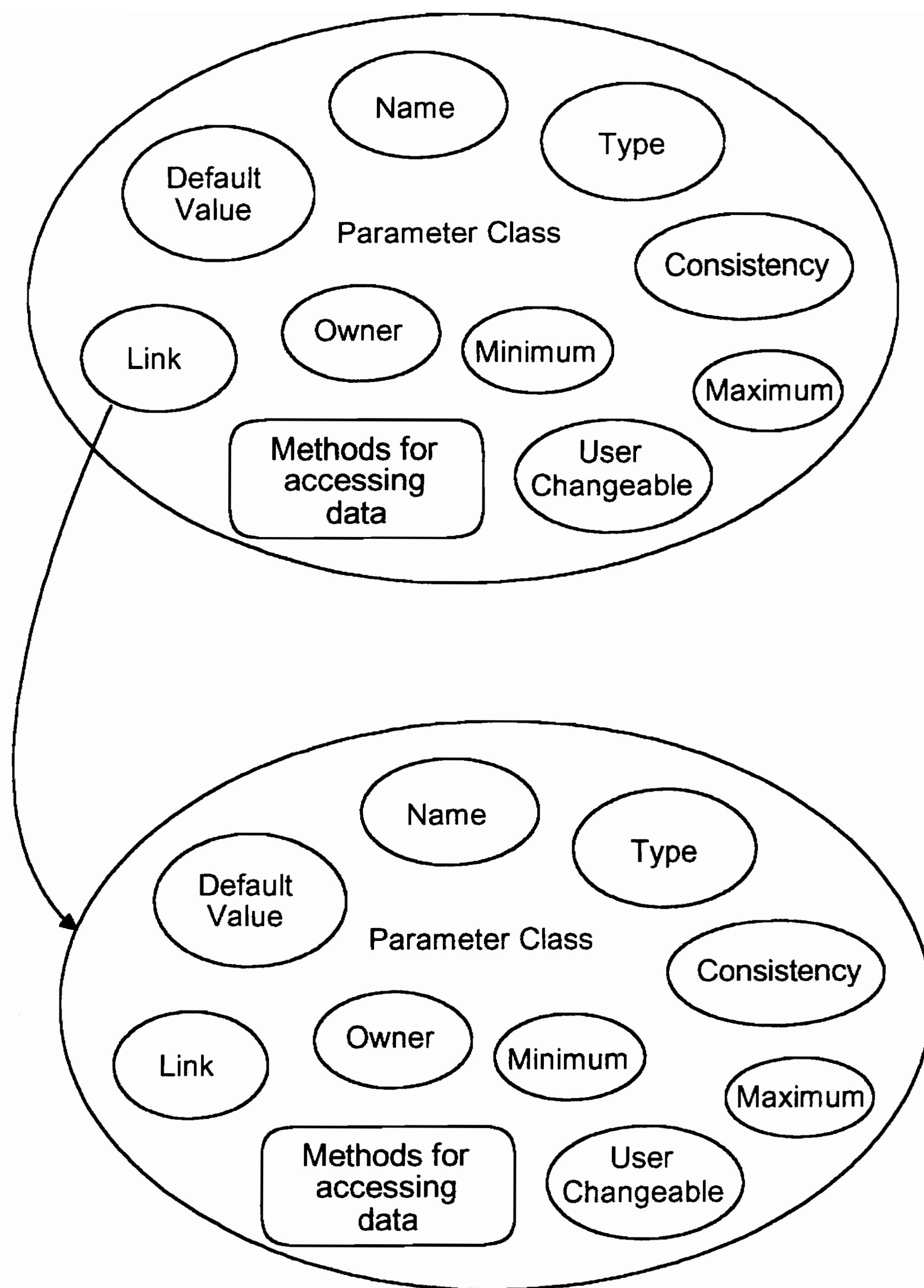


Figure 9. Representation of the Parameter Class

get_type	Returns the type of the parameter
get name	Returns the name of the parameter
get_value	Returns the current value of the parameter if the parameter is a float
get_char_value	Returns the current value of the parameter if the parameter is a string This is not overloaded with get_value since there are no arguments used by the function.
parameter	Class constructor

Public functions added or modified:

put_value	Sets the value of the parameter This is overloaded to set either a float or string value.

get_max	Returns the maximum allowable value of the parameter
get_min	Returns the minimum allowable value of the parameter
get_changeable	Returns the changeable status of the parameter

One other key member of parameter class is a flag, the validity flag. This flag can take on two values, "valid" or not "valid". If this flag is set to "valid", it means that the value of the parameter is consistent with all parameter values that are directly involved in determining its value. If the inference engine sees that this flag is "valid" it does not need to recalculate the value. This will save time in the calculation process. This flag will be set to "not valid" whenever any of the parameters that determine it are either changed or set to "not valid".

Other Modifications and Additions

As mentioned previously, the Expert Consultation Environment was designed by Parasuram Narayanan [Nara93]. However, with the additions of the previously described

classes to the ECE, several modifications and additions had to be made to existing classes. The following section will describe these modifications to the ECE. The full description of the original classes can be found in [Nara93].

Consult Class

The Consult class was modified to allow for the interaction between the user and the developed expert system. This involved the addition of five functions to the class:

Param_Effect_Increase	Callback function that allows the user to see a list of parameters when the "effect-increase" option is selected from the user interface
Param_Effect_Decrease	Callback function that allows the user to see a list of parameters when the "effect-decrease" option is selected from the user interface

Param_Effect_New_Value	Callback function tha allows the user to see a list of parameters when the "effect-new value" option is selected from the user interface
Param_Effect_Minimize	Callback function tha allows the user to see a list of parameters when the "effect-minimize" option is selected from the user interface
Param_Effect_Maximize	Callback functionthat allows the user to see a list of parameters when the "effect-maximize" option is selected from the user interface

Consult_Popup Class

The Consult_Popup class was modified to allow a list of parameters to be displayed upon the selection of menu option from the effect menu. This involved the addition of two new functions, one modification of a function and a new data definition for this class.

Create_Consult_Popup_Pulldown	Modified to allow for pulldown menus to be displayed off of the menu items
Create_Changeable_Pulldown	Creates the user interface that contains a list of parameters that can be selected by the user
Get_Changeable_List	Gets a list of parameters that have been designated changeable to be put into a pulldown menu
Effect_callback_struct	Data structure that was needed to pass information through the callbacks for the effect menu

There were also several variables that were added to this class. These can be found in Appendix C.

ET Class

The ET class was modified in several ways. These modifications were necessary for the expert system to read in the knowledge files (persistent ET object) correctly and to access the data contained within a parameter. In this class, four functions were added and three were modified.

ET	The constructor for this class was modifid to initialize new variables.
Read_KB	Modified to read in the modified Parameter class Modified to read in the Equation Rule class Modified to read in the Control Rule class
Add_Parameter	Modified to add a parameter to the linked list of parameters with the modifications made to the class
Add_Rule	Adds a control rule to the linked list of rules of an ET
Put_Value	Puts a new value into a parameter's current value

Get_Max_Value	Returns the maximum value of a parameter
Get_Min_Value	Returns the minimum value of a parameter

There were also several variables added to the class. These can be seen in Appendix C.

7. Knowledge Representation and Acquisition

In the design of an expert system for mechanical design, there must be a methodology used. The methodology used during this research went through several stages of development. The one described here is the final version that was implemented during the creation of the prototype expert system described later.

The first step in developing an expert system is the selection of an area of design that requires or would benefit by the use of expert systems. Prerau [Prer85] discusses the basic requirements for this domain. These include a domain characterized by expert knowledge, the recognition of experts in the domain and a finished system which will benefit the industry. One of the key needs for expert systems, as discussed by Dvorak [Dvor91], is the capturing of expertise that will be lost due to the loss of an engineer or designer through retirement.

After the specific domain has been chosen, the task of knowledge acquisition must be performed. This can be done in a variety of ways. The first involves talking directly to the experts in the domain chosen. This type of knowledge most likely will develop into heuristic rules or rules of thumb. This type of knowledge is invaluable. In many design expert systems, however, reliable and accurate design equations are also

needed. This can be found in research literature and design handbooks. One other place that this knowledge can be found, is in existing analysis routines and code. There has been extensive work in most engineering design domains that have developed very good analysis routines. Very often however, they are embedded in very large, hard to understand code. The joining of good analysis routines and heuristic knowledge should develop an expert system knowledge base that will prove worthwhile to the industry.

Up to this point, the steps taken have been common to all types of expert systems during the initial development stages. The next stage, though, is geared toward the use of the framework. It involves the symbolic representation of the knowledge that has been acquired so that it can be entered in the knowledge base as dictated by the framework.

The first step is to identify all parameters that pertain to the design. This includes all parameters that are constants, user-defined parameters, and parameters that are just used in intermediate calculations.

Next, the relationships between these parameters should be symbolically represented. If the system is simple, such as that shown in Fig. 10, the method used involves writing all parameters out on paper and connect them using arrows. This convention, called a directed graph representation [Jaya92a] is shown in Fig. 11. If a parameter directly affects another parameter, an arrow is drawn from that parameter towards the one that is being affected. The resulting graph for the simple system shown in Fig. 10 is shown in Fig. 12. This graph can easily be interpreted and the relationships between the parameters can be seen.

$$H = D - G$$

$$D = A / B + C$$

$$G = E * F$$

Figure 10. Example of a Simple Set of Dependent Equations

$$Z = X + Y$$

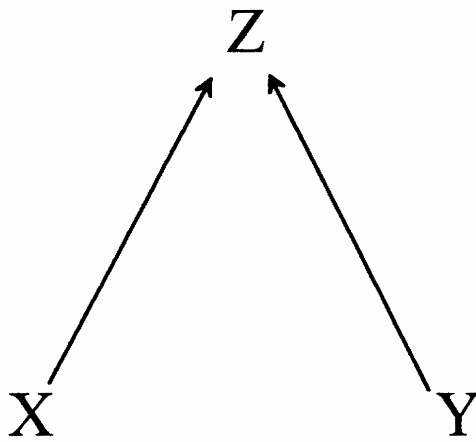


Figure 11. Example of Directed Graph Representation

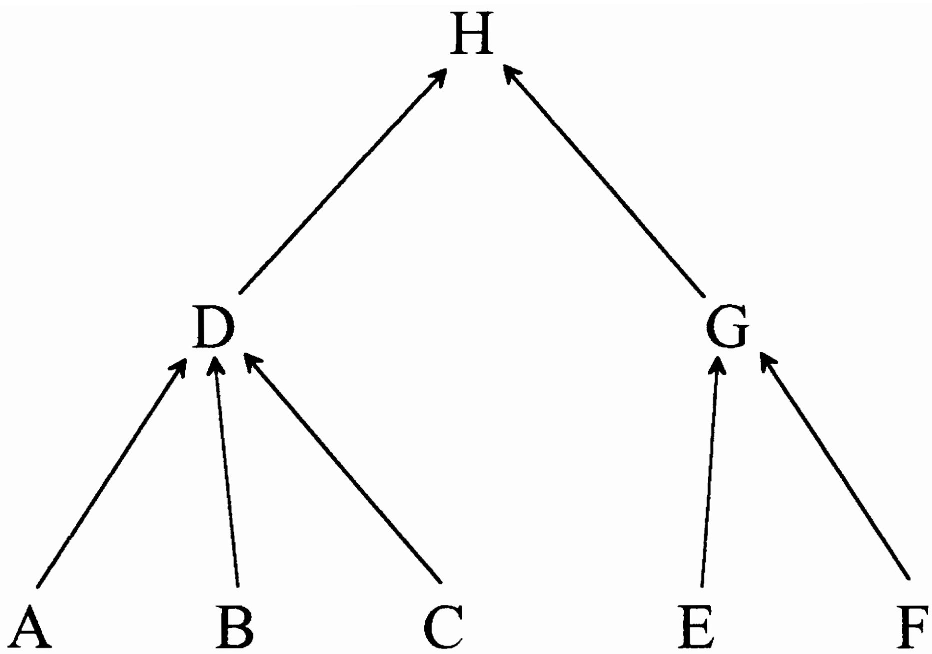


Figure 12. Directed Graph Representation of the Simple Set

If the system is complicated, such as that shown in Fig. 13, then the resulting graph, shown in Fig. 14, is difficult to follow. This is due to the bi-directional arrows often occurs as shown in the figure. The alternative method, is to decompose the graph down into smaller, more easily readable graphs. There should be a graph for each parameter that is determined from other parameters. In other words, one for every parameter that is not a leaf node in the tree. The resulting set of graphs for the complex example can be seen in Fig. 15. Although the relationship between all the parameters is not as easily interpreted, the end result is the information that is needed by the framework for the knowledge base. This information is the set of parameters that immediately effect a given parameter. The rest of the relationships will be taken care of by the inference process of the expert system.

The use of simple arrows works well in the case where there is only one equation that can be used. However, in the case where multiple equations can be used, an addition to the arrow method is needed. Take the trivial case of calculating the volume of a cylinder, shown in Fig. 16. If the user knows the radius and height, the equation used would be

$$V = \pi r^2 h$$

However, if the user only knows the diameter, the equation would be

$$V = \pi d^2 h / 4$$

The graph for this situation is shown in Fig. 17. This looks like both the diameter and the radius are needed to solve for the volume, whereas this is not the case. The addition of numbering the arrows and equations solves this problem. This is called labeled directed graph. The numbers identify which group of parameters are needed to solve for

$$G = C - D / E + J$$

$$C = A + B - D$$

$$A = C - B + D$$

$$D = (B * E) - (K * F)$$

$$E = H / I$$

$$F = I - K$$

$$B = A - H$$

$$H = (A * J) / I$$

Figure 13. Example of a Complex Set of Dependent Equations

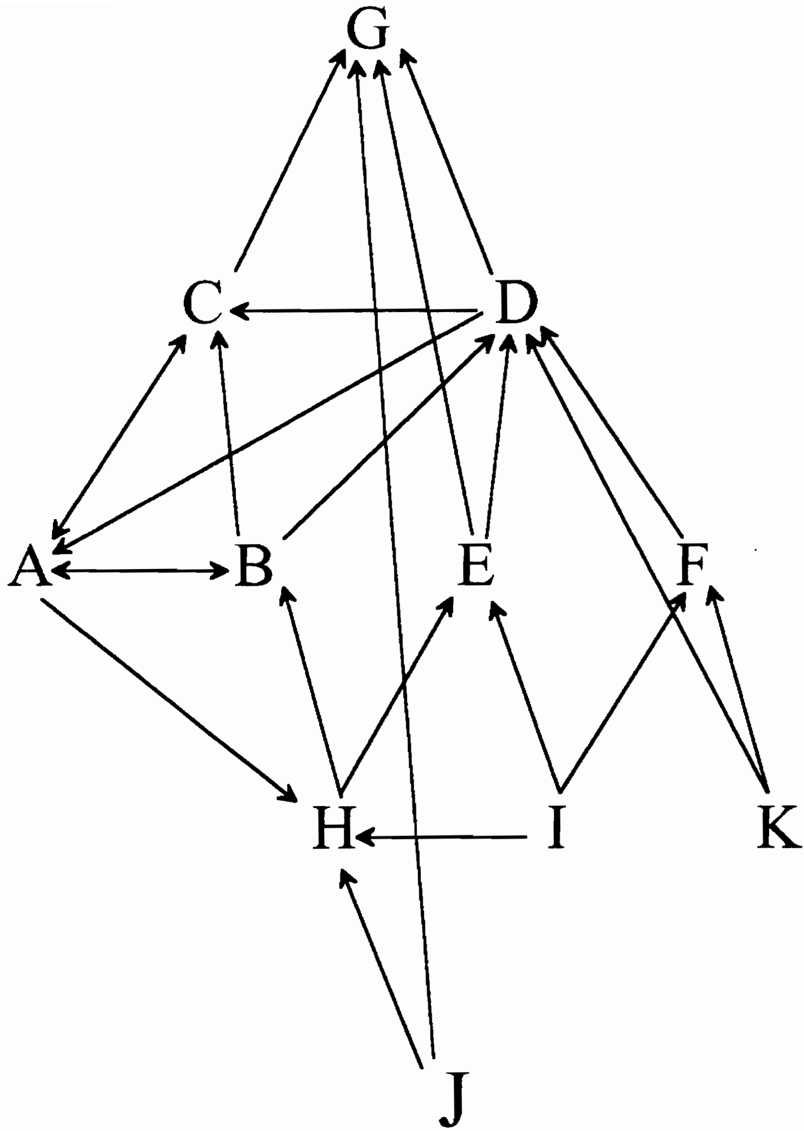
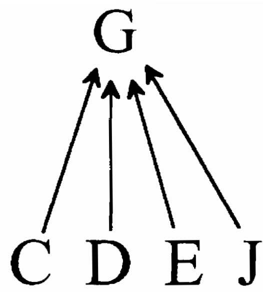
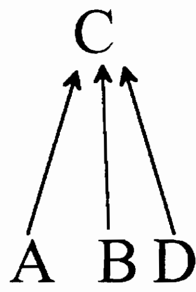
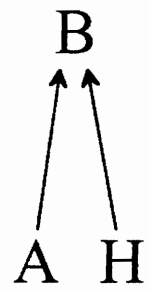
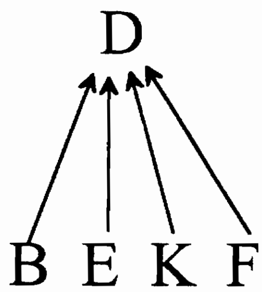


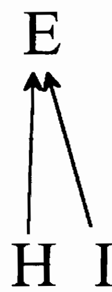
Figure 14. Directed Graph Representation of the Complex Set

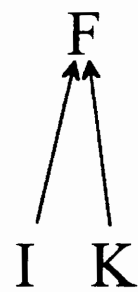


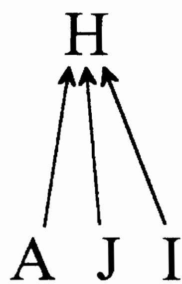












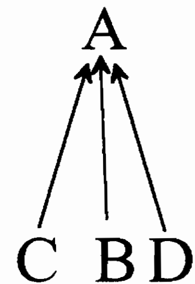
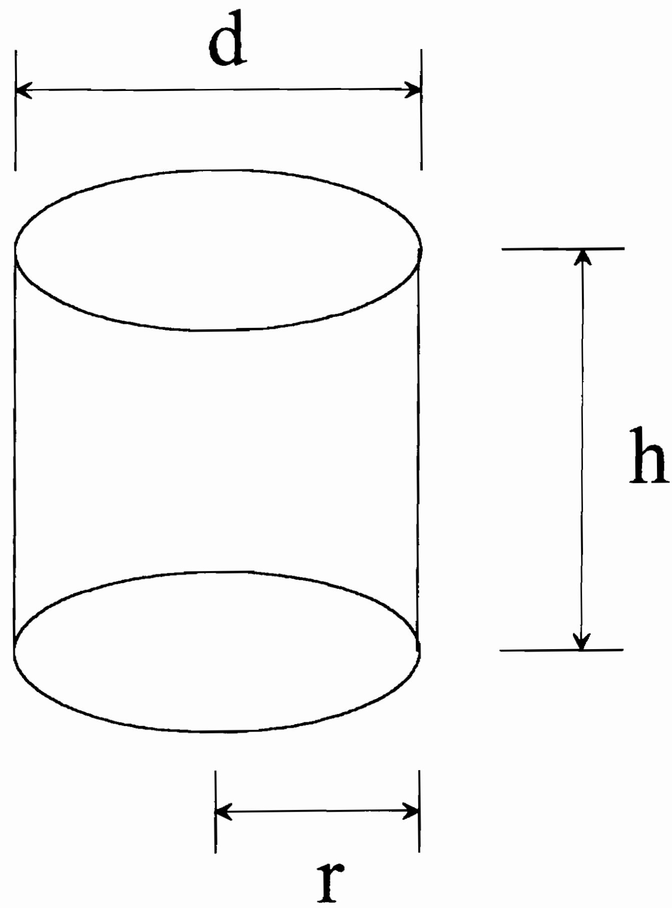


Figure 15. Decomposed Set of Graphs for the Complex Set



$$V = PI * r^2 * h$$

$$V = 1/4 * PI * d^2 * h$$

Figure 16. Cylinder Example

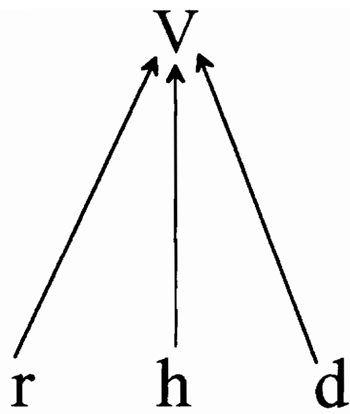


Figure 17. Directed Graph Representation of the Cylinder Example

the parameter. This is shown for the cylinder example in Fig. 18.

This method works well until the case where another parameter dictates what equation should be used. This can be seen in Fig. 19. This additional parameter has an arrow pointing to the initial parameter since it directly affects it. However, knowing this parameter's value alone does not give the expert system sufficient knowledge to solve for the required parameter. Another addition to the representation method is needed. This is where the development of control rules described in the previous section took place. By labeling the control equation parameter as a control equation, as shown in Fig. 20, it can be seen that additional knowledge, besides the control rule parameter, is needed to solve for the parameter.

The representation methods discussed above, will allow the programmer to symbolically represent all parameters and associated equation rules and control rules. Additional representation techniques must be developed for constraint rules and heuristic rules.

Now the knowledge is in a format that is compatible with that of the expert system development framework, i.e. parameters, equations rules and control rules. The next step is to associate all parameters and their governing equations with a discipline or domain expert. In the cylinder example used earlier, the volume could be associated with a geometry expert. If the stress on the walls due to a pressurized gas inside the cylinder is being calculated, these calculations and the stress may belong to a strength expert. This type of separation of parameters into different experts is the multi-disciplinary aspect of design that is built into the framework.

1 $V = PI * r^2 * h$

2 $V = 1/4 * PI * d^2 * h$

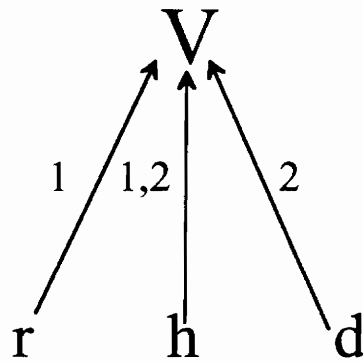


Figure 18. Labeled Directed Graph Representation of the Cylinder Example

```
1       If a > 0
2           b = c + d
        Else
3           b = e - f
```

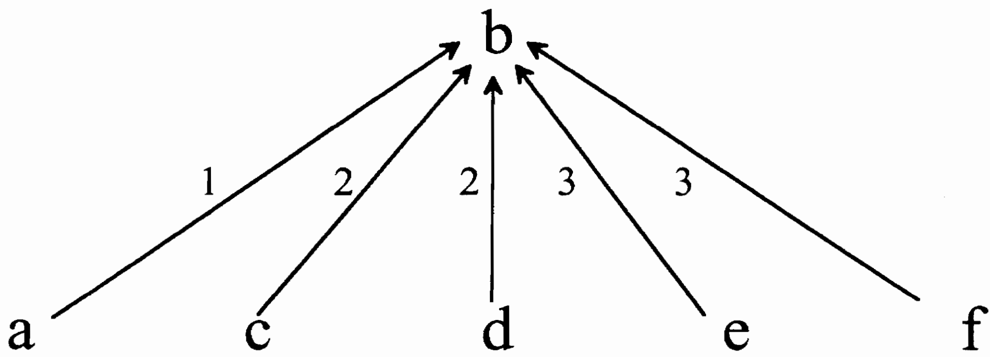


Figure 19. Example of a Control Equation

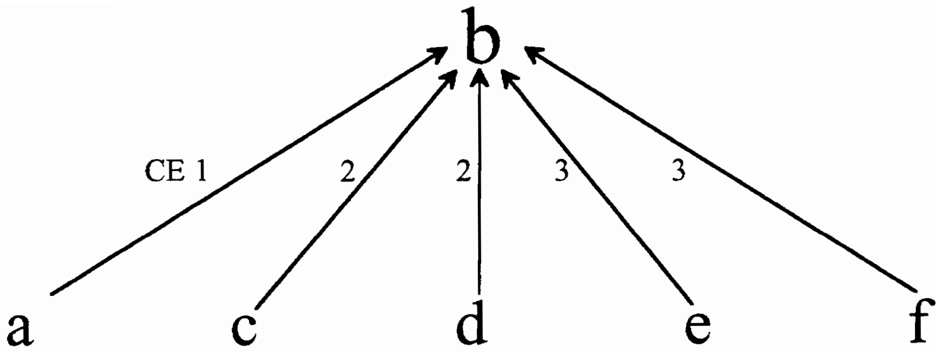


Figure 20. Revised Graph Using Control Equation Numbers

8. Creating the Persistent ETs and the Expert System

Once all of the knowledge represented in the format needed by the framework, the next step is to define the knowledge for use by the expert system. This is done by creating a set of knowledge files, which represent the persistent ET objects. Each one represents the knowledge of one of the expert technicians that will be used in the expert system. The knowledge file for each ET contains the parameters owned by that ET and rules that govern those parameters. The parameters and rules are entered using a format that is read by the expert system. This format is an abbreviated English format that is clear to read and understand. These files are read into the expert system and converted into a form that the expert system uses. The format for each of the classes previously discussed is now presented.

Parameters

The format for the Parameter class can be seen in Fig. 21. As will be seen in the other classes, the first word of the line tells the expert system what class is being entered. In this case, the word Parameter tells that the following infonnation is a parameter that

Parameter <tab> Name <tab> Type <tab> Default <tab> User_Changeable ↵

Parameter SWG Float 400 1 ↵

Figure 21. Knowledge File Format for the Parameter Class and Example

belongs to a specific expert and a Parameter object is to be created. Following a tab, the name of the parameter is entered. This is the name that will be used by all rules. Following another tab, the type of parameter is entered. Types that can currently be used are String and Float. Next, the default value for the parameter is entered. This is put in after another tab. Following the default value is another tab. Then, the minimum and maximum values are entered, separated by a tab. A sample parameter entry into the knowledge base can be seen in Fig. 21.

