AEK0673

A RULE BASED EXPERT SYSTEM WHICH CONFIGURES GAS CHROMATOGRAPHS

By

GEORGE ERIC WOLKE

Bachelor of Science

Syracuse University

Syracuse, New York

1984

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1997

A RULE BASED EXPERT SYSTEM WHICH CONFIGURES GAS CHROMATOGRAPHS

Thesis Approved:

Blayne E. Mayfield

Thesis Advisor

Jacques E. LaFrance

H. Lu

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGEMENTS

This paper is the result of much sweat and tears. It seems as though I've been working on it forever, and not just for eight months. While the paper's title suggests that it is simply an expert systems project, to me it is much more than that. It is a 'cradle to grave' software engineering project. As you read this paper you will see discussion of the software development process, software modeling, object oriented development, relational databases, and rule based expert systems. Indeed, most of my effort was expended making all the pieces of this complex puzzle fit together. I think it does all fit and I sincerely hope that, after reading the paper, you will too.

I wish to offer my sincere appreciation to my major advisor, Dr. Jacques LaFrance. His devotion to the teaching of Computer Science served to "keep me working" even when all I wanted to do was to sit and watch Syracuse basketball. In addition, I my appreciation extends to my other committee members, Dr. Blayne Mayfield and Dr. Huizhu Lu for taking the time to work with me throughout this process.

Finally, I would like to give special appreciation to my wife, Debbie, for her loving encouragement and understanding throughout this whole process. I can certainly say that she was the single most important reason this project was completed successfully. Thanks Deb and I love you.

ACKNOWLEDGEMENTS

This paper is the result of much sweat and tears. It seems as though I've been working on it forever, and not just for eight months. While the paper's title suggests that it is simply an expert systems project, to me it is much more than that. It is a 'cradle to grave' software engineering project. As you read this paper you will see discussion of the software development process, software modeling, object oriented development, relational databases, and rule based expert systems. Indeed, most of my effort was expended making all the pieces of this complex puzzle fit together. I think it does all fit and I sincerely hope that, after reading the paper, you will too.

I wish to offer my sincere appreciation to my major advisor, Dr. Jacques LaFrance. His devotion to the teaching of Computer Science served to "keep me working" even when all I wanted to do was to sit and watch Syracuse basketball. In addition, I my appreciation extends to my other committee members, Dr. Blayne Mayfield and Dr. Huizhu Lu for taking the time to work with me throughout this process.

Finally, I would like to give special appreciation to my wife, Debbie, for her loving encouragement and understanding throughout this whole process. I can certainly say that she was the single most important reason this project was completed successfully. Thanks Deb and I love you.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

This paper discusses the application of expert system technology to the configuration of process gas chromatographs (GCs). A GC is a tool for performing quantitative analysis of liquid and gaseous mixtures by separating their constituents into individual compounds. The analysis is necessary to ensure that the chemical plant is operating correctly and that the customer's product is meeting required levels of purity and consistency. The instrument can measure chemical concentrations down to parts per billion.

The customer specifies the application in the form of a "process stream". The process stream describes the type of chemicals that are present and the concentration range be analyzed. GC applications are typically, but not limited to, analysis of environmental samples and hydrocarbons. Different combinations of hydrocarbons require different combinations of hardware in order to be properly analyzed. There are an enormous number of variables that need to be considered when configuring a GC system. This makes the configuration problem a difficult one.

The GC consists of a core set of electronics that are part of every system plus custom hardware that is chosen based on the application the GC will work on. The custom hardware includes:

- Detectors: The transducer that provides a measure of the concentration of a chemical in the process stream.
- Columns: Allow the chemicals to separate into "bands". This allows the chemicals to elute from the column to the detector at different times. This chemical separation is what allows the chromatograph to measure the concentration of the component parts of the stream.
- Ovens: Allow the chemical stream to be elevated in temperature. This is important in order for the columns and detectors to work efficiently.
- Valves: Used to route the chemical stream between the detectors and columns.

A typical process GC contains one oven, two detectors, two valves and three columns.