Equation Rules

The Equation Rule class is indicated if the word Eqn_Rule starts the line in the knowledge file. The structure of this class can be seen in Fig. 22. Following a tab, the name of the Equation Rule is given. This can be any character string up to twenty-five characters. After another tab, the number of equations that are contained in the rule is given. This can be any integer number. After this integer, a new line character, or "return" is present. On the next line, the number of parameters that are on the right side of equation is given. This is needed so the system can set up an array of parameters for use by the inference process for the consistency check. Following another tab, the equation is given. This is written using C code syntax, separating each part of the equation by a space. After the equation, another return is needed. If there is more than one equation, the same format follows for the next equation. After all equations are entered, the following line contains the word Action, followed by a tab and number

Eqn_Rule <tab> Name <tab> #_of_Eqns

#_of_parameters <tab> Eqn_1

#_of_parameters <tab> Eqn_2

Action <tab> #_of_actions

Type_of_action <tab> Name

Type_of_action <tab> Name

Eqn_Rule Rule_SPAN 1

2 SPAN = sqrt (SWG * ARWG)

Action 0

Figure 22. Knowledge File Format for the Equation Rule Class and Example

telling the expert system how many Actions should be performed when this rule is called. On the next line, the type of action is entered followed by a tab and the name of the rule or function which the action is to be performed on. Each action to be performed is placed on its own line. The type of actions and their meaning is described in the Control Rule class section of Chapter 6. This rule is followed by a blank line. An example of this rule can be seen in Fig. 22.

Control Rules

The Control Equation class is determined by the key word Control_Eqn. The structure for this class is similar to that of the Equation Rule class and can be seen in Fig. 23. After the key word and a tab, the name of the rule is given. Again, the name is limited to twenty-five characters. After a tab, a Boolean equation is given, following the same rules as that of the equation in the Equation Rule. On the next line, the word THEN is entered followed by a tab and the number of actions that are to be performed if the Boolean equation is true. The next line, or lines, contain the type of action and name in the same format as in the Equation Rule class. After the THEN actions, the next line contains ELSE, a tab, and the number of actions to be carried out if the Boolean equation is false. The next line or lines contain these actions. This rule is followed by a blank line. A sample can be seen in Fig. 23.

Once all the parameters and rules are specified in the necessary files, there is only one more step in the creation of the expert system using the framework. This is the short

Control_Eqn <tab> Name <tab> Equation

THEN <tab> #_of_actions

Type_of_action <tab> Name

Type_of_action <tab> Name

ELSE <tab> #_of_actions

Type_of_action <tab> Name

Type_of_action <tab> Name

Control_Eqn CE_CLMAX CLMXTO < 0.1

THEN 1

Activate_Rule Rule_CLMAX_2

ELSE 1

Activate_Rule Rule_CLMAX_1

Figure 23. Knowledge File Format for the Control Rule Class and Example

program that contains all the information needed by the framework. This program includes the definition of the Session Manager, the definition of all the Expert Technicians and which mode they are running in, and a message to the Session Manager that controls the entire operation of the expert system. A sample program for the cylinder example can be seen in Fig. 24.

```
// include all necessary files
#include "Session_Manager.h"
#include "ET.h"
#include <STDIO.h>

main()
{
        // Declaration of Session manager
        Session_Manager  *manage_all = new Session_manager();

        // Declaration of geometry ET
        ET  geometry("Geometry",consult);

        // Declaration of strength ET
        ET  strength("Strength",transact);

        //  Call to the session manager to handle all input by the user
        manage_all->handle_events();

}
```

Figure 24. Example Program for Creating the Expert System

9. Inference Process

Two of the three areas in an expert system have been discussed in the provious chapters. These are the knowledge base and the user interface. The other key area of an expert system is the inference process. This is the part of the system that connects the user and the interface to the knowledge stored in the system. The design of the inference process was left until the other areas were complete. This was done so that the process could be tailored to the needs of a CAD system. Based on the requirements needed to support a parametric design CAD system, a number of methods or functions were identified for the inference process. These are:

flag_parameter_inconsistent	Changes the consistency flag of a given parameter to "inconsistent" and sends a message to the expert manager of the change
flag_parameter_consistent	Changes the consistency flag of a given parameter to "consistent"

broadcast_inconsistent_parameter	Sends a message from the expert manager to all the expert technicians that a given parameter has been changed to inconsistent
broadcast_parameter_change	Sends a message from the expert manager to all the expert technicians that a given parameter has been changed by the user
parameter_value_changed	Sends a message to the expert manager that the user has changed a value of a parameter
locate_inconsistent_parameters	Locates parameters that are affected by a given parameter labeled inconsistent within each expert technician
find_heuristic	Finds all heuristic rules that govern a given parameter
find_equation	Finds equation rules that govern a given parameter

find_control	Finds control rules that govern a given parameter
find_constraint	Finds constraint rules that govern a given parameter
fire_heuristic	Performs the action specified by an active heuristic and/or displays the contents of the rule
fire_equation	Recalculates and stores the value or values of any parameter in the rule and calls the function to perform any actions located in the rule
fire_control	Checks the Boolean equation for True or False and performs the actions under either the THEN or ELSE
fire_constraint	Checks the constraint rules and displays the constraint being violated

need_value	Sends a message to the expert manager that a given parameter value is needed
broadcast_need_value	Sends a message to all ET that a given parameter value is needed
get_value	Gets the current value of a parameter if the parameter is consistent
rule_status_message	Sends a message to the expert manager that the active/inactive flag of a given rule should be set to either active or inactive
broadcast_rule_status	Sends a message to the expert technicians that the active/inactive flag of a given rule should be set to either active or inactive
check_status	Used to check the active/inactive status of a rule
check_consistency	Checks to see if the current value of a a parameter is consistent

use_default	Uses default value of a parameter if no rules are active or can be activated
find_parameter	Checks to see if a given parameter is in an ET

These functions are combined to form the inferencing process of the expert system development framework. A sample inferencing process is depicted in Fig. 25. In this example, the user has requested that the value of parameter A be recalculated by selecting "Determine" from the menu for ET 1 in the Consult mode. The first process performed by the inference engine is to find the equation that calculates the value of A. Once it is found, the status of the equation is checked to make sure it is active. In this example, there is no governing control equation, so the equation is active and it is fired. In order to calculate A, the value of B is now needed. The inference process first checks the current ET, ET 1, for the parameter B. Since the parameter is located in ET 1, it does not need to broadcast out to the Expert Manager that the value is needed. It then checks the consistency of the parameter to make sure the value is valid. In this case, the value of B is consistent, so the value is used. Next, the parameter C is needed. ET 1 does not own parameter C so a message is sent to the Expert Manager that the value of C is needed. The Expert Manager then broadcasts this message to the other ETs. ET 2 receives this message and tries to find the parameter C. It is located and checked for consistency. The value is then sent back to the Expert Manager where it is sent to the ET

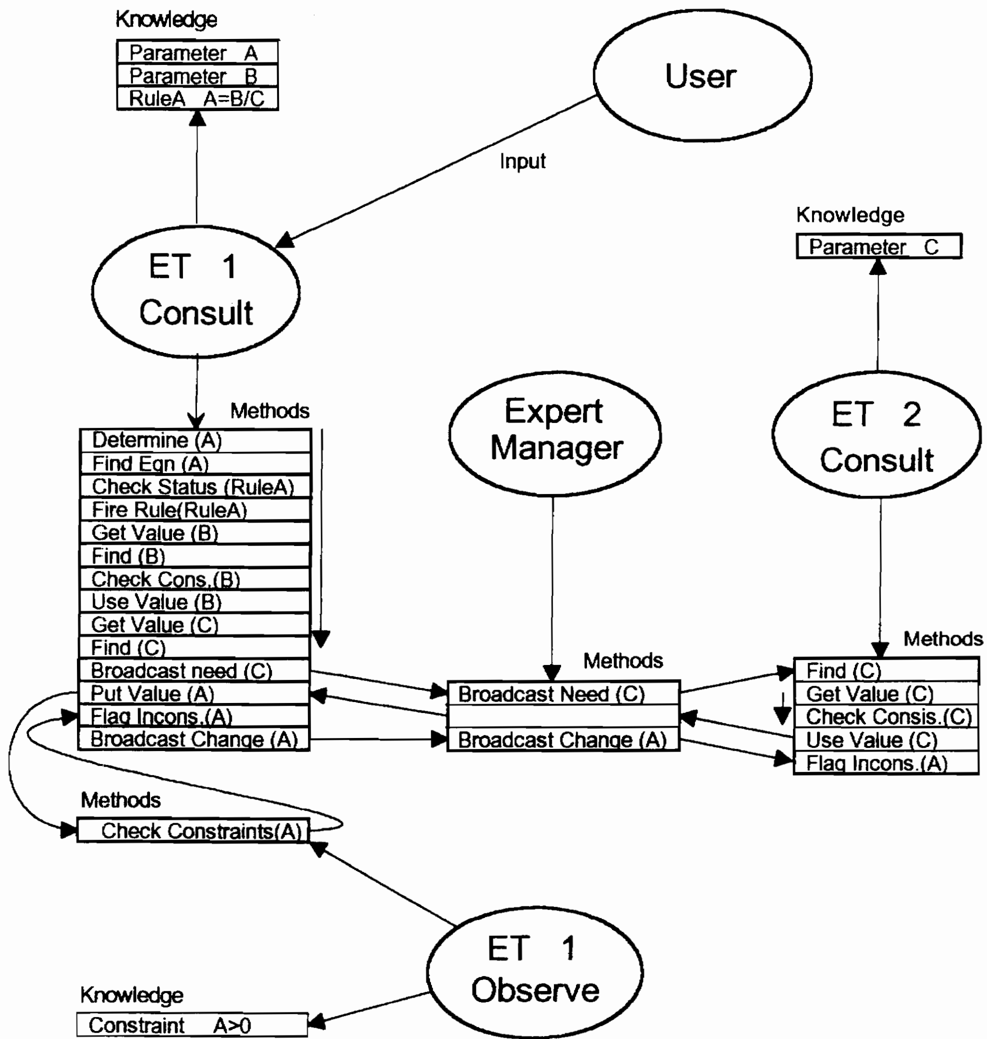


Figure 25. Example of the Inference Process

that requested it. The value of A is now calculated. Before the value is stored into the knowledge of ET 1, ET 1 in the observe mode checks the constraints on parameter A. If this checks out, then the value is changed. Once the value is changed, all other parameters that are determined by parameter A are now flagged as being inconsistent. This change in A is also broadcast out to the Expert Manager which relays the message to other ETs.

Although this process seems very complicated, neither the user nor the programmer has to be concerned with it. The user will just see the end result. The programmer just needs to be concerned about supplying the knowledge and the expert system development framework takes care of the inference process. The complexity of the inferencing process is due to the emulation of a real world situation with experts.

Due to the nature of the object-oriented design of the framework and the expert system, all the functions required for the inferencing process, described above, can be associated with one of the previously defined classes. The following is a list of the classes and the functions that can be associated with it:

Expert Manager Class:	broadcast_inconsistent_parameter
	broadcast_parameter_change
	broadcast_rule_status
	broadcast_need_value

Expert Technician Class:	locate_incosistent_parameters
	find_heuristic
	find_equation
	find_control
	find_constraint
	find_parameter
	parameter_value_changed
	rule_status_change
Parameter Class:	flag_parameter_inconsistent
	flag_parameter_consistent
	check_consistency
	use_default
Rule Class:	fire_heuristic
	fire_equation
	fire_control
	fire_constraint
	need_value
	check_status

This association of methods with the classes defined in the framework closely

resembles that in a real-world situation. Each ET that is defined in a design session, represents an expert. These experts contain knowledge, how to use the knowledge and where to go to get information outside of their domain of expertise. Therefore, in the design of the framework, there is no need for a separate inference engine class. All of the functions of the inference process are contained in specific classes.

10. Prototype Development

The development of an expert system using the framework and methodology described in the previous section is described in this chapter.

Topic Selection

To implement a prototype expert system using the expert system development framework, the selection of a field of design was required. The area of conceptual aircraft design was chosen. This area of design is one that is characterized by a high level of expertise and a competitive marketplace. There are many experts in this field, as well as numerous analytical routines that have been developed. One source of routines that was available, was a program called ACSYNT or AirCraft SYNThesis.

ACSYNT is a parametric, multi-disciplinary design tool for the conceptual design of aircraft developed by NASA and Virginia Tech [Wamp88a, Wamp88b, Jaya92]. It has been under development for over twenty years, gathering the expertise of many engineers. The code is written in FORTRAN, C and C++ and currently contains over 500,000 lines of code. It is from this pool of expertise that the equations used were

derived.

ACSYNT is broken into several disciplines in aircraft design. Examples of these are geometry, weight, trajectory, aerodynamics and propulsion. Each discipline is contained in a different module of code. These modules contain highly sophisticated functions that interact in a procedural programming technique to analyze the conceptual design. The control of these functions is performed by a control program called COPES. COPES, COntrol Program for Engineering Sythesis, was written to take any analysis code and treat is as a subroutine. These subroutines, then, can be called in any order, and any number of times by COPES to perform optimization capabilities such as sensitivity analysis. Optimizations can be performed for any objective entered by the user.

Due to the complexity of ACSYNT, a narrowed topic within ACSYNT was used for the development of the prototype. It was decided that the weights module of ACSYNT should be used. This module develops an estimated weight of the aircraft based on the current design parameters. Within this module of ACSYNT, there are four types of aircraft that can be analyzed. They are fighters, transports, bombers and general aviation. Since the methods of estimating the weight for the four types of aircraft are very different, just the general aviation portion of the weights module was used for implementing the expert system. The weights module was chosen since the weight of the aircraft ties into all other disciplines of the design. This allows for the development of a full multi-disciplinary expert system.

Control Rules and Equation Rules

The first task in developing the expert system knowledge files from the general aviation weights code was to develop the symbolic representation of the parameter relationships within the code. This was the first step in breaking away from the procedural FORTRAN code of the general aviation weights module to represent the same code non-procedurally and put it into a knowledge-based system. Since there were no subroutine calls within the code, it was thought the first method mentioned in the methodology, directed graph representation, could be used. All the parameters were drawn out on paper, without repetition. Then the arrows were drawn. The expected result was a tree like arrangement. However, the complexity of the code soon showed this is not the case. The result of this effort can be seen in Fig. 26. Therfore, to represent this parameter relationship, the labeled directed graph representation method was used. Decomposition of the system was also used. All parameters were again put down on paper. However this time, all parameters that directly affected another one were written under the affected parameter, regardless of the fact that it may appear elsewhere. Then, the arrows were drawn, showing the relationship of "this parameter affects this one." This task was not as easy as anticipated. The conversion from procedural code to a non procedural representation necessitated rewriting some of the code. These problems will be discussed in more detail in the next section.

The next step was the addition of numbers and control equations. To avoid the confusion with procedural programming, the change was made from using numbers to

Figure 26. Directed Graph Representation of Selected ACSYNT Code

using names for labeling the arrows in the graph. The names more closely identified what type of rule the arrow represented and which variable the rule governed. The result was a collection of non-connected groups of parameters, labeled to show the relationships. An example of this can be seen in Fig. 27 and the entire set can be seen in Appendix A.

The next step in the creation of the expert system was the association of the parameters and rules to experts. The experts that were used for this prototype were a weights expert, a geometry expert, a propulsion expert, a trajectory expert and a general expert. The general expert was used to accommodate knowledge that did not fit into a specific expert's domain such as how many passengers the plane was to carry and how many crew members were required.

Next, the acquired knowledge was put into the knowledge files for each of the experts. The resulting knowledge files can be seen in Appendix B. The last step was the writing of the main program and compilation of the system.

Heuristic Rules and Constraint Rules

To develop the basic necessities needed in the heuristic rule class and the constraint rule class, two meetings were held with experts in the field of aircraft design. In the first meeting, held with Dalton Sherwood and Paul Cole of Lockheed, many ideas were discussed with regard to the types of information important to designers that could be encoded as heuristic rules. Two examples that were discussed are as follows:

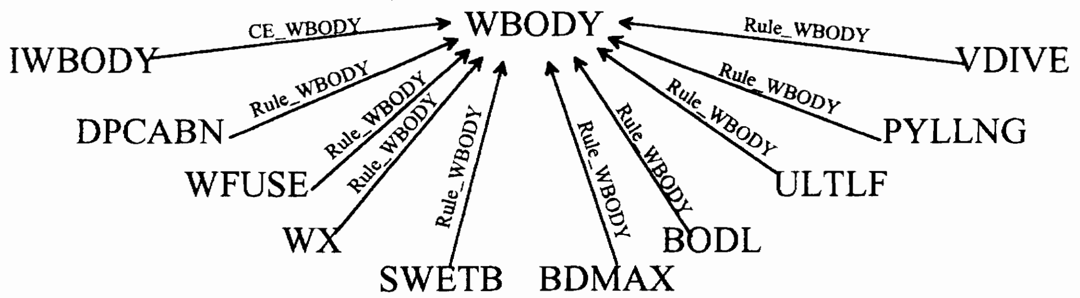


Figure 27. Decomposed Example of the ACSYNT Code

1. Commercial aircraft usually are designed to fly without an angle of attack so stewards and stewardesses do not have to push carts uphill and downhill. However, there was an exception in the design of the L-1011. The reason given for this design was that the angle of attack gave the aircraft a longer range.
2. An aircraft designer may want to include allowance in the initial design for flexibility for after construction adjustments and modifications. This was seen in the design of the Douglass DC-10. A slightly larger wing was used to allow for the creation of derivative aircraft.

The second meeting, held with Paul Gelhausen of NASA, concentrated more on constraints on specific parameter values due to code limitations of ACSYNT as well as those that apply to general aviation aircraft. Constraints that were discussed during the meeting are listed below:

1. The number of passengers should not exceed 20 for the general aviation code. Higher capacity aircraft should use transport equations due to the difference in subsystems.
2. The ultimate load factor should not exceed 6 or 7 for the general aviation equations used in ACSYNT.
3. Aspect ratio of the wing and horizontal tail usually should not exceed 10.
4. Aspect ratio of the vertical tail should be less than 5.
5. Aspect ratio of the canard should be less than either 6 or the

aspect ratio of the wing, whichever is smaller.

6. The ratio of body length to body width should be less than 8.
7. Thickness to chord ratio should not be less than 1 or 2 percent.
8. Larger thickness to chord ratios, above 15 percent, should be handled by different equations than those used in the ACSYNT code.
9. Weight to wing area ratio should be between 10 to 50 lbs./sq.ft.

These examples from both meetings typify what type of rules the constraint and heuristic rule class will encompass.

There were also many suggestions made by Sherwood, Cole and Gelhausen for future modifications to the expert system development framework. One idea that was brought up in both meetings was the addition of component libraries that could be used to select parts for the aircraft design. Many components, such as tires and landing gear, go through a stringent qualification process. If a qualified component can be used, it saves a lot of time and money in the development of the aircraft. In the meeting with Dalton and Cole, the idea of adding a reason for a constraint on a parameter was presented. This would allow a designer to see if a violated constraint was an arbitrarily set constraint that can be adjusted or a constraint that must be followed. An example of this was in the design of the F-22. There was one constraint that the Air Force had required that would not allow the design to converge on a solution. After discussion with the Air Force, it was found out that this constraint was arbitrarily set. By changing the value of this constraint, the design converged on a solution. Another idea proposed by

Gelhausen, was the ability to maximize, minimize or set the value of the ratio of two arbitrary components such as wing area to gross take-off weight.

Problems Encountered

During the development of the symbolic representation, many problems were encountered in the conversion of procedural FORTRAN code to a non-procedural representation. The first problem that was encountered was due to difference in FORTRAN's if-then-elseif statement and the knowledge base representation which lacks the elseif statement. The knowledge base can only contain if-then-else statements. This meant that all of the if-then-elseif statements that contained multiple elseif statements, such as that shown in Fig. 28, had to be changed. They were rewritten to become multiple if-then-else statements where if the first statement is not true the action taken by the else statement is to fire the next if-then-else and so on. The end result of the chaining of rules is the same as a multiple elseif statement. An example of the rewritten knowledge file form can be seen in Fig. 29.

Another problem that arose quite frequently in the FORTRAN code was the way the code was written knowing it would be procedurally performed. Take the example shown in Fig. 30. This procedural code must be changed to accommodate a non-procedural program. The rewritten code, also shown in Fig. 30, is the same number of equations but now can be used in the knowledge base.

The most difficult change in translating the code had to do with iteration and

```
IF(JFLTYP .EQ. 1) THEN
   WFLAP = WCFLAP*(VFLAP/100.)**2*SFLAP/SQRT(FLAPN)
ELSEIF(JFLTYP .EQ. 2) THEN
   IF(VFLAP .LE. 160.) THEN
      WFLAP=WCFLAP*SFLAP*0.369*VFLAP**0.2733
   ELSE
     WFLAP=WCFLAP*SFLAP*(VFLAP**2.195)/45180.
   ENDIF
ELSEIF(JFLTYP .GE. 3 .AND. JFLTYP .LE. 5) THEN
  WFLAP = WCFLAP*(VFLAP/100.)**2*SFLAP*FLAPN**.5
ELSEIF(JFLTYP .GE. 6) THEN
  WFLAP=WCFLAP*(VFLAP/100.)**2.38*SFLAP**1.19/(FLAPN**.595)
ENDIF
```

Figure 28. If-Then-Else Statement from ACSYNT Code

```
Control_Eqn   CE_WFLAP_1   JFLTYP == 1
THEN   1
Activate_Rule   Rule_WFLAP_1
ELSE   1
Fire_Rule   CE_WFLAP_2

Control_Eqn   CE_WFLAP_2   JFLTYP == 2
THEN   1
Fire_Rule   CE_WFLAP_2_1
ELSE   1
Fire_Rule   CE_WFLAP_3

Control_Eqn   CE_WFLAP_2_1   VFLAP <= 160
THEN   1
Activate_Rule   Rule_WFLAP_2
ELSE   1
Fire_Rule   Rule_WFLAP_3

Control_Eqn   CE_WFLAP_3   JFLTYP >= 3 && JFLTYP <= 5
THEN   1
Activate_Rule   Rule_WFLAP_4
ELSE   1
Fire_Rule   CE_WFLAP_4

Control_Eqn   CE_WFLAP_4   JFLTYP >= 6
THEN   1
Activate_Rule   Rule_WFLAP_5
ELSE   0
```

Figure 29. If-Then-Else Statement Converted to Knowledge File Format

```
IF(IMNTEN .EQ. 0) FPROP = 1.05
IF((EN .LE. 2.) .AND. (IMNTEN .EQ. 1)) FPROP = 0.95
IF((EN .GT. 2.) .AND. (IMNTEN .EQ. 1)) FPROP = 0.90
```

```
Control_Eqn    CE_FPROP_1    IMNTEN == 0
THEN    1
Activate_Rule    Rule_FPROP_1
ELSE    1
Fire_Rule    CE_FPROP_2

Control_Eqn    CE_FPROP_2    IMNTEN == 1 && EN <= 2
THEN    1
Activate_Rule    Rule_FPROP_2
ELSE    1
Fire_Rule    CE_FPROP_3

Control_Eqn    CE_FPROP_3    IMNTEN == 1 && EN > 2
THEN    1
Activate_Rule    Rule_FPROP_3
ELSE    0
```

Figure 30. Procedural Code Example and Converted Knowledge File Format

convergence routines. Very often in design, the value of a parameter is the result of a convergence routine based on an initial guess. This is an easy task to program in procedural code; set the initial guess and then enter into the convergence routine. However, non-procedurally, this is a little more difficult. The iteration loop as well as the convergence criteria check must be located in the same equation. Also, the initial guess must be imbedded into the code as an additional control equation that determines if the iteration is in the first loop of iteration. If it is, the control equation will activate the equation that used an initial guess. If it has already gone through the first iteration, then the control equation will activate the equation that uses the previously calculated value. A sample of this can be seen in Fig. 31.

```
Control_Eqn   CE_WWING_2   ( ABS ( WWING - WWOLD ) / WWING ) && < 20
THEN   6
Activate_Rule   CE_FWL
Activate_Rule   CE_FRPOP_1
Activate_Rule   CE_FGEAR_1
Activate_Rule   Rule_WWOLD
Fire_Rule   Rule_ITER
Fire_Rule   CE_WWING_2
ELSE   0
```

Figure 31. Example of an Iteration Process in Knowledge File Format

11. Implementation and Results

A portion of the overall framework for the system has been partially implemented in C++ on the IBM RISC/6000 platform by Narayanan [Nara93] utilizing OSF/Motif in the X Windows environment. Several of the classes and methods described in this thesis have been added to this implementation as well as the modifications and additions described. Those classes that have been implemented are the Rule class, the Control Rule class, the Equation class and the Parameter class. The methodology described for developing an expert system using the framework was performed on the general aviation section of the weights module of ACSYNT, producing the directed graph representation of the procedural code. This was then converted into the knowledge files or persistent ET files. The Heuristic Rule class and the Constraint Rule class are still in the development stages. The inference process described also has not been implemented.

The creation of knowledge from the equations in ACSYNT resulted in 1100 lines of parameters and rules. A total of 215 instances of the Parameter class were created, 160 instances of the Equation Rule class and 79 instances of the Control Rule class. The 160 instances of the Equation Rule class were developed from the 160 directed graphs created

in the decomposition of the procedural code.

The main program that was written to create the expert system can be seen in Fig. 32. It contains the five expert technicians used in the implementation and the modes of operation.

The user interface of the expert system created by the compilation of the knowledge files, main program and framework can be seen in Fig. 33. This figure shows the main menu of the system. It includes each ET created and the mode of operation. By selecting one of the ETs, the secondary menus appear. Fig. 34, shows the user's selection of the "Effect" option from the secondary menu. The user has selected the option "Maximize" and is presented a list of parameters that can be selected. If the user had selected "Cause" instead of "Effect" from the menu, a screen as shown in Fig. 35 would be presented. The user also has the option of seeing the current value of a parameter. Two examples of this are seen in Fig. 36. The top example shows a short list of parameters that can be displayed in one column. However, if the list of parameters is long, such as that in the bottom example, the parameters are automatically put into columns. Only the first column will be displayed initially. By pressing the "More" button, another column will appear and so on, until there are no more columns. If the user would like to change the value of a parameter, the "New Value" option from the "effect" menu generates a pop-up box for the entry of the value. This is seen in Fig. 37. All parameter values, however, are not changeable. This can be seen in Fig. 38. The top example shows the entire list of parameters that belong to the Trajectory expert. The bottom example shows which parameters can be changed by the user. An example of an

Main.C

```
#include "Session_Manager.h"
#include "ET.h"
#include <stdio.h>
main()
{
Session_Manager *manage_all = new Session_Manager();
ET weight("Weight");
ET geometry("Geometry");
ET general("General");
ET trajectory("Trajectory");
ET propulsion("Propulsion");
manage_all->handle_events();
}
```

Figure 32. C++ Program for Creating the Prototype

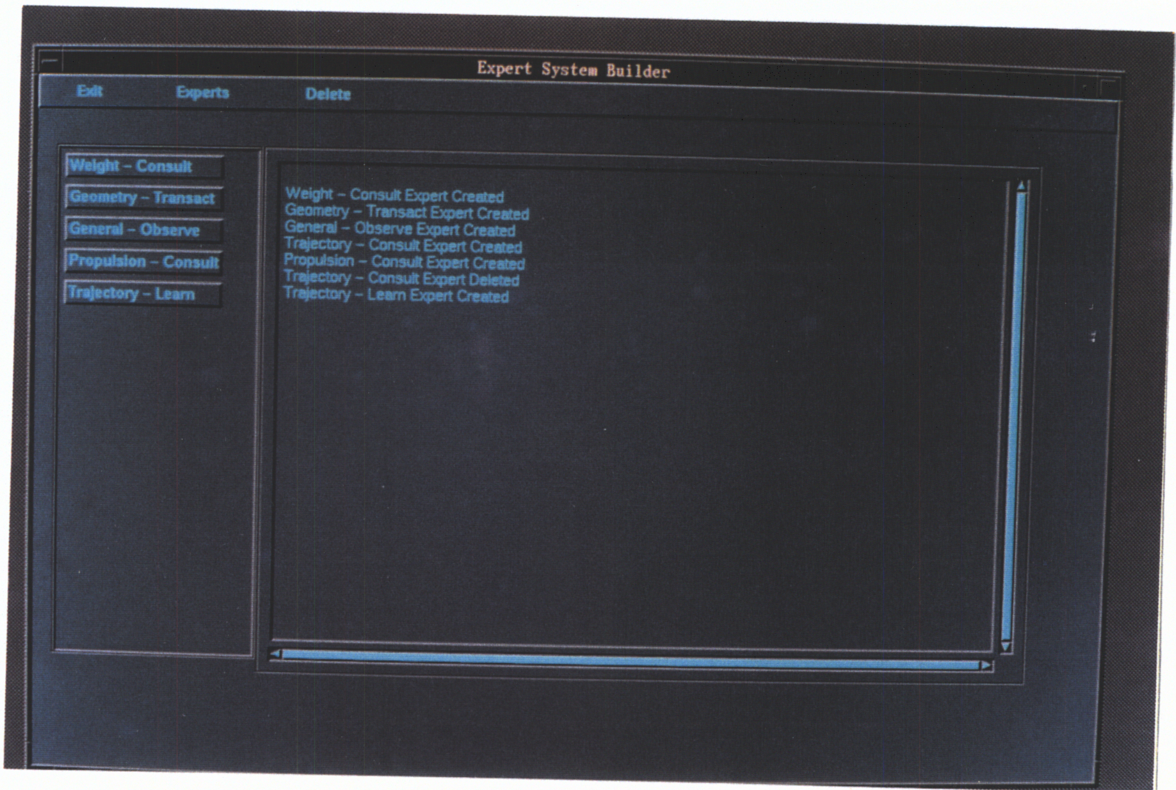


Figure 33. Main Screen of the Prototype Expert System

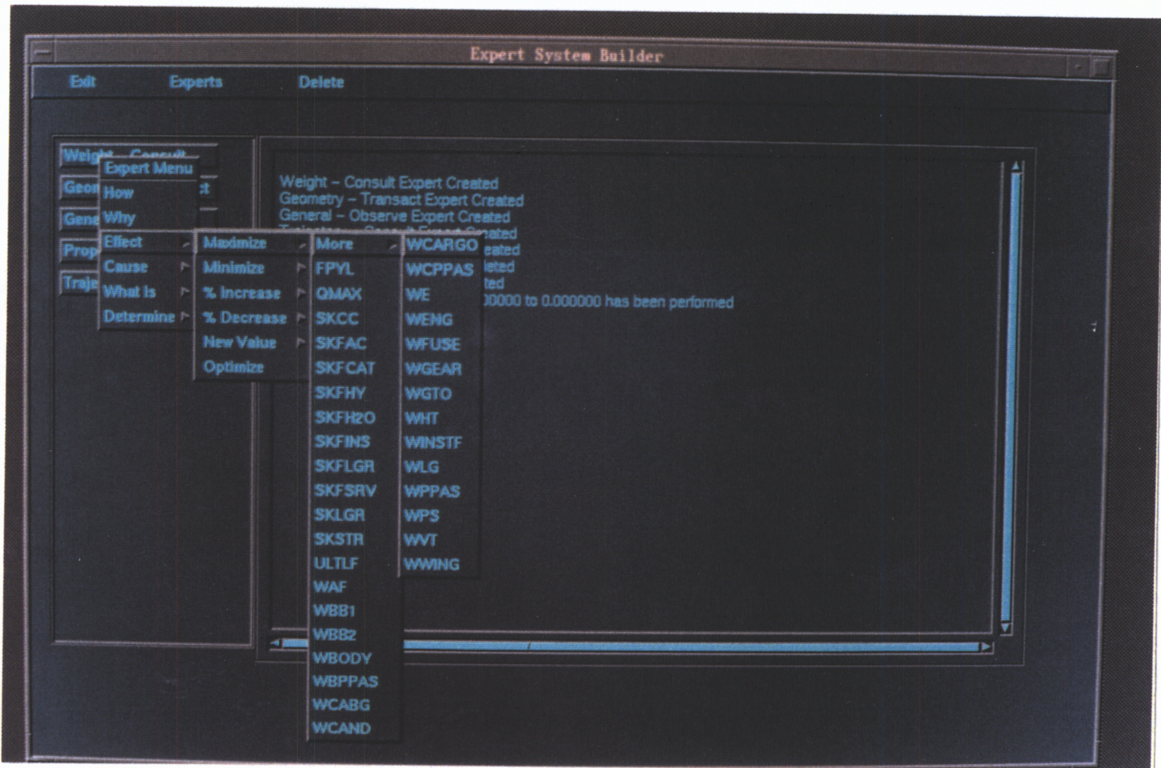


Figure 34. "Effect" Menu

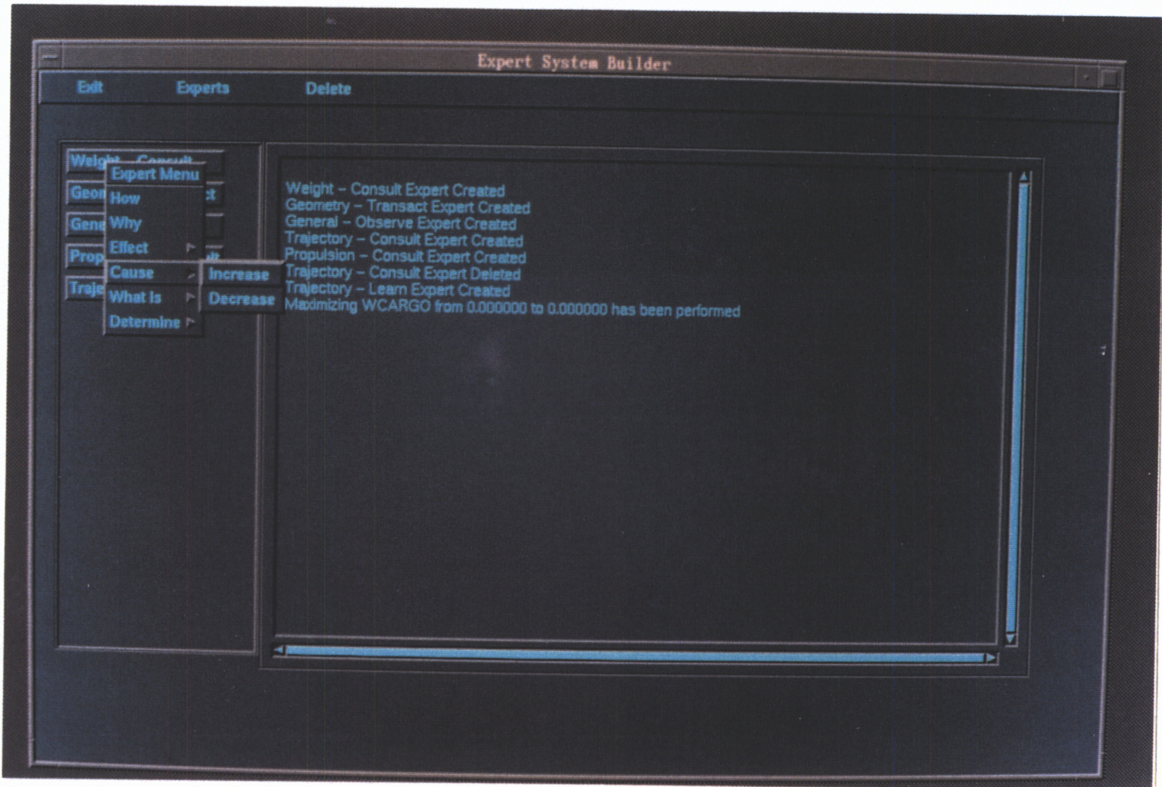
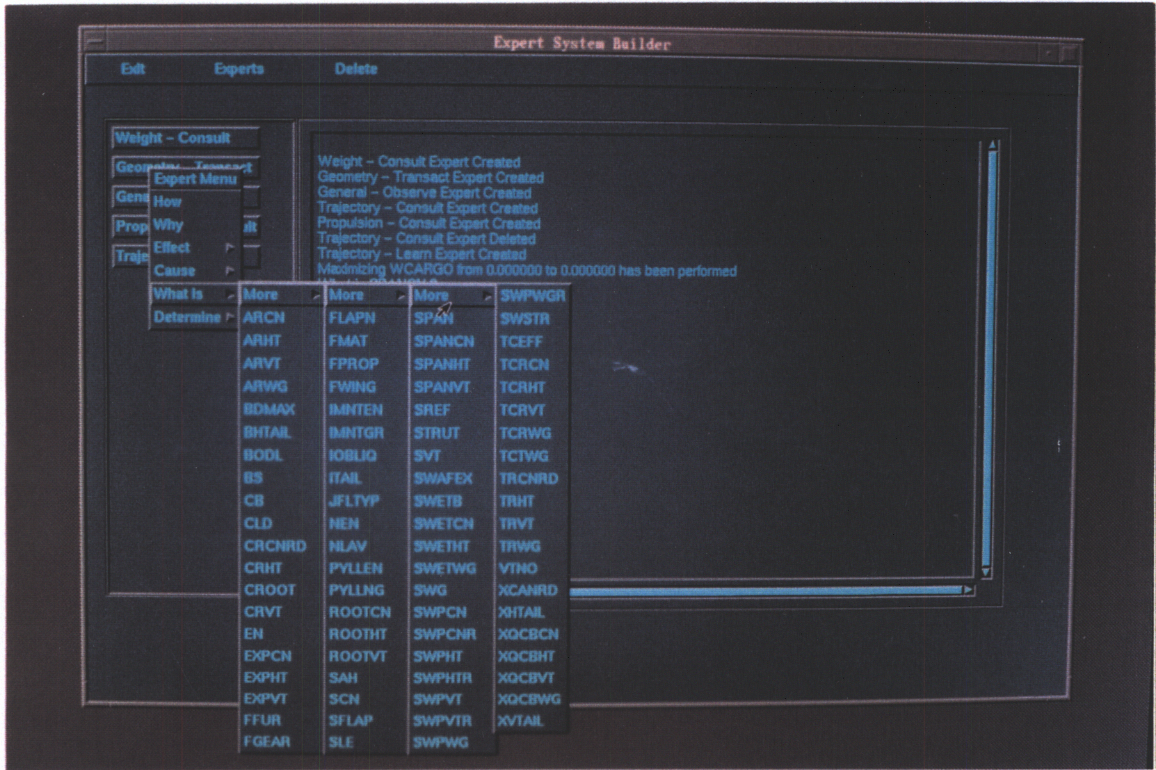


Figure 35. "Cause" Menu



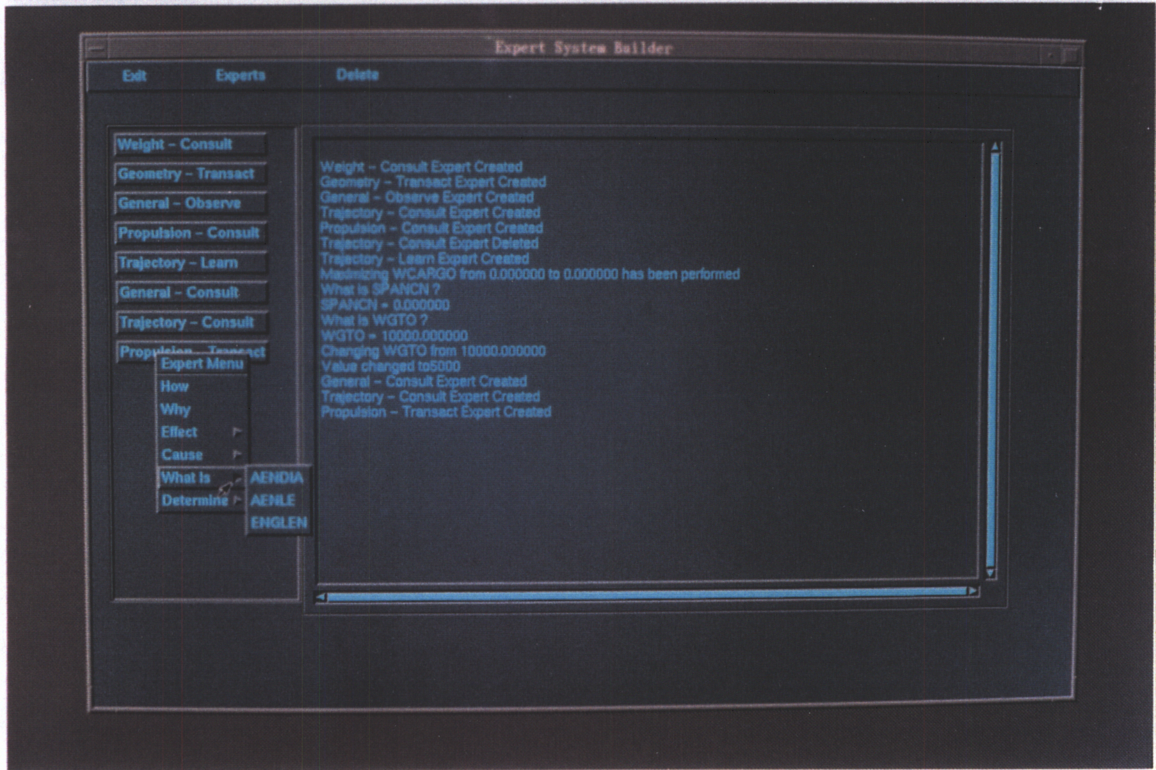


Figure 36. Determining the Current Value of a Parameter

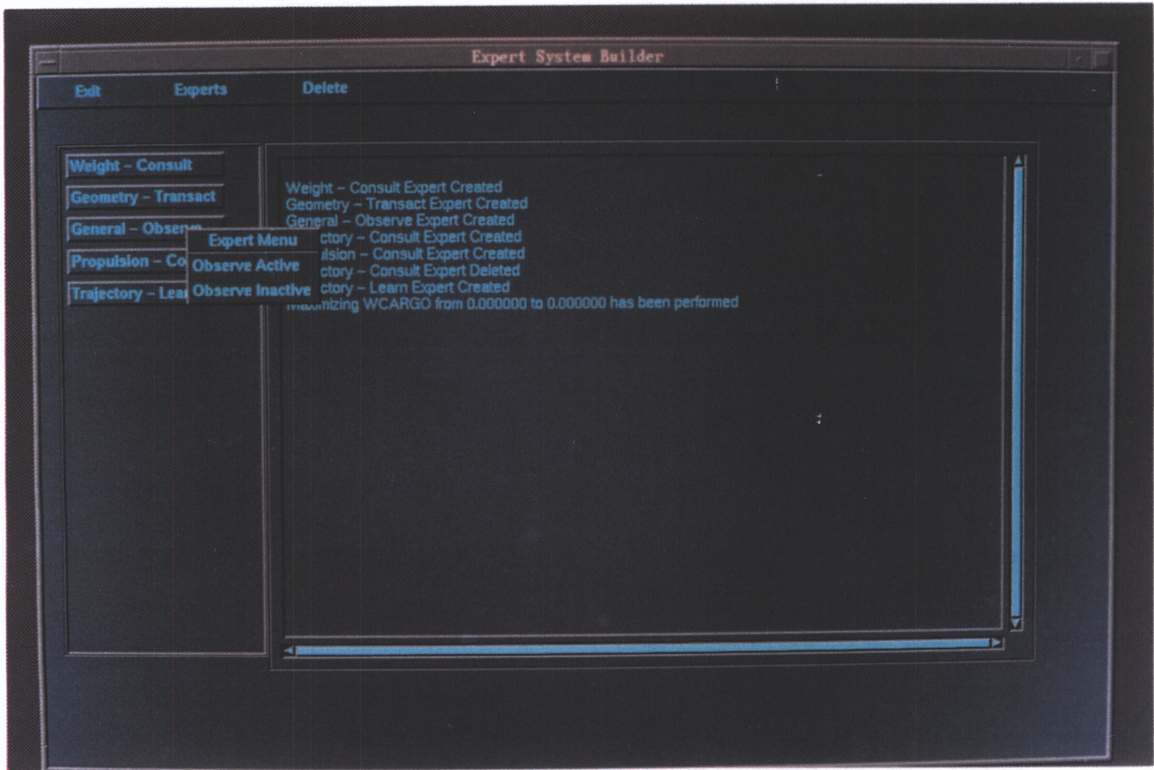


Figure 37. Observe Mode Menu

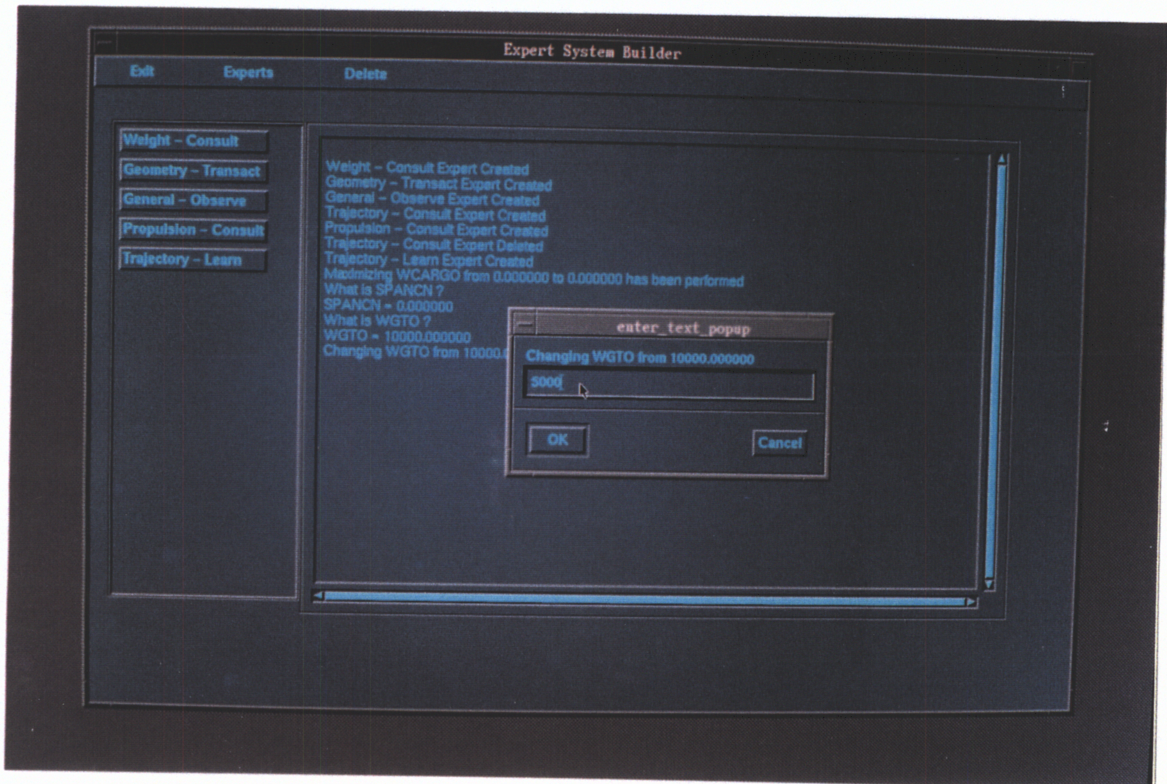
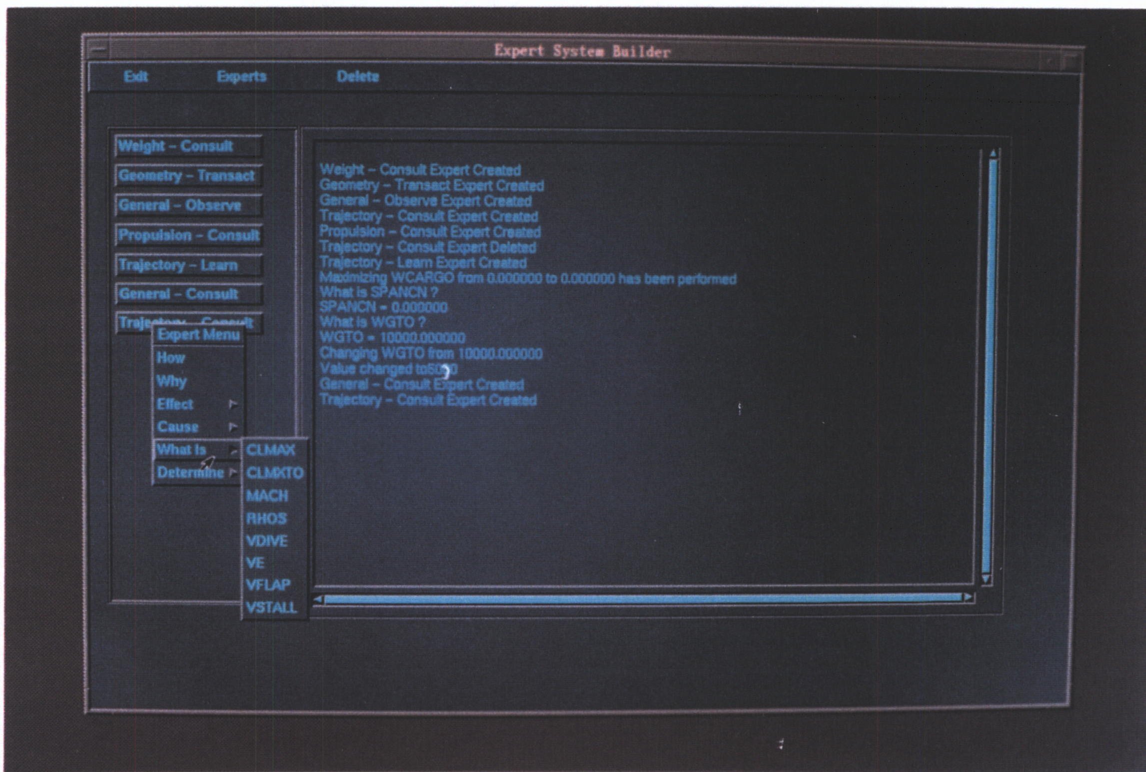


Figure 38. **Entering in a New Value for a Parameter**

expert in the Observe mode is shown in Fig. 39. The menu allows the expert to be turned off without deleting the expert. If the user decides that an expert is no longer necessary, or needs to add another expert, this can be done through the main menu interface. This can be seen in Fig. 40. The figure also shows that only those experts that exist can be deleted and an expert that already exists can not be created.



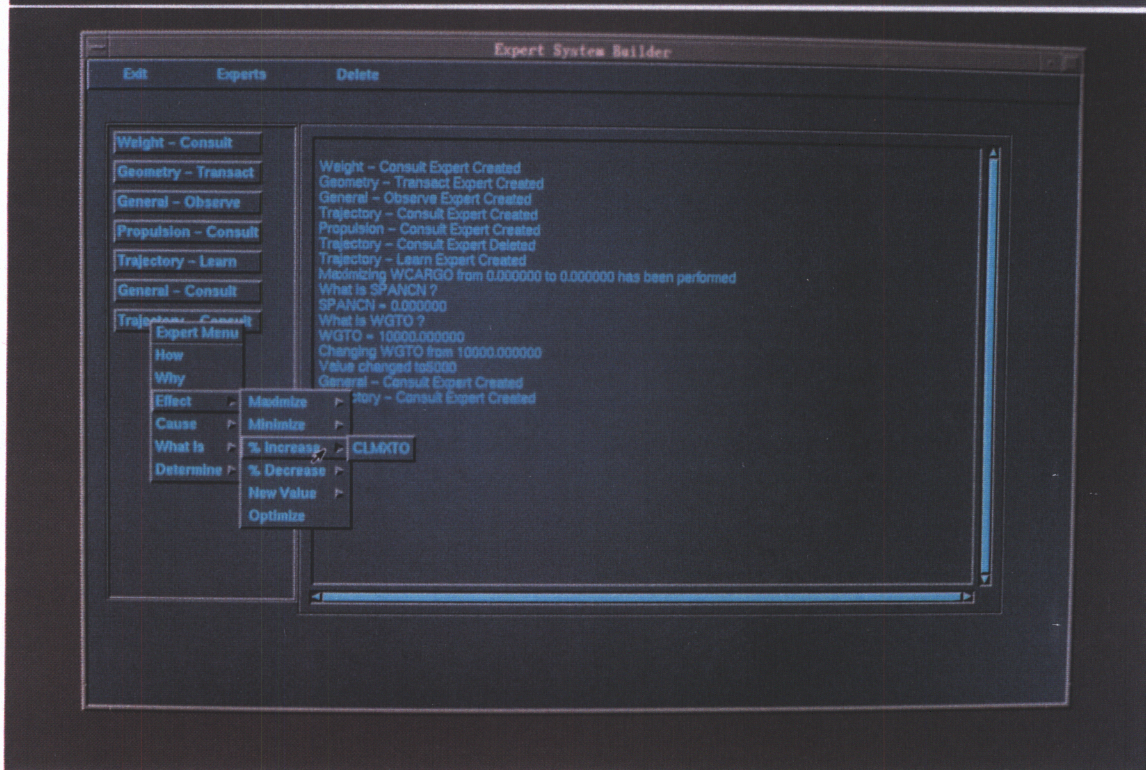
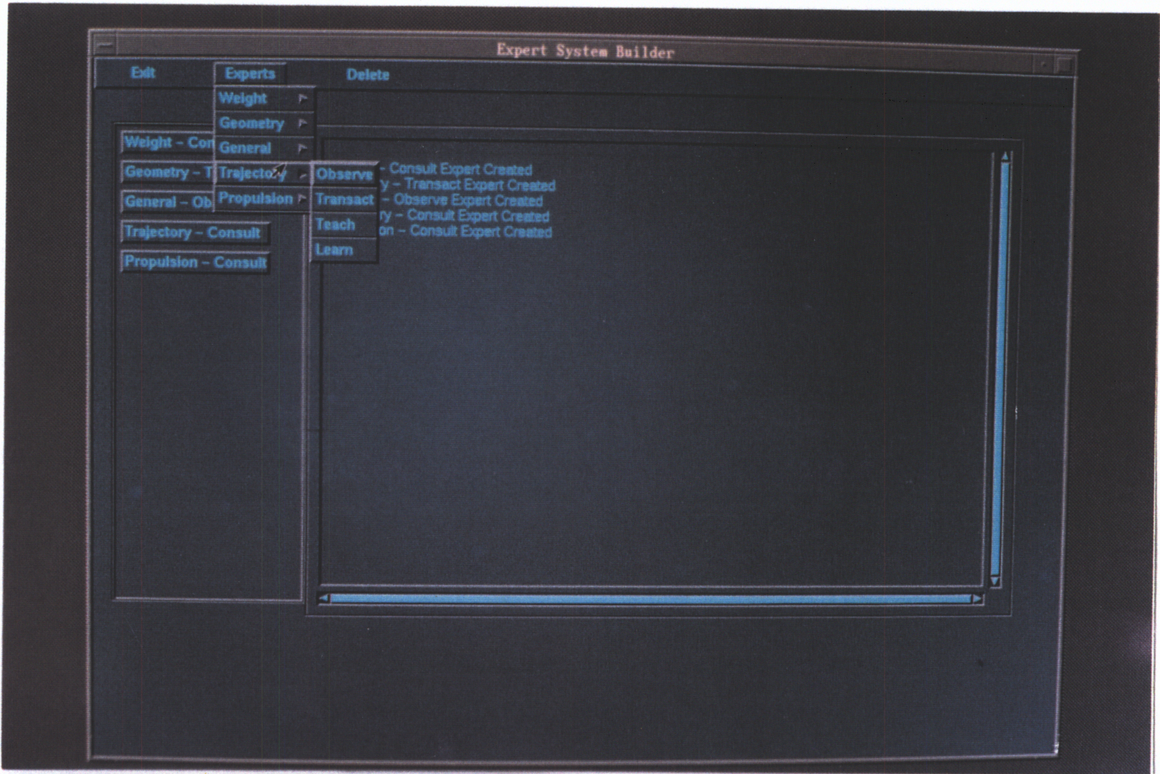


Figure 39. **Changeable vs. Not Changeable Parameters**



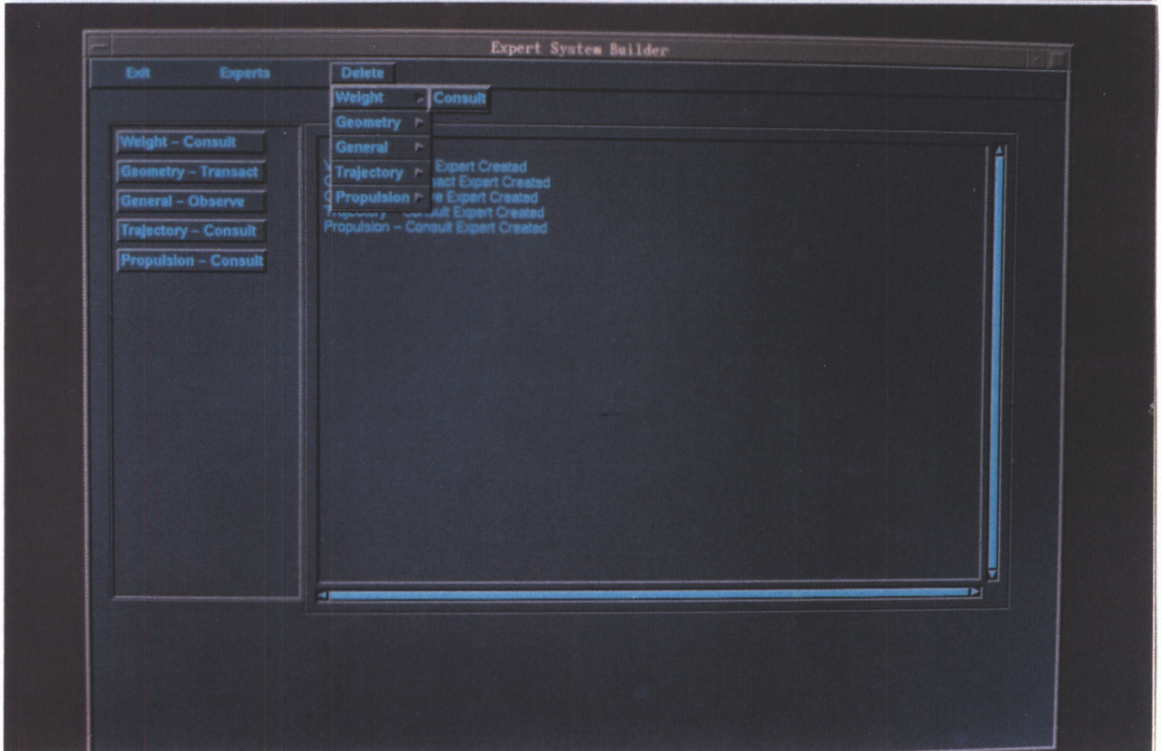


Figure 40. Creating and Deleting Experts

12. Summary and Conclusions

The rule classes to be used in the expert consultation environment framework were designed fully and partially implemented in this thesis. The methodology used in the design of an expert system using the framework was defined and demonstrated in the field of parametric, multi-disciplinary aircraft design.

The majority of the time spent in meeting the requirements set forth for this thesis, was spent in the parallel design of the rule classes and the methodology. As in any design process, many iterations in the design occurred. As the requirements for the rule classes changed, the methodology for using the rules also changed. Once the required functions of the rule classes were set forth, the task of converting the procedural code to graph theory took place. As problems occurred with the conversion due to rule capabilities, more requirements for the rule classes developed. This, in turn, changed the methodology.

As the inference process was designed, tailored to the needs of CAD systems, it was evident there was no need for a separate inference engine class. All the methods of the inference process became associated with a previously defined class. This result gives a programmer the ability to add new types of rules to the framework without

having to worry about modifying an inference engine. Methods for the inference process are added to the new rule type to communicate with the other classes.

The actual time spent in knowledge acquisition and representation for use by the framework was approximately one month. This is minimal compared to developing an entire expert system from scratch. The scope of the prototype was limited compared to a full fledged expert system that would be developed, so there would be an increase in time. However, the freedom from programming the user interface and inference process will allow a programmer to develop a customized expert system in a much shorter span of time. A programmer would be required to learn the different classes of rules that are used in the framework and their role in knowledge representation. Also, the knowledge file format would also have to be learned. The C++ syntax for the main program would have to followed for creating the system. Although the engineer or programmer may not know C++, learning to write a ten or twenty line program is much easier than writing a 100,000 line program. Since the burden of the programming difficulties are removed, the programmer can concentrate on the acquisition of knowledge; the heart of the expert system.

There are several areas of the Expert Consultation Environment that need to be worked on before it can be utilized. First, the inference process must be implemented. This would include developing the functions oultined in this thesis as well as expanding on those ideas presented. The Heuristic Rule class and Constraint Rule class must also be implemented. The existing ECE only operates in the consult mode. The development of the other modes of operation are a large and important undertaking that must be

completed to meet the objectives set out by Jayaram, et al [Jaya89, Jaya90, Jaya93] for the ECE.

13. References

[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 to Aeronautical System Design", AIAA-88-4426, presented at the *AIAA/AHS/ASEE Aircraft Systems and Operations Meeting,* Septemeber 7-9, 1988, Atlanta GA.

[Bouc92] Bouchard, E. E., "Concepts for a Future Aircraft Design Environment", AIAA-92-1188, presented at the *1992 Aerospace Design Conference*, February 3-6, 1992 Irvine, CA.

[Dvor91] Dvorak, P., "Keeping Talent with Knowledge Systems", Machine Design, August 22, 1991, pp 37-42.

[Elia86] Elias, A. L., "Knowledge Engineering of the Aircraft Design Process", *Knowledge Based Problem Solving*, Prentice-Hall, NJ, 1986.

[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.

[Jaya92] Jayaram, S., Myklebust, A., and Gelhausen, P., "ACSYNT - A Standards-Based System for Parametric Computer Aided Conceptual Design of Aircraft", presented at the *1992 Aerospace Design Conference*, February 3-6, 1992, Irvine, CA.

[Jaya92a] Jayaram, U., Jayaram, S., Myklebust, A., "Parameter Component

Modeling for Computer Aided Design", presented at the *2nd International Conference on Computational Graphics and Visualization Techniques, COMPUGRAPHICS 92*, Lisbon, Portugal, December, 1992.

[Jaya93] Jayaram, S., and Myklebust, A., "Device Independent Programming Environments for CAD/CAM Software Creation", *Computer Aided Design* Volume 25, No 2, February 1993, pp 94-105.

[Jenk84] Jenkinson, L. R., and Simos, D., "A Computer Program for Assisting in the Preliminary Design of Twin-Engined Propeller-Driven General Aviation Aircraft", *Canadian Aeronautics and Space Journal*, Volume 30, Number 3, September 1984, pp 213-224.

[Kolb88] Kolb, M. A., "A Flexible Computer Aid for Conceptual Design Based on Constraint Propogation 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/AHA/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.

[Mors91] Morse, D. V. and Hendrickson, C., "Model for Communication in Automated Interactive Engineering Design", *Journal of Computing in Civil Engineering*, Volume 5, Number 1, January, 1991, pp 4-24.

[Nara93] Narayanan, P., "An Object-Oriented Framework for the Creation of Customized Expert Systems for CAD", *M.S. Thesis*, Mechanical Engineering Department, Virginia Polytechnic Institute and State University, May 7, 1993.

[Prer85] Prerau, D. S., "Selection of an Appropriate Domain for an Expert System", The AI Magazine, Summer 1985, pp 26-30.

[Rein90a] Reinschmidt, K. F., "Engineering in the Future", presented at Technology Today and Tomorrow, The CATIA Operator's Exchange, October 30, 1990, Costa Mesa, CA.

[Rein90b] Reinschmidt, K. F., and Finn, G. A., "Integration of Expert Systems, Databases, and Computer-Aided Design", To appear in *Intelligent Design and Manufacturing.*

[Rose86] Rosenman, A., Gero, J. S., Hutchinson. P. J., and Oxman, A., "Expert Systems Applications in Computer-Aided Design", *Computer Aided Design,* 1986, pp 546-551.

[Sobi91] Sobieszczanski-Sobieski, J., and Tulinius, J., "MDO can help resolve the designer's dilemma", *Aerospace America*, September 1991, pp 32-35,63.

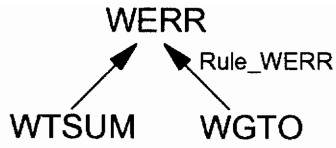
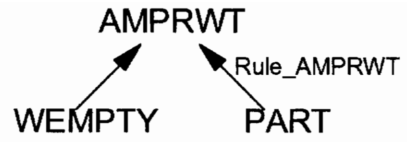
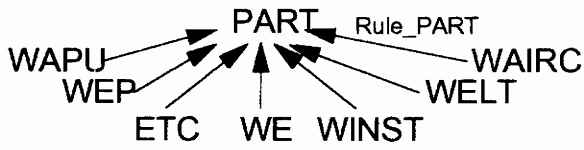
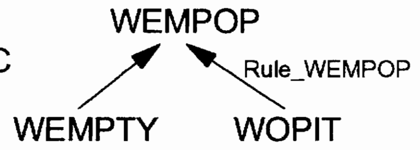
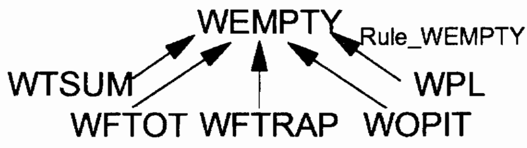
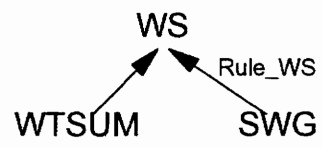
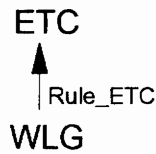
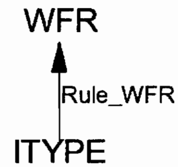
[Tong92] Tong, S. S., Powell, D., and Goel, S., "Integration of Artificial Intelligence and Numerical Optimization Techniques for the Design of Complex Aerospace Systems", AIAA-92-1189, presented at the *1992 Aerospace Design Conference*, February 3-6, 1992, Irvine, CA.

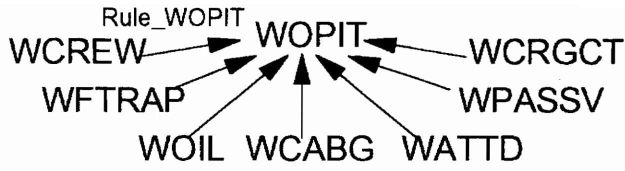
[Ullm87] Ullman, D. G., and Dietterich, T. A., "Mechanical Design Methodology: Implications on Future Developments of Computer-Aided Design and Knowledge-Based Systems", *Engineering with Computers,* Volume 2, 1987, pp 21-29.

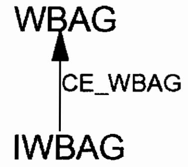
[Wamp88a] Wampler, S. G. "Development of a CAD System for Automated Conceptual Design", *M.S. Thesis*, Mechanical Engineering Department, Virginia Polytechnic Institue and State University, Blacksburg, VA, 1988.

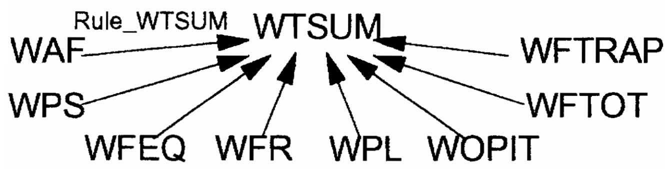
[Wamp88b] Wampler, S. G., Myklebust, A., Jayaram, S. and Gelhausen, P., "Improving Aircraft Conceptual Design - A PHIGS Interactive Graphics Interface for ACSYNT", AA-88-4481, presented at *AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference*, Atlanta, GA. September 7-9, 1988.

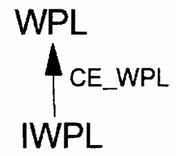
Appendix A - Graph Representation

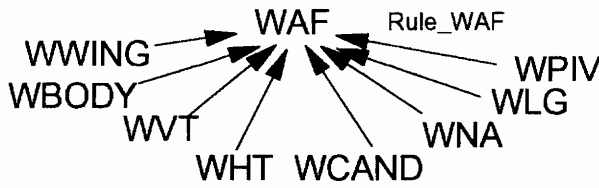

WERR
Rule_WERR
WTSUM
WGTO

AMPRWT
Rule_AMPRWT
WEMPTY
PART

PART
Rule_PART
WAPU
WEP
ETC
WE
WINST
WELT
WAIRC

WEMPOP
Rule_WEMPOP
WEMPTY
WOPIT

WEMPTY
Rule_WEMPTY
WTSUM
WFTOT
WFTRAP
WOPIT
WPL

WS
Rule_WS
WTSUM
SWG

ETC
Rule_ETC
WLG

WFR
Rule_WFR
ITYPE

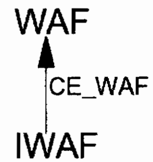

Rule_WOPIT
WOPIT
WCREW
WCRGCT
WFTRAP
WPASSV
WOIL
WCABG
WATTD

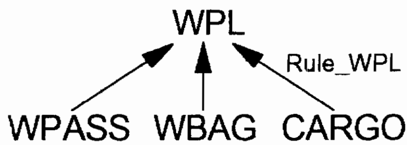

WBAG
CE_WBAG
IWBAG

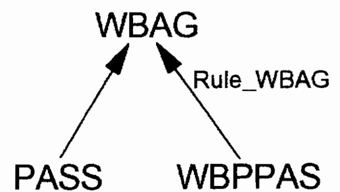

Rule_WTSUM
WTSUM
WAF
WFTRAP
WPS
WFTOT
WFEQ
WFR
WPL
WOPIT

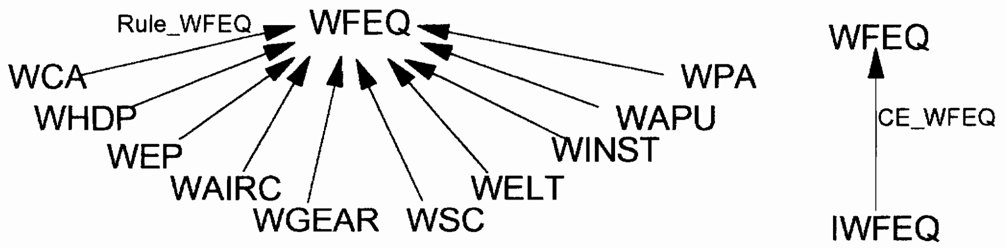

WPL
CE_WPL
IWPL

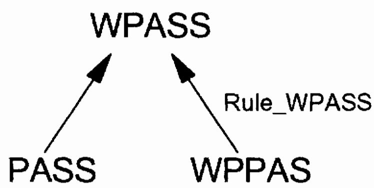

WAF
Rule_WAF
WWING
WPIV
WBODY
WLG
WVT
WNA
WHT
WCAND

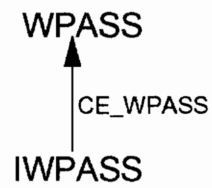

WAF
CE_WAF
IWAF

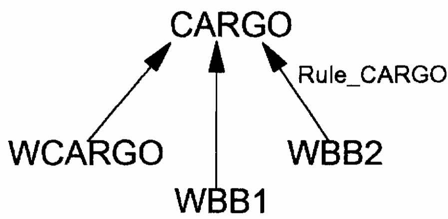

WPL
Rule_WPL
WPASS
WBAG
CARGO

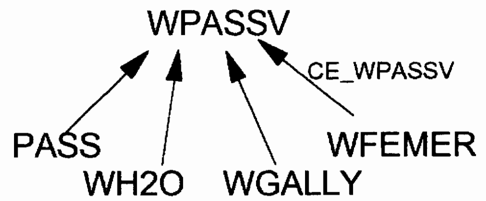

WBAG
Rule_WBAG
PASS
WBPPAS

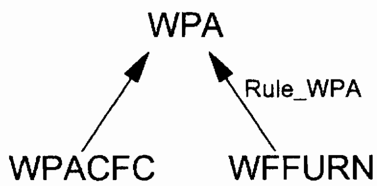

Rule_WFEQ
WFEQ
WCA
WHDP
WEP
WAIRC
WGEAR
WSC
WELT
WINST
WAPU
WPA
WFEQ
CE_WFEQ
IWFEQ

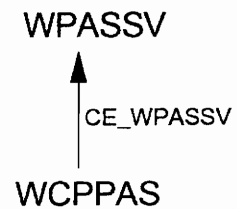

WPASS
Rule_WPASS
PASS
WPPAS

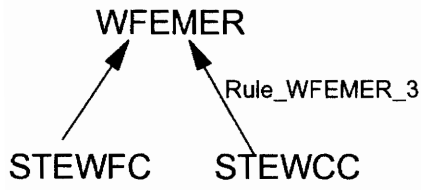

WPASS
CE_WPASS
IWPASS

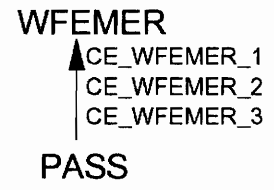

CARGO
Rule_CARGO
WCARGO
WBB1
WBB2

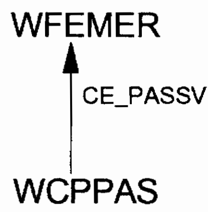

WPASSV
CE_WPASSV
PASS
WH2O
WGALLY
WFEMER

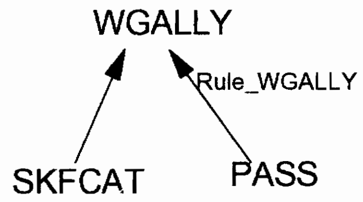

WPA
Rule_WPA
WPACFC
WFFURN

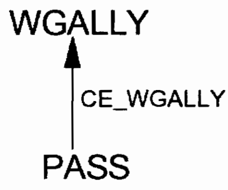

WPASSV
CE_WPASSV
WCPPAS

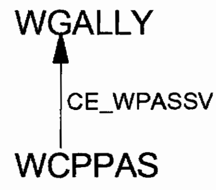

WFEMER
Rule_WFEMER_3
STEWFC
STEWCC

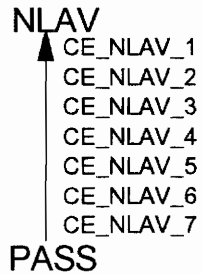

WFEMER
CE_WFEMER_1
CE_WFEMER_2
CE_WFEMER_3
PASS

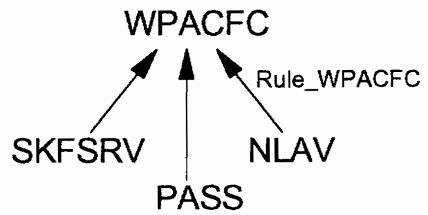

WFEMER
CE_PASSV
WCPPAS

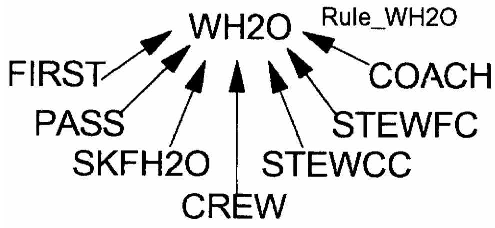

WGALLY
Rule_WGALLY
SKFCAT
PASS

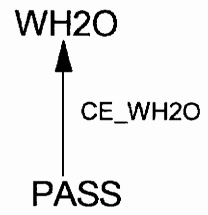

WGALLY
CE_WGALLY
PASS

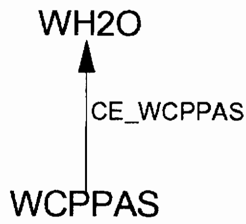

WGALLY
CE_WPASSV
WCPPAS

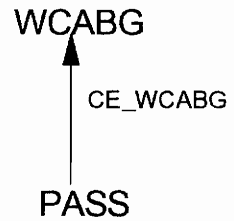

NLAV
CE_NLAV_1
CE_NLAV_2
CE_NLAV_3
CE_NLAV_4
CE_NLAV_5
CE_NLAV_6
CE_NLAV_7
PASS

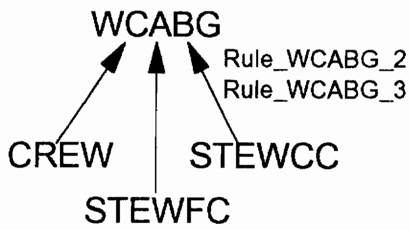

WPACFC
Rule_WPACFC
SKFSRV
NLAV
PASS

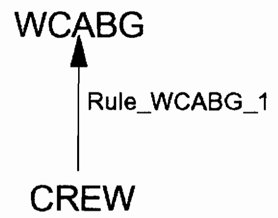

WH2O
Rule_WH2O
FIRST
COACH
PASS
STEWFC
SKFH2O
STEWCC
CREW

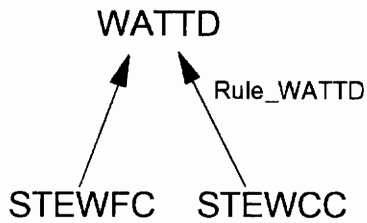

WH2O
CE_WH2O
PASS

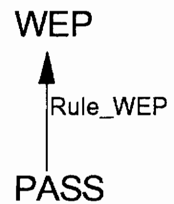

WH2O
CE_WCPPAS
WCPPAS

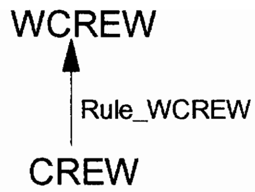

WCABG
CE_WCABG
PASS

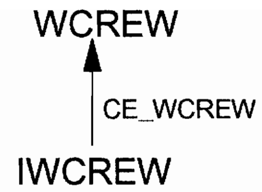

WCABG
Rule_WCABG_2
Rule_WCABG_3
CREW
STEWCC
STEWFC

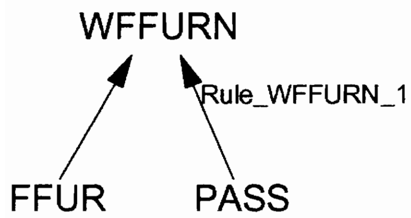

WCABG
Rule_WCABG_1
CREW

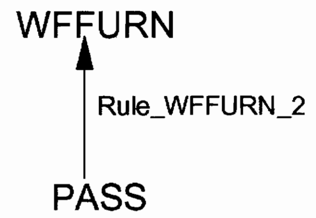

WATTD
Rule_WATTD
STEWFC
STEWCC

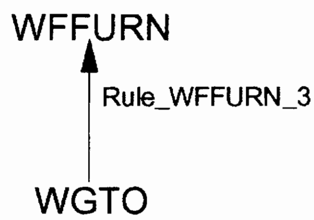

WEP
Rule_WEP
PASS

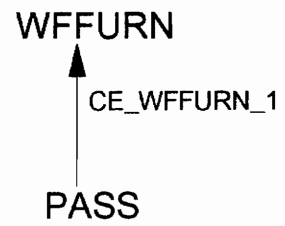

WCREW
Rule_WCREW
CREW

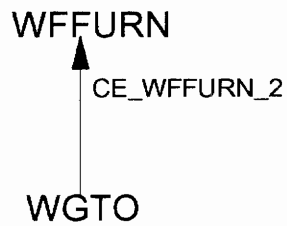

WCREW
CE_WCREW
IWCREW

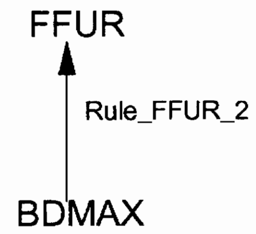

WFFURN
Rule_WFFURN_1
FFUR
PASS

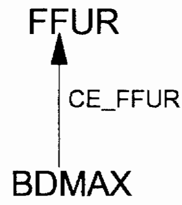

WFFURN
Rule_WFFURN_2
PASS

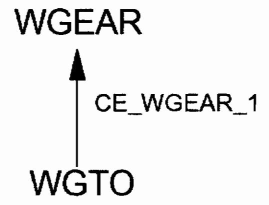

WFFURN
Rule_WFFURN_3
WGTO

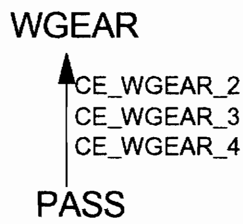

WFFURN
CE_WFFURN_1
PASS

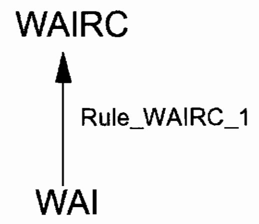

WFFURN
CE_WFFURN_2
WGTO

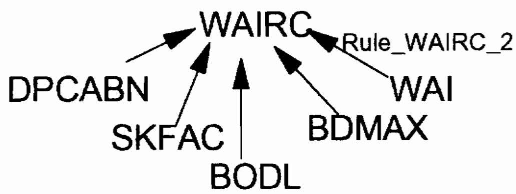

FFUR
Rule_FFUR_2
BDMAX

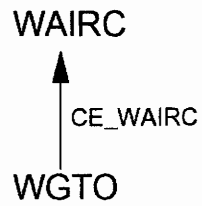

FFUR
CE_FFUR
BDMAX

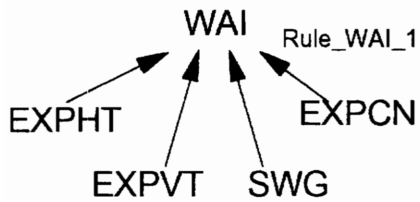

WGEAR
CE_WGEAR_1
WGTO

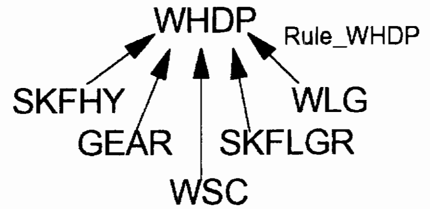

WGEAR
CE_WGEAR_2
CE_WGEAR_3
CE_WGEAR_4
PASS

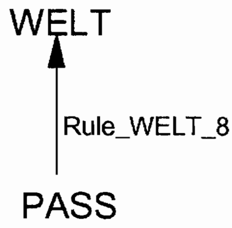

WAIRC
Rule_WAIRC_1
WAI

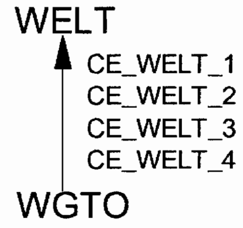

WAIRC
Rule_WAIRC_2
DPCABN
WAI
SKFAC
BDMAX
BODL

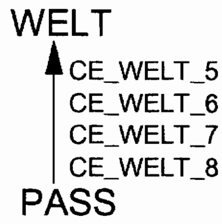

WAIRC
CE_WAIRC
WGTO

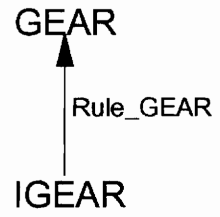

WAI
Rule_WAI_1
EXPHT
EXPCN
EXPVT
SWG

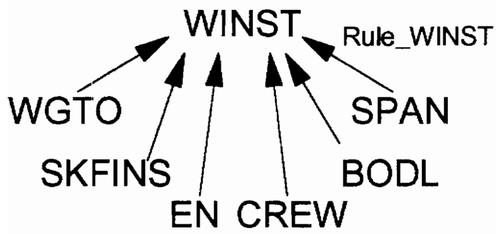

WHDP
Rule_WHDP
SKFHY
WLG
GEAR
SKFLGR
WSC

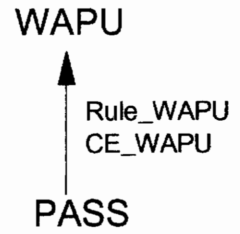

WELT
Rule_WELT_8
PASS

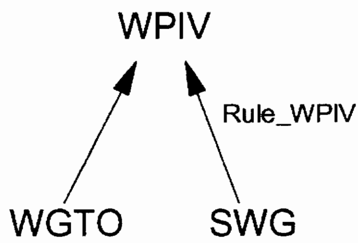

WELT
CE_WELT_1
CE_WELT_2
CE_WELT_3
CE_WELT_4
WGTO

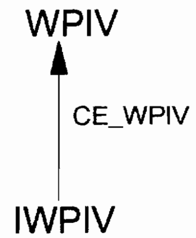

WELT
CE_WELT_5
CE_WELT_6
CE_WELT_7
CE_WELT_8
PASS

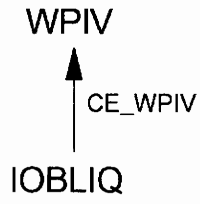

GEAR
Rule_GEAR
IGEAR

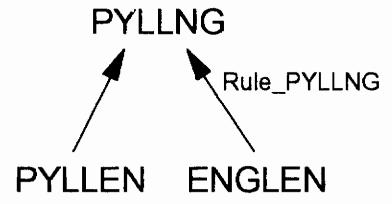

WINST
Rule_WINST
WGTO
SPAN
SKFINS
BODL
EN
CREW

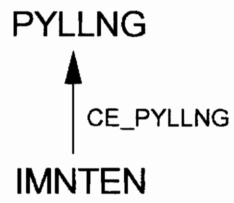

WAPU
Rule_WAPU
CE_WAPU
PASS

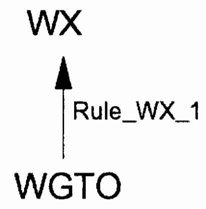

WPIV
Rule_WPIV
WGTO
SWG

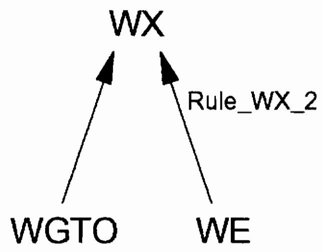

WPIV
CE_WPIV
IWPIV

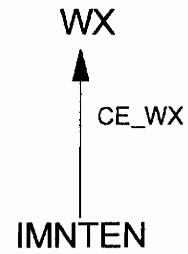

WPIV
CE_WPIV
IOBLIQ

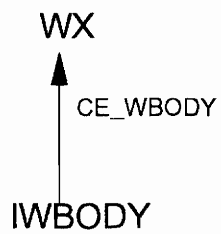

PYLLNG
Rule_PYLLNG
PYLLEN
ENGLEN

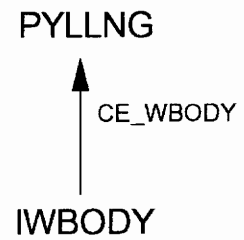

PYLLNG
CE_PYLLNG
IMNTEN

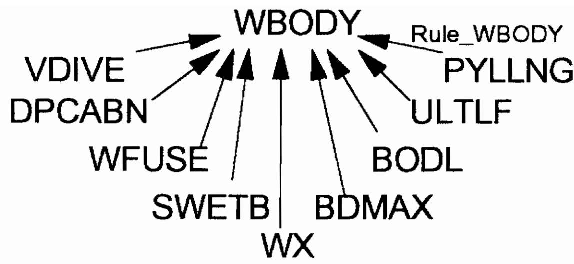

WX
Rule_WX_1
WGTO

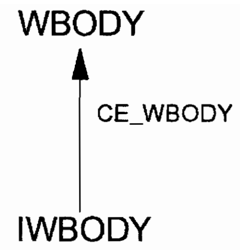

WX
Rule_WX_2
WGTO
WE

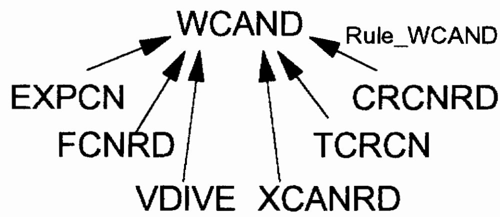

WX
CE_WX
IMNTEN

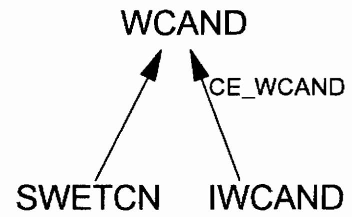

WX
CE_WBODY
IWBODY

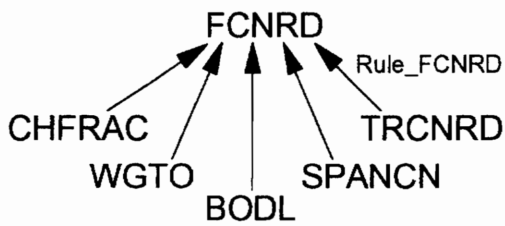

PYLLNG
CE_WBODY
IWBODY

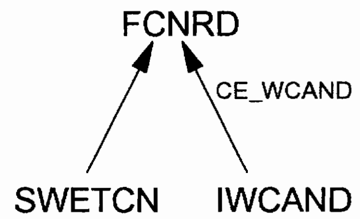

WBODY
Rule_WBODY
VDIVE
PYLLNG
DPCABN
ULTLF
WFUSE
BODL
SWETB
BDMAX
WX

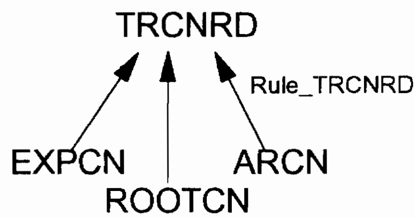

WBODY
CE_WBODY
IWBODY

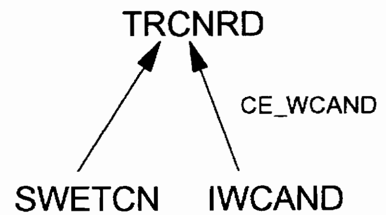

WCAND
Rule_WCAND
EXPCN
CRCNRD
FCNRD
TCRCN
VDIVE
XCANRD

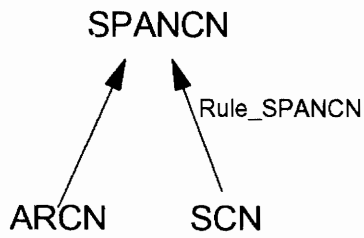

WCAND
CE_WCAND
SWETCN
IWCAND

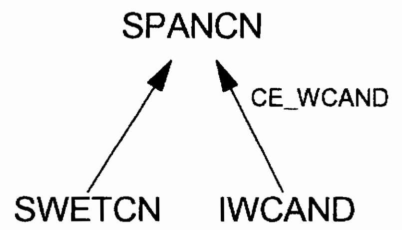

FCNRD
Rule_FCNRD
CHFRAC
TRCNRD
WGTO
SPANCN
BODL

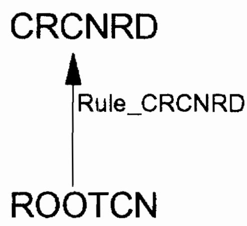

FCNRD
CE_WCAND
SWETCN
IWCAND

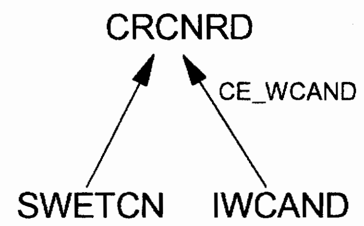

TRCNRD
Rule_TRCNRD
EXPCN
ARCN
ROOTCN

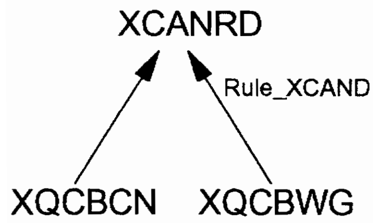

TRCNRD
CE_WCAND
SWETCN
IWCAND

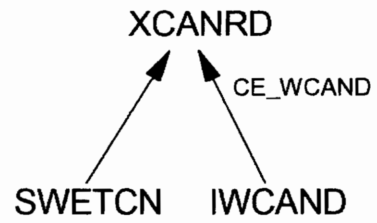

SPANCN
Rule_SPANCN
ARCN
SCN

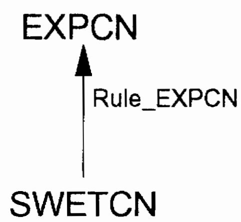

SPANCN
CE_WCAND
SWETCN
IWCAND

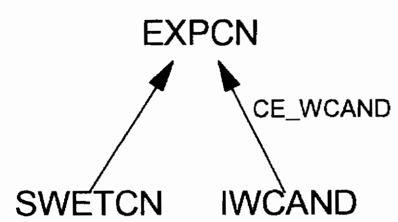

CRCNRD
Rule_CRCNRD
ROOTCN

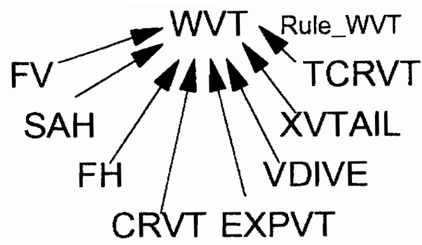

CRCNRD
CE_WCAND
SWETCN
IWCAND

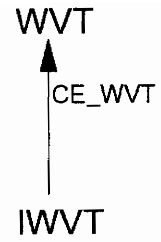

XCANRD
Rule_XCAND
XQCBCN
XQCBWG

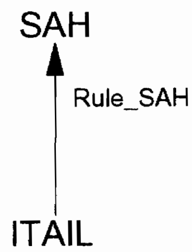

XCANRD
CE_WCAND
SWETCN
IWCAND

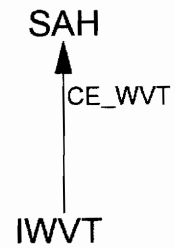

EXPCN
Rule_EXPCN
SWETCN

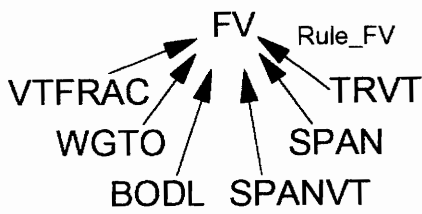

EXPCN
CE_WCAND
SWETCN
IWCAND

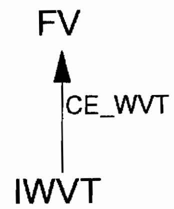

WVT
Rule_WVT
FV
TCRVT
SAH
XVTAIL
FH
VDIVE
CRVT
EXPVT

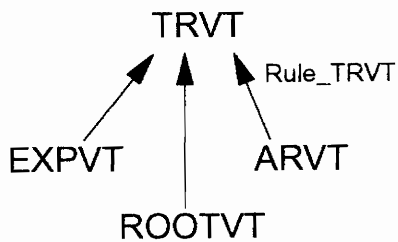

WVT
CE_WVT
IWVT

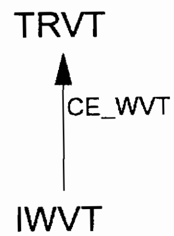

SAH
Rule_SAH
ITAIL

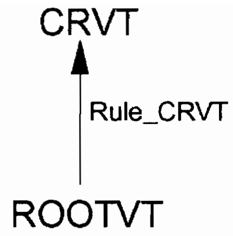

SAH
CE_WVT
IWVT

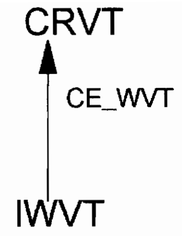

FV
Rule_FV
VTFRAC
TRVT
WGTO
SPAN
BODL
SPANVT

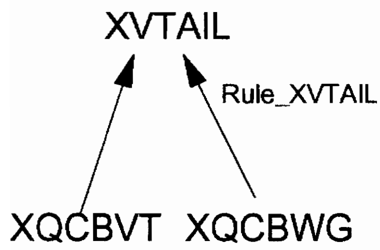

FV
CE_WVT
IWVT

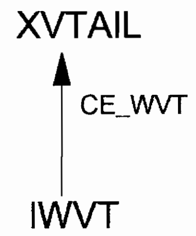

TRVT
Rule_TRVT
EXPVT
ARVT
ROOTVT

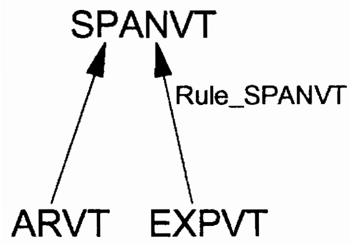

TRVT
CE_WVT
IWVT

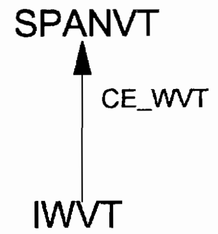

CRVT
Rule_CRVT
ROOTVT

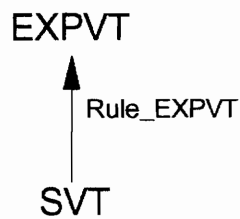

CRVT
CE_WVT
IWVT

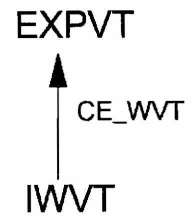

XVTAIL
Rule_XVTAIL
XQCBVT
XQCBWG

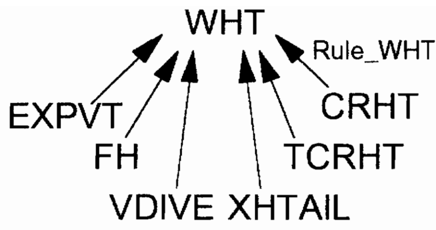

XVTAIL
CE_WVT
IWVT

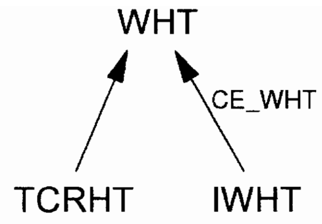

SPANVT
Rule_SPANVT
ARVT
EXPVT

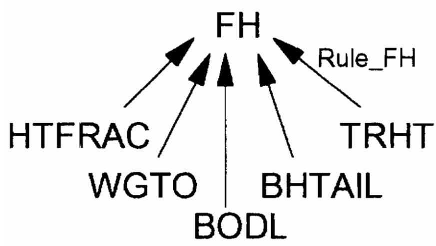

SPANVT
CE_WVT
IWVT

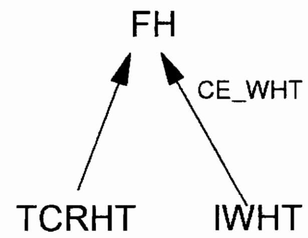

EXPVT
Rule_EXPVT
SVT

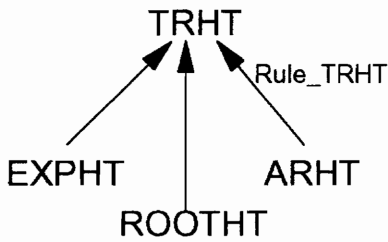

EXPVT
CE_WVT
IWVT

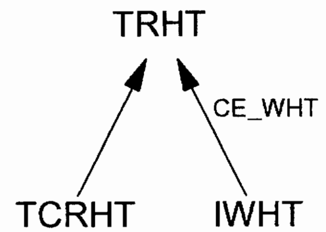

WHT
Rule_WHT
EXPVT
FH
VDIVE
XHTAIL
TCRHT
CRHT

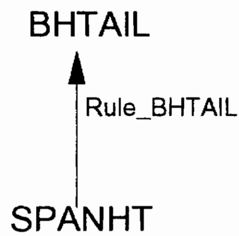

WHT
CE_WHT
TCRHT
IWHT

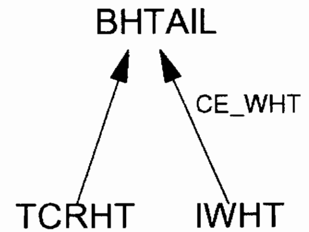

FH
Rule_FH
HTFRAC
WGTO
BODL
BHTAIL
TRHT

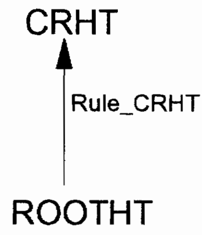

FH
CE_WHT
TCRHT
IWHT

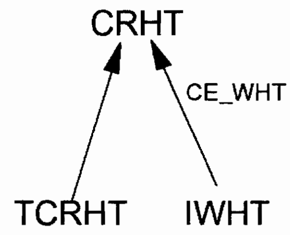

TRHT
Rule_TRHT
EXPHT
ROOTHT
ARHT

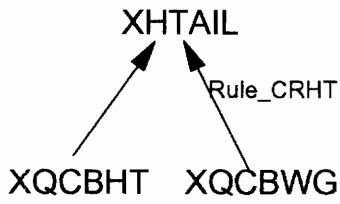

TRHT
CE_WHT
TCRHT
IWHT

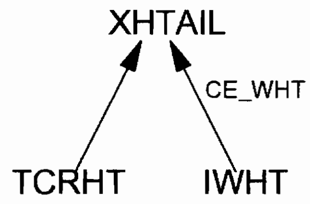

BHTAIL
Rule_BHTAIL
SPANHT

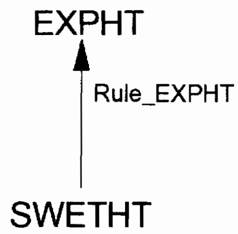

BHTAIL
CE_WHT
TCRHT
IWHT

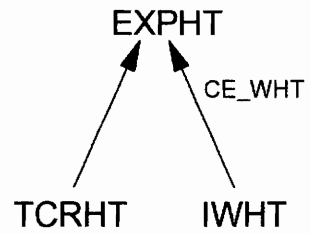

CRHT
Rule_CRHT
ROOTHT

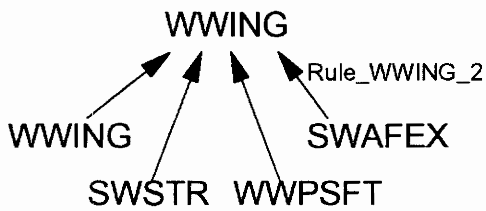

CRHT
CE_WHT
TCRHT
IWHT

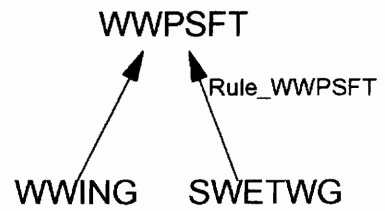

XHTAIL
Rule_CRHT
XQCBHT
XQCBWG

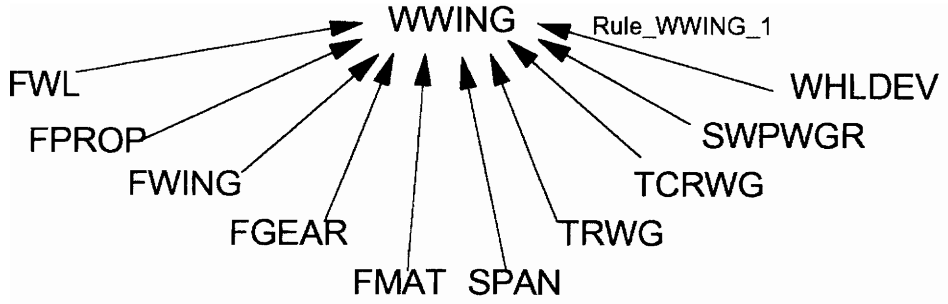

XHTAIL
CE_WHT
TCRHT
IWHT

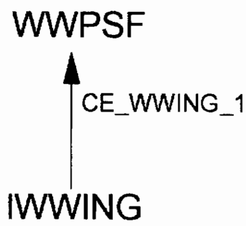

EXPHT
Rule_EXPHT
SWETHT

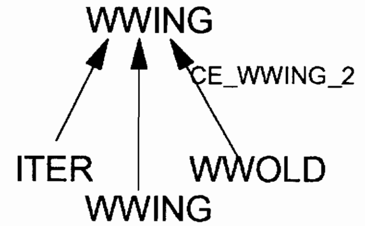

EXPHT
CE_WHT
TCRHT
IWHT

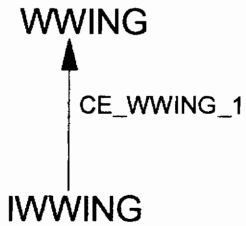

WWING
Rule_WWING_2
WWING
SWAFEX
SWSTR
WWPSFT

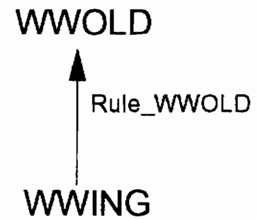

WWPSFT
Rule_WWPSFT
WWING
SWETWG

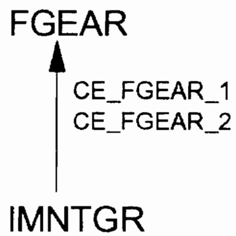

WWING
Rule_WWING_1
FWL
FPROP
FWING
FGEAR
FMAT
SPAN
TRWG
TCRWG
SWPWGR
WHLDEV

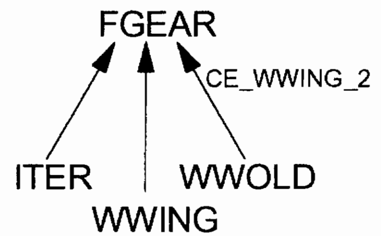

WWPSF
CE_WWING_1
IWWING

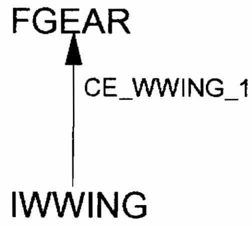

WWING
CE_WWING_2
ITER
WWOLD
WWING

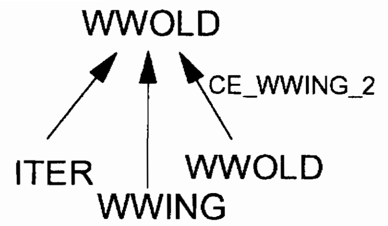

WWING
CE_WWING_1
IWWING

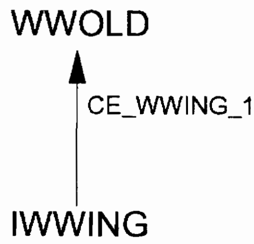

WWOLD
Rule_WWOLD
WWING

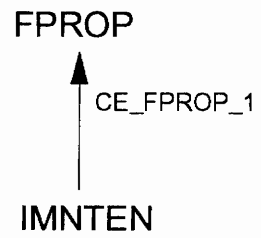

FGEAR
CE_FGEAR_1
CE_FGEAR_2
IMNTGR

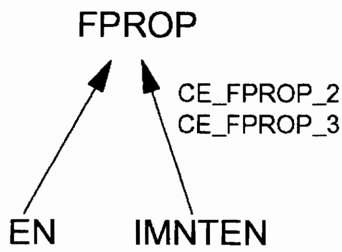

FGEAR
CE_WWING_2
ITER
WWOLD
WWING

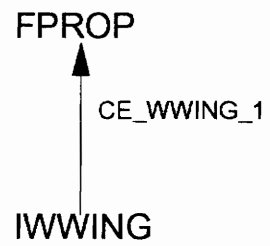

FGEAR
CE_WWING_1
IWWING

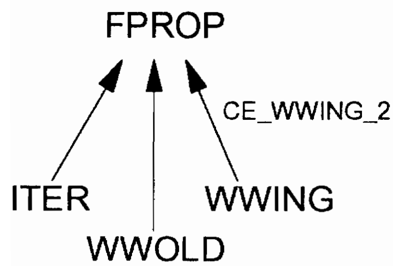

WWOLD
CE_WWING_2
ITER
WWOLD
WWING

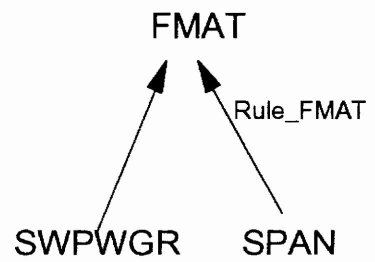

WWOLD
CE_WWING_1
IWWING

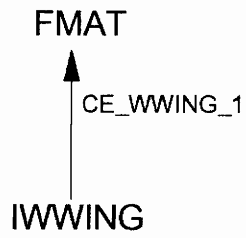

FPROP
CE_FPROP_1
IMNTEN

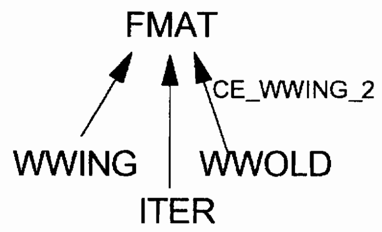

FPROP
CE_FPROP_2
CE_FPROP_3
EN
IMNTEN

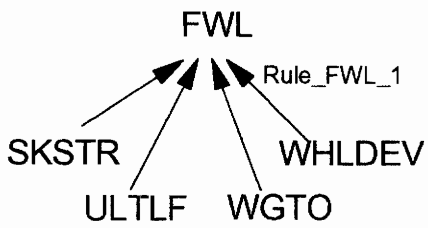

FPROP
CE_WWING_1
IWWING

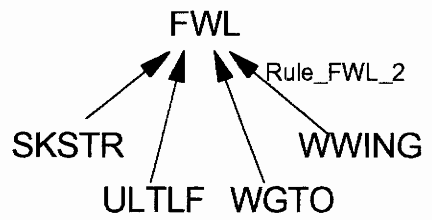

FPROP
CE_WWING_2
ITER
WWING
WWOLD

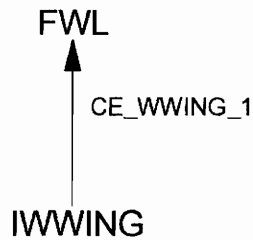

FMAT
Rule_FMAT
SWPWGR
SPAN

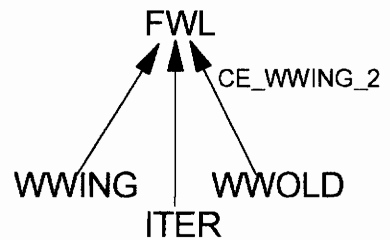

FMAT
CE_WWING_1
IWWING

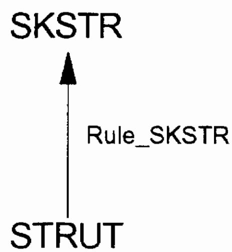

FMAT
CE_WWING_2
WWING
WWOLD
ITER

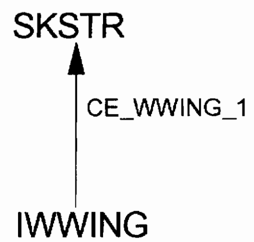

FWL
Rule_FWL_1
SKSTR
WHLDEV
ULTLF
WGTO

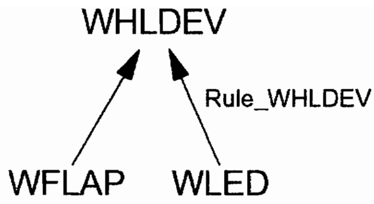

FWL
Rule_FWL_2
SKSTR
WWING
ULTLF
WGTO

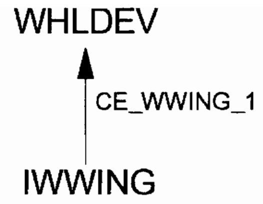

FWL
CE_WWING_1
IWWING

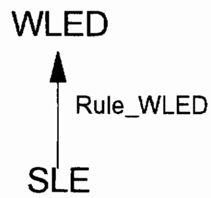

FWL
CE_WWING_2
WWING
WWOLD
ITER

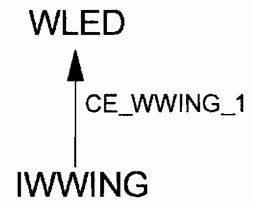

SKSTR
Rule_SKSTR
STRUT

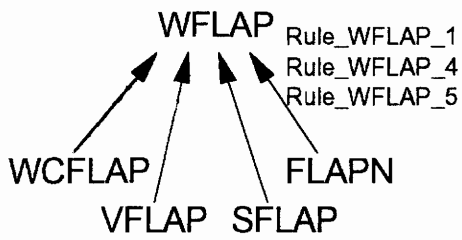

SKSTR
CE_WWING_1
IWWING

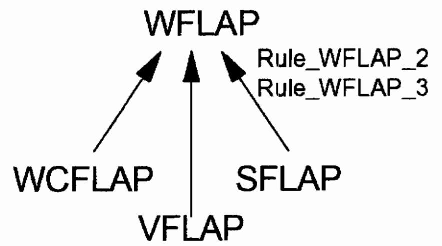

WHLDEV
Rule_WHLDEV
WFLAP
WLED

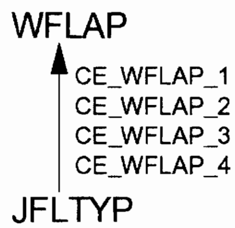

WHLDEV
CE_WWING_1
IWWING

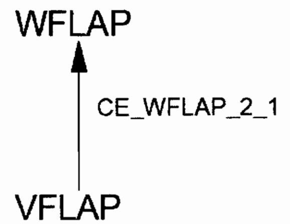

WLED
Rule_WLED
SLE

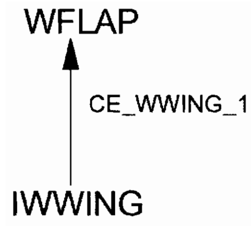

WLED
CE_WWING_1
IWWING

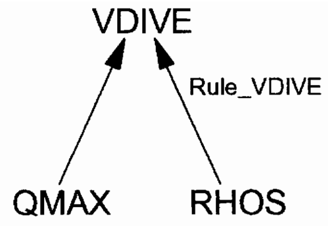

WFLAP
Rule_WFLAP_1
Rule_WFLAP_4
Rule_WFLAP_5
WCFLAP
FLAPN
VFLAP
SFLAP

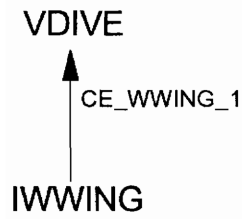

WFLAP
Rule_WFLAP_2
Rule_WFLAP_3
WCFLAP
SFLAP
VFLAP

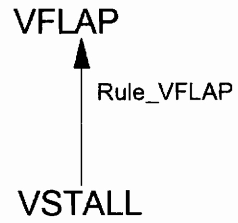

WFLAP
CE_WFLAP_1
CE_WFLAP_2
CE_WFLAP_3
CE_WFLAP_4
JFLTYP

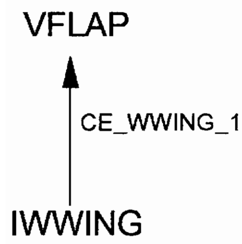

WFLAP
CE_WFLAP_2_1
VFLAP

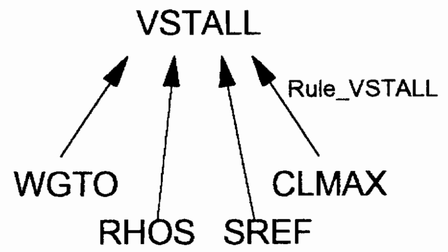

WFLAP
CE_WWING_1
IWWING

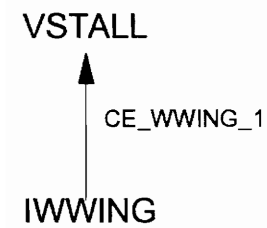

VDIVE
Rule_VDIVE
QMAX
RHOS

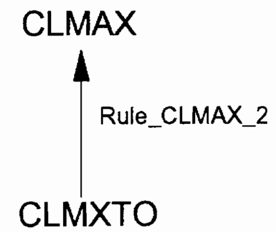

VDIVE
CE_WWING_1
IWWING

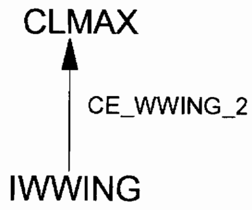

VFLAP
Rule_VFLAP
VSTALL

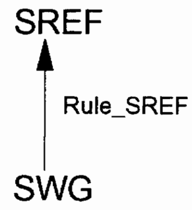

VFLAP
CE_WWING_1
IWWING

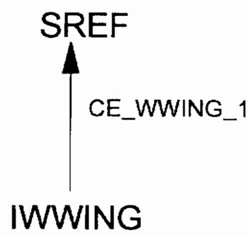

VSTALL
Rule_VSTALL
WGTO
CLMAX
RHOS
SREF

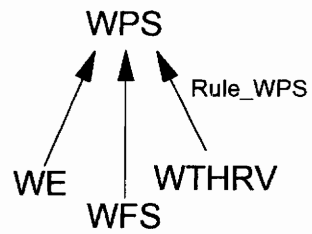

VSTALL
CE_WWING_1
IWWING

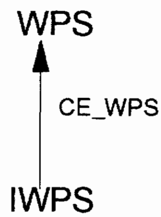

CLMAX
Rule_CLMAX_2
CLMXTO

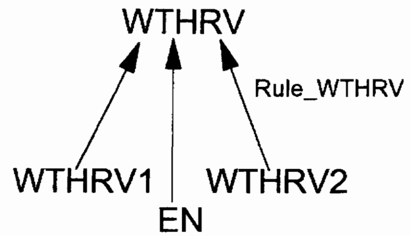

CLMAX
CE_WWING_2
IWWING

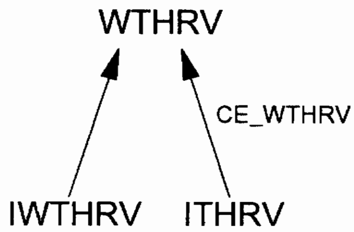

SREF
Rule_SREF
SWG

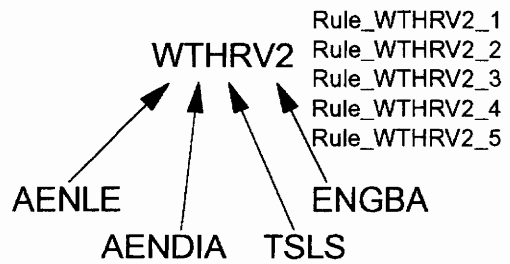

SREF
CE_WWING_1
IWWING

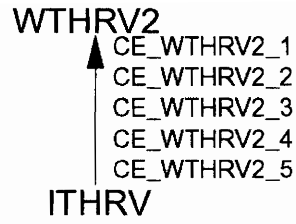

WPS
Rule_WPS
WE
WTHRV
WFS

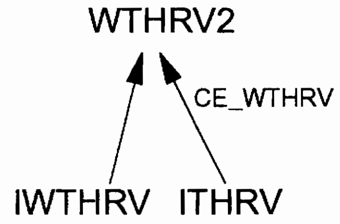

WPS
CE_WPS
IWPS

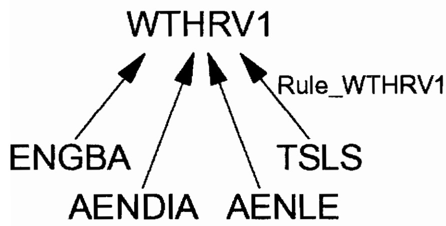

WTHRV
Rule_WTHRV
WTHRV1
WTHRV2
EN

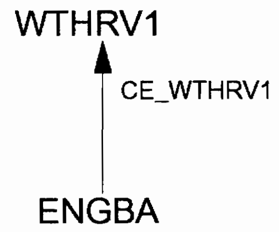

WTHRV
CE_WTHRV
IWTHRV
ITHRV

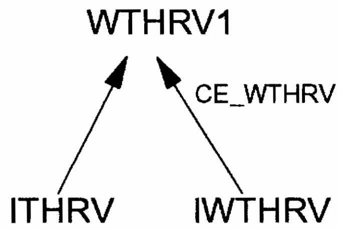

Rule_WTHRV2_1
WTHRV2
Rule_WTHRV2_2
Rule_WTHRV2_3
Rule_WTHRV2_4
Rule_WTHRV2_5
AENLE
ENGBA
AENDIA
TSLS

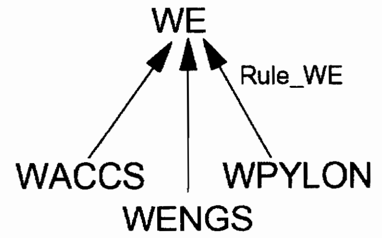

WTHRV2
CE_WTHRV2_1
CE_WTHRV2_2
CE_WTHRV2_3
CE_WTHRV2_4
CE_WTHRV2_5
ITHRV

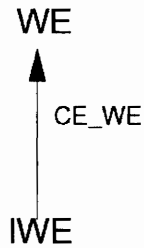

WTHRV2
CE_WTHRV
IWTHRV
ITHRV

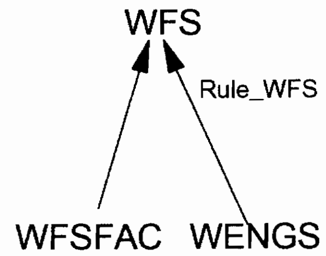

WTHRV1
Rule_WTHRV1
ENGBA
TSLS
AENDIA
AENLE

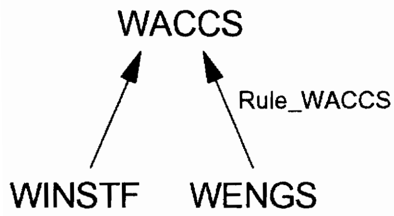

WTHRV1
CE_WTHRV1
ENGBA

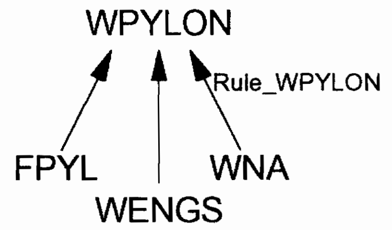

WTHRV1
CE_WTHRV
ITHRV
IWTHRV

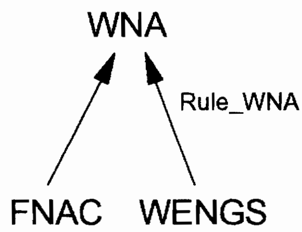

WE
Rule_WE
WACCS
WPYLON
WENGS

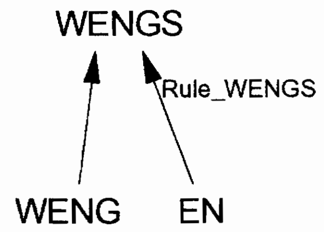

WE
CE_WE
IWE

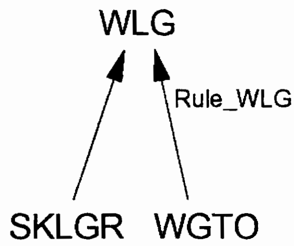

WFS
Rule_WFS
WFSFAC
WENGS

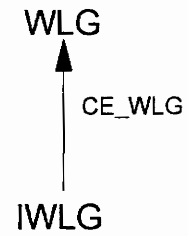

WACCS
Rule_WACCS
WINSTF
WENGS

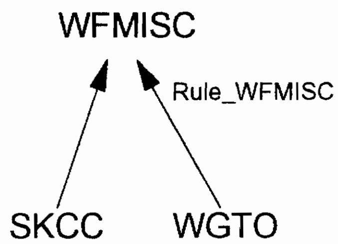

WPYLON
Rule_WPYLON
FPYL
WNA
WENGS

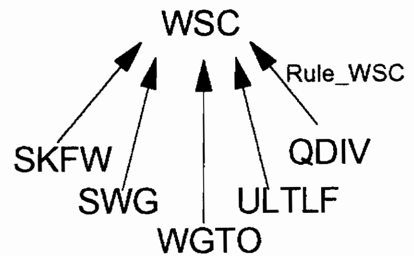

WNA
Rule_WNA
FNAC
WENGS

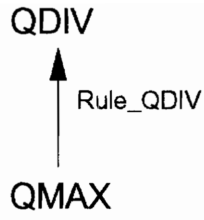

WENGS
Rule_WENGS
WENG
EN

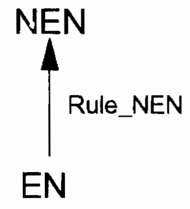

WLG
Rule_WLG
SKLGR
WGTO

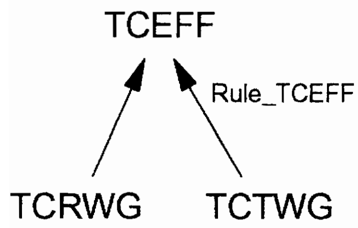

WLG
CE_WLG
IWLG

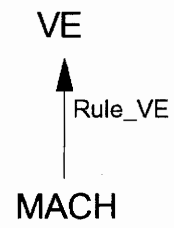

WFMISC
Rule_WFMISC
SKCC
WGTO

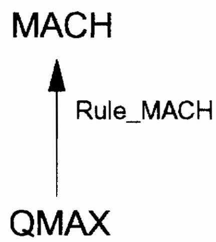

WSC
Rule_WSC
SKFW
QDIV
SWG
ULTLF
WGTO

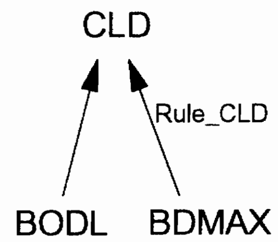

QDIV
Rule_QDIV
QMAX

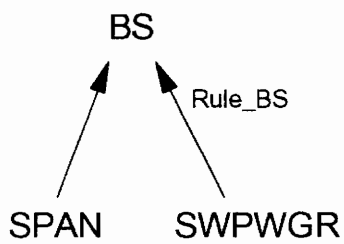

NEN
Rule_NEN
EN

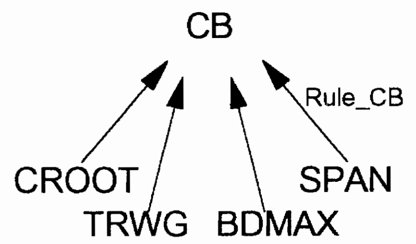

TCEFF
Rule_TCEFF
TCRWG
TCTWG

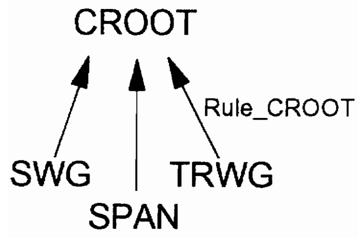

VE
Rule_VE
MACH

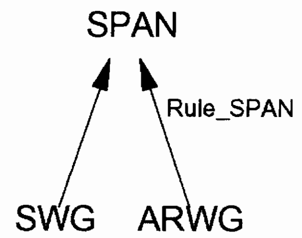

MACH
Rule_MACH
QMAX

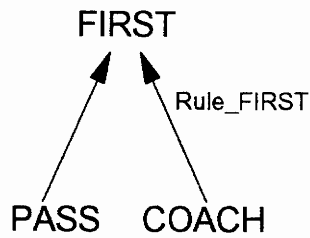

CLD
Rule_CLD
BODL
BDMAX

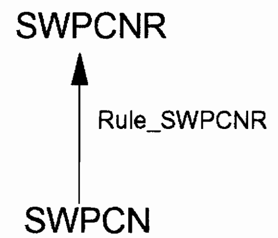

BS
Rule_BS
SPAN
SWPWGR

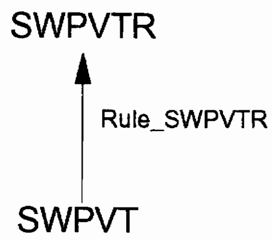

CB
Rule_CB
CROOT
SPAN
TRWG
BDMAX

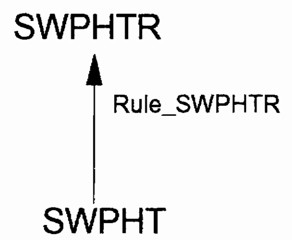

CROOT
Rule_CROOT
SWG
TRWG
SPAN

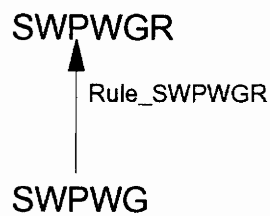

SPAN
Rule_SPAN
SWG
ARWG


FIRST
Rule_FIRST
PASS
COACH


SWPCNR
Rule_SWPCNR
SWPCN


SWPVTR
Rule_SWPVTR
SWPVT


SWPHTR
Rule_SWPHTR
SWPHT


SWPWGR
Rule_SWPWGR
SWPWG

Appendix B - Knowledge Files

Weight.kb

Parameter	AMPRWT	float	0.0	0
Parameter	CARGO	float	0.0	0
Parameter	CNFRAC	float	.18	0
Parameter	ETC	float	0.0	0
Parameter	FCNRD	float	0.0	0
Parameter	FH	float	0.0	0
Parameter	FNAC	float	0.0	0
Parameter	FPYL	float	0.0	1
Parameter	FV	float	0.0	0
Parameter	FWL	float	0.0	0
Parameter	GEAR	float	1	0
Parameter	HTFRAC	float	.18	0
Parameter	IGEAR	float	0.0	0
Parameter	ITHRV	float	0.0	0
Parameter	IWAF	float	0.0	0
Parameter	IWBODY	float	0.0	0
Parameter	IWCAND	float	0.0	0
Parameter	IWCREW	float	0.0	0
Parameter	IWE	float	0.0	0
Parameter	IWFEQ	float	0.0	0
Parameter	IWHT	float	0.0	0
Parameter	IWLG	float	0.0	0
Parameter	IWPIV	float	0.0	0
Parameter	IWPS	float	0.0	0
Parameter	IWTHRV	float	0.0	0
Parameter	IWVT	float	0.0	0
Parameter	IWWING	float	0.0	0
Parameter	PART	float	0.0	0
Parameter	QDIV	float	700.0	0
Parameter	QMAX	float	700	1
Parameter	SKCC	float	11	1
Parameter	SKFAC	float	1	1
Parameter	SKFCAT	float	.7	1
Parameter	SKFHY	float	.1	1
Parameter	SKFH2O	float	1.0	1
Parameter	SKFINS	float	.0862	1

Parameter	SKFLGR	float	.16	1
Parameter	SKFSRV	float	2.0	1
Parameter	SKFW	float	.404	1
Parameter	SKLGR	float	.032	1
Parameter	SKSTR	float	0.0	1
Parameter	ULTLF	float	3.75	1
Parameter	VTFRAC	float	.22	0
Parameter	WACCS	float	0.0	0
Parameter	WAF	float	0.0	1
Parameter	WAI	float	0.0	0
Parameter	WAIRC	float	0.0	0
Parameter	WAPU	float	0.0	0
Parameter	WATTD	float	0.0	0
Parameter	WBAG	float	0.0	0
Parameter	WBB1	float	0.0	1
Parameter	WBB2	float	0.0	1
Parameter	WBODY	float	0.0	1
Parameter	WBPPAS	float	0.0	1
Parameter	WCA	float	0.0	0
Parameter	WCABG	float	0.0	1
Parameter	WCAND	float	0.0	1
Parameter	WCARGO	float	0.0	1
Parameter	WCFLAP	float	.733	0
Parameter	WCPPAS	float	0.0	1
Parameter	WCREW	float	0.0	0
Parameter	WCRGCT	float	0.0	0
Parameter	WE	float	0.0	1
Parameter	WELT	float	0.0	27
Parameter	WEMPOP	float	0.0	0
Parameter	WEMPTY	float	0.0	0
Parameter	WENG	float	1421	1
Parameter	WENGS	float	2842	0
Parameter	WEP	float	0.0	0
Parameter	WERR	float	0.0	0
Parameter	WFEMER	float	0.0	0
Parameter	WFEQ	float	0.0	0
Parameter	WFFURN	float	0.0	0
Parameter	WFLAP	float	0.0	0
Parameter	WFMISC	float	0.0	0
Parameter	WFR	float	0.0	0
Parameter	WFS	float	0.0	0
Parameter	WFSFAC	float	.0195	0
Parameter	WFTOT	float	0.0	0
Parameter	WFTRAP	float	100	0

```
Parameter        WFUSE    float   136     1
Parameter        WGALLY   float   0.0     0
Parameter        WGEAR    float   0.0     1
Parameter        WGTO     float   10000   1
Parameter        WHDP     float   0.0     0
Parameter        WHLDEV   float   0.0     0
Parameter        WHT      float   0.0     1
Parameter        WH2O     float   0.0     0
Parameter        WINST    float   0.0     0
Parameter        WINSTF   float   .135    1
Parameter        WLED     float   0.0     0
Parameter        WLG      float   0.0     1
Parameter        WNA      float   0.0     0
Parameter        WOIL     float   0.0     0
Parameter        WOPIT    float   0.0     0
Parameter        WPA      float   0.0     0
Parameter        WPACFC   float   0.0     0
Parameter        WPASS    float   0.0     0
Parameter        WPASSV   float   0.0     0
Parameter        WPIV     float   0.0     1
Parameter        WPL      float   0.0     0
Parameter        WPPAS    float   0.0     1
Parameter        WPS      float   0.0     1
Parameter        WPYLON   float   0.0     0
Parameter        WS       float   0.0     0
Parameter        WSC      float   0.0     0
Parameter        WTHRV1   float   0.0     0
Parameter        WTHRV2   float   0.0     0
Parameter        WVT      float   0.0     1
Parameter        WWING    float   0.0     1
Parameter        WWOLD    float   0.0     0
Parameter        WWPSFT   float   0.0     0
Parameter        WX       float   0.0     0
Eqn_Rule         Rule_QDIV        1
1 QDIV = QMAX
Action  0
Eqn_Rule         Rule_WSC 1
5 WSC = SKFW * pow ( SWG , 0.317 ) * pow ( ( WGTO / 1000. ) , 0.602 ) * pow (
ULTLF , 0.525 ) * pow ( QDIV , 0.345 ) Action  0
Eqn_Rule         Rule_WFMISC      1
2 WFMISC = SKCC * pow ( ( WGTO / 1000. ) , 0.41 )
Action  0
Eqn_Rule         Rule_WLG 1
2 WLG = SKLGR * WGTO
```

```
Action 0
Eqn_Rule                Rule_WENGS            1
2 WENGS = WENG * EN
Action 0

Eqn_Rule   Rule_WNA1
2 WNA = FNAC * WENGS
Action 0
Eqn_Rule   Rule_WPYLON                        1
3 WPYLON = FPYL * pow ( (WENGS + WNA ) , 0.736 )
Action 0
Eqn_Rule   Rule_WACCS                         1
2 WACCS = WINSTF * WENGS
Action 0
Eqn_Rule   Rule_WFS 1
2 WFS = WFSFAC * WENGS
Action 0
Eqn_Rule   Rule_WE 1
3 WE = WACCS + WENGS + WPYLON
Action 0
Eqn_Rule   Rule_WTHRV1                        1
5 WTHRV1 = (.218 * AENDIA * ( AENLE / 4. ) + .012 * ( TSLS - ( TSLS / (ENGBA
+ 1 ) ) ) )
Action 0
Eqn_Rule   Rule_WTHRV2_1                      1
4 WTHRV2 = ( .179 * ( AENDIA /2. ) * ( ENGLE / 7.5 ) + .0389 * ( TSLS / ( ENGBA
+ 1 ) ) )
Action 0
Eqn_Rule   Rule_WTHRV2_2                      1
4 WTHRV2 = ( .131 * ( AENDIA /2. ) * ( ENGLE / 7.5 ) + .0289 * ( TSLS / ( ENGBA
+ 1 ) ) )
Action 0
Eqn_Rule   Rule_WTHRV2_3                      1
4 WTHRV2 = ( .105 * ( AENDIA /2. ) * ( ENGLE / 7.5 ) + .0122 * ( TSLS / ( ENGBA
+ 1 ) ) )
Action 0
Eqn_Rule   Rule_WTHRV2_4                      1
4 WTHRV2 = ( .113 * ( AENDIA /2. ) * ( ENGLE / 7.5 ) + .0144 * ( TSLS / ( ENGBA
+ 1 ) ) )
Action 0
Eqn_Rule   Rule_WTHRV2_5                      1
4 WTHRV2 = ( .096 * ( AENDIA /2. ) * ( ENGLE / 7.5 ) + .0094 * ( TSLS / ( ENGBA
+ 1 ) ) )
Action 0
```

```
Eqn_Rule   Rule_WTHRV                    1
3 WTHRV = ( WTHRV1 + WTHRV2 ) * EN
Action  0
Eqn_Rule   Rule_WPS 1
3 WPS = WE + WFS + WTHRV
Action  0
Eqn_Rule   Rule_WCFLAP_1                 1
0 WCFLAP = 0.62
Action  0
Eqn_Rule   Rule_WCFLAP_2                 1
0 WCFLAP = 1.0
Action  0
Eqn_Rule   Rule_WCFLAP_3                 1
0 WCFLAP = 0.733
Action  0
Eqn_Rule   Rule_WCFLAP_4                 1
0 WCFLAP = 1.2
Action  0
Eqn_Rule   Rule_WCFLAP_5                 1
0 WCFLAP = 1.32
Action  0
Eqn_Rule   Rule_WCFLAP_6                 1
0 WCFLAP = .633
Action  0
Eqn_Rule   Rule_WCFLAP_7                 1
0 WCFLAP = .678
Action  0
Eqn_Rule   Rule_WFLAP_1                  1
4 WFLAP = WCFLAP * pow ( ( VFLAP / 100. ) , 2 ) * SFLAP / sqrt ( FLAPN )
Action  0
Eqn_Rule   Rule_WFLAP_2                  1
3 WFLAP = WCFLAP * SFLAP * 0.369 * pow ( VFLAP , .2733 )
Action  0
Eqn_Rule   Rule_WFLAP_3                  1
3 WFLAP = WCFLAP * SFLAP * ( pow ( VFLAP , 2.195 ) ) / 45180.
Action  0
Eqn_Rule   Rule_WFLAP_4                  1
4 WFLAP = WCFLAP * pow ( ( VFLAP / 100. ) , 2 ) * SFLAP * pow ( FLAPN , 0.5 )
Action  0
Eqn_Rule   Rule_WFLAP_5                  1
4 WFLAP = WCFLAP * pow ( ( VFLAP / 100. ) , 2.38 ) * pow ( SFLAP , 1.19 ) / pow (
FLAPN , .595 ) Action  0
Eqn_Rule   Rule_WLED                     1
1 WLED = 3.28 * pow ( SLE , 1.13 )
```

Action 0
Eqn_Rule Rule_WHLDEV 1
2 WHLDEV = WFLAP + WLED
Action 0
Eqn_Rule Rule_SKSTR 1
1 SKSTR = 1. - STRUT * STRUT
Action 0
Eqn_Rule Rule_FWL_1 1
5 FWL = pow (SKSTR * ULTLF * (WGTO - .8 * (.08 * WGTO + WHLDEV)) , .757)
Action 0
Eqn_Rule Rule_FWL_2 1
4 FWL = pow (SKSTR * ULTLF * (WGTO - .8 * WWING) , .757)
Action 0
Eqn_Rule Rule_WWOLD 1
1 WWOLD = WWING
Action 0
Eqn_Rule Rule_WWING_1 1
10 WWING = (FWL * FWING * FMAT * FPROP * FGEAR * pow (SPAN , 1.049) * pow ((1. + TRWG) , 0.4) / (pow (10 , 5) * pow (TCRWG , 0.4) * pow (cos (SWPWGR) , 1.535))
Action 0
Eqn_Rule Rule_WWPSFT 1
2 WWPSFT = WWING / SWETWG
Action 0

Eqn_Rule Rule_WWING_2 1
4 WWING = WWING + SWSTR * WWPSFT + SWAFEX * WWPSFT
Action 0
Eqn_Rule Rule_FH 1
5 FH = HTFRAC * WGTO * BODL * BHTAIL * (1. + 2. * TRHT) / (pow (10. , 6) * (1. + TRHT))
Action 0
Eqn_Rule Rule_WHT_1 1
6 WHT = 350. * pow (((EXPHT * FH * log10 (VDIVE)) / (100. * XHTAIL * TCRHT * CRHT)) , .54)
Action 0
Eqn_Rule Rule_WHT_2 1
0 WHT = 0
Action 0
Eqn_Rule Rule_FV 1
6 FV = 0.5 * VTFRAC * WGTO * (BODL + SPAN) * (1. + 2. * TRVT) / (pow (10 , 6) * (1 . + TRVT))
Action 0

Eqn_Rule Rule_WVT 1
8 WVT = 380. * pow ((((FV + SAH * 0.5 * FH) * EXPVT * log10 (VDIVE)) / (100. * XVTAIL * TCRVT * CRVT)) , 0.54) * VTNO
Action 0
Eqn_Rule Rule_FCNRD 1
5 FCNRD = CNFRAC * WGTO * BODL * SPANCN * (1. + 2. * TRCNRD) / (pow (10. , 6) * (1. + TRCNRD))
Action 0
Eqn_Rule Rule_WCAND_1 1
6 WCAND = 350. * pow (((EXPCN * FCNRD * log (VDIVE)) / (100. * XCANRD * TCRCN * CRCNRD)) , .54)
Action 0
Eqn_Rule Rule_WCAND_2 1
0 WCAND = 0
Action 0
Eqn_Rule Rule_WX_1 1
1 WX = WGTO
Action 0
Eqn_Rule Rule_WX_2 1
2 WX = WGTO - WE
Action 0
Eqn_Rule Rule_WBODY 1
9 WBODY = WFUSE * pow (pow (WX / 1000 , 0.7) * (SWETB / 1000.) * BDMAX * sqrt (BODL + PYLLNG) * log10 (VDIVE) * pow (DPCABN , 0.2) * pow (ULTLF , 0.3) , .508)
Action 0
Eqn_Rule Rule_WPIV 1
3 WPIV = 0.13 * pow ((WGTO / SWG) , .2454) * pow (SWG , 1.046) + 1903. + 0.0007967 * (WGTO - 500000.)
Action 0
Eqn_Rule Rule_WAPU1
1 WAPU = 26.2 * pow (PASS , 0.944) - 13.6 * PASS
Action 0
Eqn_Rule Rule_WINST 1
6 WINST = SKFINS * pow (WGTO , 0.386) * pow (EN , 0.687) * pow (CREW , 0.31) * pow (BODL , 0.05) * pow (SPAN , 0.696)
Action 0
Eqn_RuleRule_GEAR 1
1 GEAR = 1 - IGEAR
Action 0
Eqn_RuleRule_WHDP 1
5 WHDP = SKFHY * WSC + SKFLGR * WLG * GEAR
Action 0
Eqn_Rule Rule_WEP 1

```
1 WEP = 16.2 * PASS + 110.
Action  0
Eqn_Rule        Rule_WELT_1             1
0 WELT = 65.
Action  0
Eqn_Rule        Rule_WELT_2             1
0 WELT = 113.
Action  0
Eqn_Rule        Rule_WELT_3             1
0 WELT = 163.
Action  0
Eqn_Rule        Rule_WELT_4             1
0 WELT = 340.
Action  0
Eqn_Rule        Rule_WELT_5             1
0 WELT = 400.
Action  0
Eqn_Rule        Rule_WELT_6             1
0 WELT = 500.
Action  0
Eqn_Rule        Rule_WELT_7             1
0 WELT = 600.
Action  0
Eqn_Rule        Rule_WELT_8             1
1 WELT = 2.8 * PASS + 1010.
Action  0
Eqn_Rule        Rule_WAI_1              1
4 WAI = 22.7 * sqrt ( SWG + EXPHT + EXPVT + EXPCN ) - 385
Action  0
Eqn_Rule        Rule_WAI_2              1
0 WAI = 0
Action  0
Eqn_Rule        Rule_WAIRC_1            1
1 WAIRC = 5. + WAI
Action  0
Eqn_Rule        Rule_WAIRC_2            1
5 WAIRC = SKFAC * ( 1.5 + DPCABN ) * sqrt ( 0.358 * BODL + pow ( BDMAX , 2 )
) + WAI
Action  0
Eqn_Rule        Rule_WGEAR_1            1
0 WGEAR = 3.0
Action  0
Eqn_Rule        Rule_WGEAR_2            1
0 WGEAR =   10.
```

```
Action 0
Eqn_Rule        Rule_WGEAR_3            1
0 WGEAR =   20.
Action 0
Eqn_Rule  Rule_WGEAR_4                  1
0 WGEAR = 50.
Action 0
Eqn_Rule  Rule_WFFURN_1                 1
2 WFFURN = FFUR * PASS + 310.
Action 0
Eqn_Rule Rule_WFFURN_2  1
1 WFFURN = 118.4 * PASS - 4190.
Action 0
Eqn_Rule  Rule_WFFURN_3                 1
1 WFFURN = 0.065 * WGTO - 59.
Action 0
Eqn_Rule  Rule_WCREW                    1
1 WCREW = CREW * 170.
Action 0
Eqn_Rule  Rule_WATTD                    1
2 WATTD = 130. *  floor ( STEWFC + STEWCC )
Action 0
Eqn_Rule  Rule_WCABG_1                  1
1 WCABG = 25. * CREW
Action 0
Eqn_Rule  Rule_WCABG_2                  1
3 WCABG = 10. * ( CREW + STEWFC + STEWCC ) + 25.
Action 0
Eqn_Rule  Rule_WCABG_3                  1
3 WCABG = 20. * ( CREW + STEWFC + STEWCC ) + 25. * CREW
Action 0
Eqn_Rule  Rule_WPACFC                   1
3 WPACFC = SKFSRV * PASS + 16. * NLAV
Action 0
Eqn_Rule  Rule_WPA 1
2 WPA = WPACFC + WFFURN
Action 0
Eqn_Rule  Rule_WH2O_1                   1
7 WH2O = SKFH2O * ( CREW + PASS + STEWFC * FIRST + STEWCC * CAOCH )
Action 0
Eqn_Rule  Rule_WH2O_2                   1
0 WH2O = 0
Action 0
Eqn_Rule  Rule_WGALLY_1                 1
```

2 WGALLY = SKFCAT * PASS
Action 0
Eqn_Rule Rule_WGALLY_2 1
0 WGALLY = 0
Action 0
Eqn_Rule Rule_WFEMER_1 1
0 WFEMER = 10.
Action 0
Eqn_Rule Rule_WFEMER_2 1
0 WFEMER = 15.
Action 0
Eqn_Rule Rule_WFEMER_3 1
2 WFEMER = 25. * (STEWFC + STEWCC) + 15.
Action 0
Eqn_Rule Rule_WPASSV_1 1
3 WPASSV = WH2O + WGALLY + WFEMER
Action 0
Eqn_Rule Rule_WPASSV_2 1
2 WPASSV = WCPPAS * PASS
Action 0
Eqn_Rule Rule_CARGO 1
3 CARGO = WCARGO + WBB1 + WBB2
Action 0
Eqn_Rule Rule_WPASS 1
2 WPASS = PASS * WPPAS
Action 0
Eqn_Rule Rule_WBAG 1
2 WBAG = PASS * WBPPAS
Action 0
Eqn_Rule Rule_WPL 1
3 WPL = WPASS + WBAG + CARGO
Action 0
Eqn_Rule Rule_WCRGCT 1
1 WCRGCT = 0.10 * WCARGO
Action 0
Eqn_Rule Rule_WFEQ 1
10 WFEQ = WHDP + WEP + WAIRC + WELT + WINST + WAPU + WPA + WSC + WCA + WGEAR
Action 0
Eqn_Rule Rule_WOPIT 1
7 WOPIT = WCREW + WFTRAP + WOIL + WCABG + WATTD + WPASSV + WCRGCT
Action 0
Eqn_Rule Rule_WAF 1

```
8 WAF = WWING + WBODY + WVT + WHT + WCAND + WNA + WLG + WPIV
Action  0
Eqn_Rule   Rule_WFR 1
0 WFR = 0
Action  0
Eqn_Rule   Rule_WTSUM                    1
8 WTSUM = WAF + WPS + WFEQ + WFR + WPL + WOPIT + WFTOT - WFTRAP
Action  0
Eqn_Rule   Rule_WS 1
2 WS = WTSUM / SWG
Action  0
Eqn_Rule   Rule_WEMPTY                   1
5 WEMPTY = WTSUM - WFTOT + WFTRAP + WPL + WOPIT
Action  0
Eqn_Rule   Rule_WEMPOP                   1
2 WEMPOP = WEMPTY + WOPIT
Action  0

Eqn_Rule   Rule_ETC  1
1 ETC = 0.15 * WLG
Action  0
Eqn_Rule   Rule_PART                     1
7 PART = ETC + WE + WINST + WELT + WEP + WAIRC + WAPU
Action  0
Eqn_Rule   Rule_AMPRWT                   1
2 AMPRWT = WEMPTY - PART
Action  0
Eqn_Rule   Rule_ERR  1
2 ERR = ( WTSUM / WGTO - 1. ) * 100
Action  0
Control_Eqn          CE_WLG  IWLG == 0
THEN      1
Activate_Rule        Rule_WLG
ELSE      0
Control_Eqn          CE_WE   IWE == 0
THEN      1
Activate_Rule        Rule_WE
ELSE      0
Control_Eqn CE_WTHRV    IWTHRV == 0 && ITHRV > 0 THEN       3
Activate_Rule        CE_WTHRV1
Activate_Rule        CE_WTHRV2_1
Activate_Rule        Rule_WTHRV
ELSE      0
Control_Eqn          CE_WTHRV1           ITHRV > 1
```

```
THEN	1
Activate_Rule		Rule_WTHRV1
ELSE	0
Control_Eqn		CE_WTHRV2_1	ITHRV == 1
THEN	1
Activate_Rule		Rule_WTHRV2_1
ELSE	1
Fire_Rule	CE_WTHRV2_2
Control_Eqn		CE_WTHRV2_2	ITHRV == 2
THEN	1
Activate_Rule		Rule_WTHRV2_2
ELSE	1
Fire_Fule	CE_WTHRV2_3
Control_Eqn		CE_WTHRV2_3	ITHRV == 3
THEN	1
Activate_Rule		Rule_WTHRV2_3
ELSE	1
Fire_Rule	CE_WTHRV2_4
Control_Eqn		CE_WTHRV2_4	ITHRV == 4
THEN	1
Activate_Rule		Rule_WTHRV2_4
ELSE	1
Fire_Rule	CE_WTHRV2_5
Control_Eqn		CE_WTHRV2_5	ITHRV == 5
THEN	1
Activate_Rule		Rule_WTHRV2_5
ELSE	0
Control_Eqn		CE_WPS	IWPS == 0
THEN	1
Activate_Rule		Rule_WPS
ELSE	0
Control_Eqn		CE_WWING_1	IWWING == 0
THEN	13
Activate_Rule		CE_WCFLAP_1
Activate_Rule		Rule_SREF
Activate_Rule		CE_CLMAX
Activate_Rule		Rule_VSTALL
Activate_Rule		Rule_VFLAP
Activate_Rule		Rule_VDIVE
Activate_Rule		CE_WFLAP_1
Activate_Rule		Rule_WLED
Activate_Rule		Rule_WHLDEV
Activate_Rule		Rule_SKSTR
Activate_Rule		CE_WWING_2
```

```
Activate_Rule           Rule_WWPSFT
Activate_Rule           Rule_WWING_2
ELSE       0
Control_Eqn             CE_WCFLAP_1         JFLTYP == 1
THEN       1
Activate_Rule           Rule_WCFLAP_1
ELSE       1
Fire_Rule  CE_WCFLAP_2
Control_Eqn             CE_WCFLAP_2         JFLTYP == 2
THEN       1
Activate_Rule           Rule_WCFLAP_2
ELSE       1
Fire_Rule  CE_WCFLAP_3
Control_Eqn             CE_WCFLAP_3         JFLTYP == 3
THEN       1
Activate_Rule           Rule_WCFLAP_3
ELSE       1
Fire_Rule  CE_WCFLAP_4
Control_Eqn             CE_WCFLAP_4         JFLTYP == 4
THEN       1
Activate_Rule           Rule_WCFLAP_4
ELSE       1
Fire_Rule  CE_WCFLAP_5
Control_Eqn             CE_WCFLAP_5         JFLTYP == 5
THEN       1
Activate_Rule           Rule_WCFLAP_5
ELSE       1
Fire_Rule  CE_WCFLAP_6
Control_Eqn             CE_WCFLAP_6         JFLTYP == 6
THEN       1
Activate_Rule           Rule_WCFLAP_6
ELSE       1
Fire_Rule  CE_WCFLAP_7
Control_Eqn             CE_WCFLAP_7         JFLTYP == 7
THEN       1
Activate_Rule           Rule_WCFLAP_7
ELSE       0
Control_Eqn             CE_WFLAP_1          JFLTYP == 1
THEN       1
Activate_Rule           Rule_WFLAP_1
ELSE       1
Fire_Rule  CE_WFLAP_2
Control_Eqn             CE_WFLAP_2          JFLTYP == 2
THEN       1
```

```
Activate_Rule           CE_WFLAP_2_1
ELSE        1
Fire_Rule   CE_WFLAP_3
Control_Eqn             CE_WFLAP_2_1        VFLAP <= 160
THEN        1
Activate_Rule           Rule_WFLAP_2
ELSE        1
Fire_Rule   Rule_WFLAP_3
Control_Eqn             CE_WFLAP_3          JFLTYP >= 3 && JFLTYP <= 5
THEN        1
Activate_Rule           Rule_WFLAP_4
ELSE        1
Fire_Rule   CE_WFLAP_4
Control_Eqn             CE_WFLAP_4          JFLTYP >= 6
THEN        1
Activate_Rule           Rule_WFLAP_5
ELSE        0
Control_Eqn             CE_WWING_2          ( ABS ( WWING - WWOLD ) /
WWING ) && ITER < 20
THEN        6
Activate_Rule           CE_FWL
Activate_Rule           Rule_FMAT
Activate_Rule           CE_FPROP_1
Activate_Rule           CE_FGEAR_1
Activate_Rule           Rule_WWOLD
Fire_Rule   Rule_ITER
Fire_Rule   Rule_CE_WWING_2
Else        0
Control_Eqn             CE_FWL  ITER == 1
THEN        1
Activate_Rule           Rule_FWL_1
ELSE        1
Activate_Rule           Rule_FWL_2
Control_Eqn             CE_FPROP_1          IMNTEN == 0
THEN        1
Activate_Rule           Rule_FPROP_1
ELSE        1
Fire_Rule   CE_FPROP_2
Control_Eqn             CE_FPROP_2          IMNTEN == 1 && EN <= 2
THEN        1
Activate_Rule           Rule_FPROP_2
ELSE        1
Fire_Rule   CE_FPROP_3
Control_Eqn             CE_FPROP_3          IMNTEN == 1 && EN > 2
```

```
THEN        1
Activate_Rule        Rule_FPROP_3
ELSE        0
Control_Eqn          CE_FGEAR_1          IMNTGR == 1
THEN        1
Activate_Rule        Rule_FGEAR_1
ELSE        1
Fire_Rule   CE_FGEAR_2
Control_Eqn          CE_FGEAR_2          IMNTGR == 0
THEN        1
Activate_Rule        Rule_FGEAR_2
ELSE        0
Control_Eqn          CE_WHT  TCRHT != 0 && IWHT == 0
THEN        7
Activate_Rule        Rule_EXPHT
Activate_Rule        Rule_XHTAIL
Activate_Rule        Rule_CRHT
Activate_Rule        Rule_BHTAIL
Activate_Rule        Rule_TRHT
Activate_Rule        Rule_FH
Activate_Rule        Rule_WHT_1
ELSE        1
Activate_Rule        Rule_WHT_2
Control_Eqn          CE_WVT  IWVT == 0
THEN        8
Activate_Rule        Rule_EXPVT
Activate_Rule        Rule_SPANVT
Activate_Rule        Rule_XVTAIL
Activate_Rule        Rule_CRVT
Activate_Rule        Rule_TRVT
Activate_Rule        Rule_FV
Activate_Rule        Rule_SAH
Activate_Rule        Rule_WVT
ELSE        0
Control_Eqn          CE_WCAND       SWETCN != 0 && IWCAND == 0
THEN        7
Activate_Rule        Rule_EXPCN
Activate_Rule        Rule_XCANRD
Activate_Rule        Rule_CRCNRD
Activate_Rule        Rule_SPANCN
Activate_Rule        Rule_TRCNRD
Activate_Rule        Rule_FCNRD
Activate_Rule        Rule_WCAND_1
ELSE        1
```

```
Activate_Rule          Rule_WCAND_2
Control_Eqn            CE_WBODY      IWBODY == 0
THEN      3
Activate_Rule          CE_WX
Activate_Rule          CE_PYLLNG
Activate_Rule          Rule_WBODY
ELSE      0
Control_Eqn            CE_WX   IMNTEN == 1
THEN      1
Activate_Rule          Rule_WX_2
ELSE      1
Activate_Rule          Rule_WX_1
Control_Eqn            CE_WPIV_1      IOBLIQ >= 1
THEN      1
Activate_Rule          CE_WPIV_2
ELSE      0
Control_Eqn            CE_WPIV_2      IWPIV == 0
THEN      1
Activate_Rule          Rule_WPIV_1
ELSE      0
Control_Eqn            CE_APU  PASS > 35
THEN      1
Activate_Rule          Rule_WAPU
ELSE      0
Control_Eqn            CE_WELT_1           WGTO >= 3000
THEN      1
Activate_Rule          Rule_WELT_1
ELSE      1
Fire_Rule   CE_WELT_2
Control_Eqn            CE_WELT_2           WGTO >= 5500
THEN      1
Activate_Rule          Rule_WELT_2
ELSE      1
Fire_Rule   CE_WELT_3
Control_Eqn            CE_WELT_3           WGTO >= 7500
THEN      1
Activate_Rule          Rule_WELT_3
ELSE      1
Fire_Rule   CE_WELT_4
Control_Eqn            CE_WELT_4           WGTO >= 11000
THEN      1
Activate_Rule          Rule_WELT_4
ELSE      1
Fire_Rule   CE_WELT_5
```

```
Control_Eqn         CE_WELT_5           PASS >= 20 && PASS < 30
THEN     1
Activate_Rule       Rule_WELT_5
ELSE     1
Fire_Rule   CE_WELT_6
Control_Eqn         CE_WELT_6           PASS >= 30 && PASS <= 50
THEN     1
Activate_Rule       Rule_WELT_6
ELSE     1
Fire_Rule   CE_WELT_7
Control_Eqn         CE_WELT_7           PASS > 50
THEN     1
Activate_Rule       Rule_WELT_7
ELSE     1
Fire_Rule   CE_WELT_8
Control_Eqn         CE_WELT_8           PASS > 100
THEN     1
Activate_Rule       Rule_WELT_8
ELSE     0
Control_Eqn         CE_WAIRC            WGTO > 3500
THEN     1
Activate_Rule       Rule_WAIRC_2
ELSE     1
Activate_Rule       Rule_WAIRC_1
Control_Eqn         CE_WGEAR_1          WGTO > 3000 && PASS < 9
THEN     1
Activate_Rule       Rule_WGEAR_1
ELSE     1
Fire_Rule   CE_WGEAR_2
Control_Eqn         CE_WGEAR_2          PASS >= 9 && PASS <= 19
THEN     1
Activate_Rule       Rule_WGEAR_2
ELSE     1
Fire_Rule   CE_WGEAR_3
Control_Eqn         CE_WGEAR_3          PASS > 19 && PASS <= 74
THEN     1
Activate_Rule       Rule_WGEAR_3
ELSE     1
Fire_Rule   CE_WGEAR_4
Control_Eqn         CE_WGEAR_4          PASS > 74
THEN     1
Activate_Rule       Rule_WGEAR_4
ELSE     0
Control_Eqn         CE_WFFURN_1         PASS > 80 && WGTO >= 10000
```

```
THEN        1
Activate_Rule           Rule_WFFURN_1
ELSE        1
Fire_Rule   CE_WFFURN_2
Control_Eqn             CE_WFFURN_2         WGTO < 10000
THEN        1
Activate_Rule           Rule_WFFURN_2
ELSE        1
Activate_Rule           Rule_WFFURN_3
Control_Eqn             CE_WCREW            IWCREW == 0
THEN        1
Activate_Rule           Rule_WCREW
ELSE        0
Control_Eqn             CE_WCABG_1          PASS < 20
THEN        1
Activate_Rule           Rule_WCABG_1
ELSE        1
Fire_Rule   CE_WCABG_2
Control_Eqn             CE_WCABG_2          PASS >= 20 && PASS < 40
THEN        1
Activate_Rule           Rule_WCABG_2
ELSE        1
Fire_Rule   CE_WCABG_3
Control_Eqn             CE_WCABG_3          PASS >= 40
THEN        1
Activate_Rule           Rule_WCABG_3
ELSE        0
Control_Eqn             CE_WPASSV WCPPAS == 0
THEN        4
Activtate_Rule  CE_WH2O
Activate_Rule           CE_WGALLY
Activate_Rule           CE_WFEMER_1
Activate_Rule           Rule_WPASSV_1
ELSE        1
Activate_Rule           Rule_WPASSV_2
Control_Eqn             CE_WGALLY           PASS > 19
THEN        1
Activate_Rule           Rule_WGALLY
ELSE        0
Control_Eqn             CE_WH20 PASS > 19
THEN        1
Activate_Rule           Rule_WH2O_1
ELSE        1
Activate_Rule           Rule_WH2O_2
```

```
Control_Eqn		CE_FEMER_1		PASS > 5 && PASS <= 9
THEN	1
Activate_Rule		Rule_WFEMER_1
ELSE	1
Fire_Rule	CE_WFEMER_2
Control_Eqn		CE_FEMER_2	PASS > 9 && PASS < 35
THEN	1
Activate_Rule		Rule_WFEMER_2
ELSE	1
Fire_Rule	CE_WFEMER_3
Control_Eqn		CE_FEMER_3	PASS >= 35
THEN	1
Activate_Rule		Rule_WFEMER_3
ELSE	0
Control_Eqn		CE_WPASS	IWPASS == 0
THEN	1
Activate_Rule		Rule_WPASS
ELSE	0
Control_Eqn		CE_WBAG IWBAG == 0
THEN	1
Activate_Rule		Rule_WBAG
ELSE	0
Control_Eqn		CE_WPL	IWPL == 0
THEN	1
Activate_Rule		Rule_WPASS
ELSE	0
Control_Eqn		CE_WFEQ IWFEQ == 0
THEN	1
Activate_Rule		Rule_WFEQ
ELSE	0
Control_Eqn		CE_WAF	IWAF == 0
THEN	1
Activate_Rule		Rule_WAF
ELSE	0
Control_Eqn		CE_WFR	ITYPE != 3
THEN	1
Activate_Rule		Rule_WFR
ELSE	0
```

Geometry.kb

Parameter	ARCN	float	5.0	1
Parameter	ARHT	float	5.0	1
Parameter	ARVT	float	5.0	1
Parameter	ARWG	float	5	1
Parameter	BDMAX	float	0.0	1
Parameter	BHTAIL	float	0.0	0
Parameter	BODL	float	0.0	1
Parameter	BS	float	0.0	0
Parameter	CB	float	0.0	0
Parameter	CLD	float	0.0	0
Parameter	CRCNRD	float	0.0	0
Parameter	CRHT	float	0.0	0
Parameter	CROOT	float	0.0	0
Parameter	CRVT	float	0.0	0
Parameter	EN	float	2.0	1
Parameter	EXPCN	float	0.0	0
Parameter	EXPHT	float	0.0	0
Parameter	EXPVT	float	0.0	0
Parameter	FFUR	float	0.0	0
Parameter	FGEAR	float	1.0	1
Parameter	FLAPN	float	1.0	1
Parameter	FMAT	float	1.0	1
Parameter	FPROP	float	.95 1	
Parameter	FWING	float	133.41	
Parameter	IMNTEN	float	1.0	1
Parameter	IMNTGR	float	0.0	1
Parameter	IOBLIQ	float	0.0	1
Parameter	ITAIL	float	0.0	1
Parameter	JFLTYP	float	3.0	1
Parameter	NEN	float	2.0	0
Parameter	NLAV	float	0.0	0
Parameter	PYLLEN	float	.20	1
Parameter	PYLLNG	float	0.0	0
Parameter	ROOTCN	float	0.0	1
Parameter	ROOTHT	float	0.0	1
Parameter	ROOTVT	float	0.0	1
Parameter	SAH	float	0.0	0
Parameter	SCN	float	0.0	1
Parameter	SFLAP	float	36.85	1
Parameter	SLE	float	0.0	1
Parameter	SPAN	float	0.0	0

```
Parameter             SPANCN  float         0.0         0
Parameter             SPANHT  float         0.0         1
Parameter             SPANVT  float         0.0         0
Parameter             SREF     float         0.0         0
Parameter             STRUT    float         0.0         1
Parameter             SVT      float         0.0         1
Parameter             SWAFEX  float         0.0         1
Parameter             SWETB    float         0.0         1
Parameter             SWETCN  float         0.0         1
Parameter             SWETHT  float         0.0         1
Parameter             SWETWG  float         0.0         1
Parameter             SWG      float         0.0         1
Parameter             SWPCN    float         0.0         1
Parameter             SWPCNR  float         0.0         0
Parameter             SWPHT    float         0.0         1
Parameter             SWPHTR  float         0.0         0
Parameter             SWPVT    float         0.0         1
Parameter             SWPVTR  float         0.0         0
Parameter             SWPWG    float         0.0         1
Parameter             SWPWGR  float         0.0         0
Parameter             SWSTR    float         0.0         1
Parameter             TCEFF    float         0.0         0
Parameter             TCRCN    float         0.1         1
Parameter             TCRHT    float         0.1         1
Parameter             TCRVT    float   .10   1
Parameter   TCRWG    float      0.0        1
Parameter   TCTWG    float      0.0        1
Parameter   TRCNRD  float      .99       0
Parameter   TRHT     float      .99       0
Parameter   TRVT     float      .99       0
Parameter   TRWG     float      .99       1
Parameter   VTNO     float      1.0        1
Parameter   XCANRD  float      0.0        0
Parameter   XHTAIL  float      1.0        0
Parameter   XQCBCN  float      0.0        1
Parameter   XQCBHT  float      0.0        1
Parameter   XQCBVT  float      0.0        1
Parameter   XQCBWG  float      0.0        1
Parameter   XVTAIL  float      1.0        0
Eqn_Rule    Rule_SWPWGR      1
2           SWPWGR = PI * SWPWG / 180.
Action  0
Eqn_Rule    Rule_SWPHTR      1
2           SWPHTR = PI * SWPHT / 180.
```

```
Action  0
Eqn_Rule   Rule_SWPVTR          1
2          SWPVTR = PI * SWPVT / 180.
Action  0
Eqn_Rule   Rule_SWPCNR          1
2          SWPCNR = PI * SWPCN / 180.
Action  0
Eqn_Rule   Rule_SPAN            1
2          SPAN = sqrt ( SWG * ARWG )
Action  0
Eqn_Rule   Rule_CROOT           1
3          CROOT = 2. * SWG / ( SPAN * ( 1. + TRWG ) )
Action  0
Eqn_Rule   Rule_CB 1
4          CB = CROOT * (        1. + ( TRWG - 1. ) * BDMAX / SPAN )
Action  0
Eqn_Rule   Rule_BS 1
2          BS = 0.5 * SPAN / cos ( SWPWGR )
Action  0
Eqn_Rule   Rule_CLD  1
2          CLD = BODL / BDMAX
Action  0
Eqn_Rule   Rule_TCEFF           1
2          TCEFF = ( 4. * TCRWG + TCTWG ) / 5.
Action  0
Eqn_Rule   Rule_NEN  1
1          NEN = EN
Action  0
Eqn_Rule   Rule_SREF1
1          SREF = SWG
Action  0
Eqn_Rule   Rule_FMAT            1
2          FMAT =   1. + 2.5 * sqrt ( cos ( SWPWGR ) / SPAN )
Action  0
Eqn_Rule   Rule_FPROP_1                      1
0          FPROP = 1.05
Action  0
Eqn_Rule   Rule_FPROP_2                      1
0          FPROP = 0.95
Action  0
Eqn_Rule   Rule_FPROP_3                      1
0          FPROP = 0.90
Action  0
Eqn_Rule   Rule_FGEAR_1                      1
```

```
0           FGEAR = 1
Action  0
Eqn_Rule   Rule_FGEAR_2                      1
0           FGEAR = 0.95
Action  0
Eqn_Rule   Rule_EXPHT                        1
1           EXPHT = 0.5 * SWETHT
Action  0
Eqn_Rule   Rule_XHTAIL                       1
2           XHTAIL = XQCBHT - XQCBWG
Action  0
Eqn_Rule   Rule_CRHT                         1
1           CRHT = ROOTHT
Action  0
Eqn_Rule Rule_BHTAIL    1
1           BHTAIL = SPANHT
Action  0
Eqn_Rule   Rule_TRHT                         1
3           TRHT =   1. + 2. * EXPHT / ( ROOTHT * sqrt ( ARHT * EXPHT ) )
Action  0
Eqn_Rule   Rule_EXPVT                        1
1           EXPVT = SVT
Action  0
Eqn_Rule   Rule_SPANVT                       1
2           SPANVT = sqrt ( ARVT * SVT )
Action  0
Eqn_Rule   Rule_XVTAIL                       1
2           XVTAIL = XQCBVT - XQCBWG
Action  0
Eqn_Rule   Rule_CRVT                         1
1           CRVT = ROOTVT
Action  0
Eqn_Rule   Rule_TRVT                         1
3           TRVT =   1. + 2. * EXPVT / ( ROOTVT * sqrt ( ARVT * EXPVT ) )
Action  0
Eqn_Rule   Rule_SAH  1
1           SAH = ITAIL
Action  0
Eqn_Rule   Rule_EXPCN                        1
1           EXPCN = 0.5 * SWETCN
Action  0
Eqn_Rule Rule_XCANRD    1
2           XCANRD = XQCBCN - XQCBWG
Action  0
```

```
Eqn_Rule Rule_CRCNRD    1
1           CRCNRD = ROOTCN
Action  0
Eqn_Rule Rule_SPANCN    1
2           SPANCN = sqrt ( ARCN , SCN )
Action  0
Eqn_Rule Rule_TRCNRD    1
3           TRCNRD = 1. + 2. * EXPCN / ( ROOTCN * sqrt ( ARCN * EXPCN ) )
Action  0
Eqn_Rule   Rule_PYLLNG              1
2           PYLLNG = PYLLEN * ENGLEN
Action  0
Eqn_Rule   Rule_FFUR_1              1
0           FFUR = 28.
Action  0
Eqn_Rule   Rule_FFUR_2              1
1           FFUR = 28. + 10.516 * ( BDMAX - 5.667 )
Action  0
Eqn_Rule   Rule_NLAV_1              1
0           NLAV = 0
Action  0
Eqn_Rule   Rule_NLAV_2              1
0           NLAV = 1
Action  0
Eqn_Rule   Rule_NLAV_3              1
0           NLAV = 2
Action  0
Eqn_Rule   Rule_NLAV_4              1
0           NLAV = 3
Action  0
Eqn_Rule   Rule_NLAV_5              1
0           NLAV = 4
Action  0
Eqn_Rule   Rule_NLAV_6              1
0           NLAV = 5
Action  0
Eqn_Rule   Rule_NLAV_7              1
0           NLAV = 6
Action  0
Control_Eqn          CE_PYLLEN          IMNTEN == 1
THEN       1
Activate_Rule        Rule_PYLLNG
ELSE       0
Control_Eqn CE_FFUR  BDMAX <= 5.667
```

```
THEN 1
Activate_Rule Rule_FFUR_1
ELSE 1
Activate_Rule Rule_FFUR_2
Control_Eqn          CE_NLAV_1          PASS <= 25
THEN      1
Activate_Rule        Rule_NLAV_1
ELSE      1
Fire_Rule  CE_NLAV_2
Control_Eqn          CE_NLAV_2          PASS > 25 && PASS < 51
THEN      1
Activate_Rule        Rule_NLAV_2
ELSE      1
Fire_Rule  CE_NLAV_3
Control_Eqn          CE_NLAV_3          PASS >= 51 && PASS < 101
THEN      1
Activate_Rule        Rule_NLAV_3
ELSE      1
Fire_Rule  CE_NLAV_4
Control_Eqn          CE_NLAV_4          PASS >= 101 && PASS < 151
THEN      1
Activate_Rule        Rule_NLAV_4
ELSE      1
Fire_Rule  CE_NLAV_5
Control_Eqn          CE_NLAV_5          PASS >= 151 && PASS < 201
THEN      1
Activate_Rule        Rule_NLAV_5
ELSE      1
Fire_Rule  CE_NLAV_6
Control_Eqn          CE_NLAV_6          PASS >= 201 && PASS < 251
THEN      1
Activate_Rule        Rule_NLAV_6
ELSE      1
Fire_Rule  CE_NLAV_7
Control_Eqn          CE_NLAV_7          PASS >= 251
THEN      1
Activate_Rule        Rule_NLAV_7
ELSE      0
```

General.kb

```
Parameter   COACH    float    0.0        1
Parameter   CREW     float    0.0        1
Parameter   FIRST    float    0.0        0
Parameter   ITYPE    float    0.0        0
Parameter   PASS     float    0.0        1
Parameter   STEWCC   float    0.0        1
Parameter   STEWFC   float    0.0        1
Parameter   TYPE     string   "GENAVI"   0
Eqn_Rule    Rule_FIRST        2
2           FIRST = PASS - COACH
1           FIRST = 100
ACTION  0
```

Propulsion.kb

Parameter	AENDIA	float	0.0	1
Parameter	AENLE	float	0.0	0
Parameter	ENGLEN	float	0.0	1

Trajectory.kb

```
Parameter  CLMAX    float      0.0 0
Parameter  CLMXTO   float      0.0 1
Parameter  MACH     float      0.0 0
Parameter  RHOS     float      0.0 0
Parameter  VDIVE    float      0.0 0
Parameter  VE       float      0.0 0
Parameter  VFLAP    float      0.0 0
Parameter  VSTALL   float      0.0 0
Eqn_Rule   Rule_MACH           1
1          MACH = sqrt ( QMAX / 1482.5 )
Action  0
Eqn_Rule   Rule_VE 1
1          VE = MACH * 661.7
Action  0
Eqn_Rule   Rule_CLMAX_1        1
1          CLMAX = CLMXTO
Action  0
Eqn_Rule   Rule_CLMAX_2        1
0          CLMAX = 1
Action  0
Eqn_Rule   Rule_VSTALL         1
4          VSTALL = 0.5921 * sqrt ( 2. * WGTO / ( RHOS * SREF * CLMAX ) )
Action  0
Eqn_Rule   Rule_VFLAP          1
1          VFLAP = 1.8 * VSTALL
Action  0
Eqn_Rule   Rule_VDIVE          1
2          VDIVE = sqrt (       2. * QMAX /
RHOS )
Action  0
Control_Eqn          CE_CLMAX
CLMXTO < 0.1
THEN       1
Activate_Rule        Rule_CLMAX_2
ELSE       0
```

Appendix C - Detailed Class Descriptions

This appendix lists those classes that were developed as part of this thesis. Also, modifications that were made to existing classes are presented. It is assumed that the reader has a knowledge of C++ and OSF/Motif. For a full description of those classes that are only shown in part, refer to [Nara93].

Rule Class

This class is the base rule class for all rule classes.

Class inheritance

None

Private Variables

int active Flag for determining whether a rule is active or not active.

Public Variables

Rule *next Pointer to the next rule contained in the link list.

Private Functions

None

Public Functions

int Get_Status()

Function Description

Returns the active state of a variable

Argument Description

None.

Eqn_Rule Class

This class is used to store knowledge in the form of mathematical equations.

Class Inheritance

Rule Class

Private Variables

char *name Stores the name of the rule.

EQN *eqn_listLink list of EQN structure to store multiple equations in one rule

Action *action_list Link list of Action structures to store actions to be performed when a rule is fired.

Public Variables
None

Private Functions
None

Public Functions
Eqn_Rule(char *rule_name, EQN *eqns, Action *actions)
Function Description
Constructor for the Eqn_Rule class
Argument Description
rule_name Pointer to the name of the rule
eqns Pointer to the equations link list
actions Pointer to the actions link list

Control_Eqn Class

This class is used to store rules that determine which equations should be set active during a design session for a given set of conditions.

Class Inheritance
Rule Class

Private Variables
char *name Pointer to the name of the rule
char *eqn Pointer to a Boolean Equation that evaluates a given condition
Action *then_condition Link list of actions to be performed if the equation is true
Action *else_condition Link list of actions to be performed if the equation is false

Public Variables
None

Private Functions
None

Public Functions
Control_Eqn(char *rule_name,char *eqn_string, Action *then_cond, Action *else_cond)
Function Description
Constructor for the Control_Eqn Class
Argument Description
rule_name Pointer to the name of the rule

eqn_string Pointer to the Boolean equation
then_cond Pointer to the link list of actions for true
else_cond Pointer to the link list of actions for false

Parameter Class

This class represents a design parameter used in the design. It can be of 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 The current value of the parameter if the parameter is of type string
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 Float 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 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 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

Consult Class - Modifications

This class is used to represent the Consult operation mode of the ET.

Class Inheritance

Consult_Popup

Private Variables

None

Public Variables

None

Private Functions

None

Public Functions

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

Consult_Popup Class - Modifications

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

Class Inheritance

None

Private Variables

int no_changeable Number of changeable parameters

char ** p_list List of the changeable parameters

int no_of_changeable_panes Number of panes needed to display changeable parameter list

Public Variables

None

Private Functions

void Get_Changeable_List()

Function Description

Gets the list of changeable parameters

Argument Description

None

int Create_Changeable_Pulldown()

Function Description

Creates the the pulldown menus for the changeable parameters

Argument Description

None

Public Functions

void Create_Consult_Popup_Pulldown()

Function Description

Creates the consult pulldown menu

Argument Description
None

Protected variables

Widget Prompt_user	Widget for the Prompt User window
float entered_number	Value entered by the user during Propmt Value function
float New_Value	Value entered by the user to change the value of a parameter or the value changed as a result of a percentage increase or decrease
float percentage	Percentage value enetered by the user
effect_callback_struct *params	Pointer to a struct to send needed parameters to the callback function of the effect buttons

ET class - Modifications

This class represents an expert in the real world in a particular domain.

Class Inheritance
None

Private Variables

int No_of_Changeable   Nunber of changeable parameters owned by the ET

Public Variables
None

Private Functions
void Read_KB()
Function Description
Reads the knowledge file into the expert system
Argument Description
None

Public Fuctions
ET(char*etname,char*mode)
Function Description
Constuctor for the ET class

Argument Description

etname	Pointer to the name of the expert
mode	Pointer to the initial mode the ET operates in

int Get_No_Changeable()
Function Description
Returns the no of changeable parameters owned by the ET

Argument Description

None

void Put_Value(Parameter *parametr, float value)

Function Description

Gives a parameter a new value

Argument Description

parametr Pointer to the name of the parameter

value Valau to be put into a parameter

float Get_Min_Value(Parameter *parametr)

Function Description

Returns the minimum allowed value for a parameter

Argument Description

parametr Pointer to the name of a parameter

float Get_Max_Value((Parameter *parametr)

Function Description

Returns the maximum allowed value for a parameter

Argument Description

paramtr Pointer to the name of a parameter

void Add_Rule(char*CE_Name,char*eqn,
Action*then_condtion, Action *else_condtion)

Function Description

Adds a new Control Equation Rule to the link list of rules

Argument Description

CE_Name Pointer to the name of the rule

eqn Pointer to the Boolean equation

then_condition Pointer to the true result link list of actions

else_condtion Pointer to the false result link list of actions

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

Function Description

Creates a parameter object and adds it to the linked list

Argument Description

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

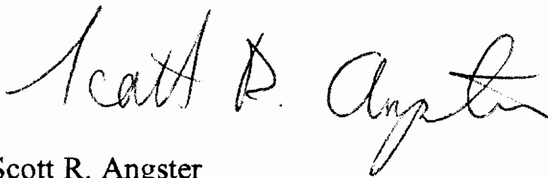
0 - not user changeable

min_value Minimum value of the parameter

max_value Maximum value of the parameter

Vita

Scott R. Angster was born on February 17, 1970. He grew up on the coast of New Jersey in Wall Township. He attended Wall High School, where his interest in mathematics and science led him to the field of engineering. While in high school, he met Elizabeth Kate Hendrickson, whom he married after graduating from Virginia Tech in Mechanical Engineering. As an undergraduate, Scott was the captain for the Virginia Tech Solaray solar car team. This experience motivated him to further his studies in mechanical engineering at Virginia Tech. Scott will be attending Washington State University to pursue his Doctorate degree, also in mechanical engineering.



Scott R. Angster