The configuration task requires a deep understanding of chemistry and chromatography and is performed by chemists. In order to configure the GC a chemist will:

- Search an "application data base" consisting of historical data of applications developed by the GC manufacturer. The database used for this project is a relational database housed within Microsoft Access.
- If the search finds a perfect match to the customers needs, the matched configuration is chosen for this application. Often, no match is found.
- Typically, the search will return a set of partial matches. These are configurations that match a part of the current application. The chemist then uses their knowledge of chemistry and chromatography to chose a configuration from a combination of these possible choices.

The amount of time necessary to configure a GC ranges from hours to days. As a result, it takes longer to respond to customer requests for quotation. This can cause the customer to look elsewhere for a solution to their problem.

As part of a emphasis towards increasing efficiency, GC manufacturers are interested in ways to reduce the time and cost of configuring a GC. The ultimate goal is for configurations to be created by a sales engineer who can then use this information to create a quotation for the customer. This would allow the sales engineer to provide faster response to customer request and lead to increased sales. This paper describes the system that was developed to meet this objective.

Section 2 deals with production systems, conflict resolution, and reasoning strategies. Then, two systems that have bearing on this work are introduced. They are the Dendral[5] and XCON[13] systems. Finally, the Business Object Notation (BON) method of Object-oriented software design is discussed. Since this method was used in my work, it is important that the terminology of BON be introduced. The BON method is well suited to the design environment I used for this project (ISE Eiffel V3).

Section 3 then describes the project in further detail. Section 4 describes the result of my work. Section 5 presents conclusions drawn from my experiences gained through development of the system.

2.0 LITERATURE REVIEW

2.1 Rule-based Expert System Fundamentals

Rule-based expert systems are computer programs that solve domain specific problems. They perform tasks that are normally performed by domain experts. Therefore, this technology is a branch of the area of Artificial Intelligence. Rule-based expert systems have been used to solve problems ranging from the classification of chemical compounds, as in the Dendral Project [5], to the configuration of computer systems, as in the XCON Project [13], to the control of airport traffic [17].

While the three tasks mentioned above differ with respect to their area of application, they are similar in regard to the fact that they require domain specific information to perform their tasks. In the case of Dendral, the domain was organic chemistry. In XCON, the domain was computer system configuration. And in the final case the domain requires detailed understanding of airport logistics.

In each case the problem could not be solved using only the normal method of applying algorithms. The programs needed to make decisions and adapt to their environment. McDermott and Forgy [19] make the statement that expert systems must be able to deal with changes in their environment. As noted, this is indeed a characteristic of all three systems.

Buchanan and Smith [3] define the following characteristics of an expert system:

- They reason with domain specific knowledge that is symbolic as well as numeric. In rule based expert systems these symbolic representations are the domain specific rules of the system.
- They use domain specific knowledge that are heuristic as well as follow procedures that are numeric. Expert systems tend not to use first principles, but rather, their rules are based on human experience.

- They perform well in their problem area. However, they generally will not reason as well as their human counterparts. But they may perform better than the domain experts due to their systematic method of solving the problem.
- They explain, or make understandable, both what they know and the reasons for their answers.
- They retain flexibility and are adaptable to change.

These last two properties are necessary for the system to achieve a high level of performance, but are equally as important during the design and implementation process. Expert systems tend to be developed incrementally since not all of the domain knowledge is known when a project starts. Therefore the system must be flexible in order to allow for changes due to increased, or enhanced, knowledge throughout the product lifecycle.

Explanation is important in that it can provide skeptical users an assurance that the system is performing as promised and to aid users in understanding its results.

Generally, rule based expert systems utilize production systems (the rules and environment for making decisions). They employ a specific strategy to reason towards their solution. And they require a method of resolving conflict when more than one decision can be made. The following sections will examine these properties of expert systems.

2.2 Production Systems

Production systems have their roots in the rules of formal language theory and formal grammars. They were developed in order to represent knowledge in a general way, so that expert systems could be used to solve a wide variety of problems using a "standard engine". Therefore, they are a programming tool that has allowed AI systems to exist in more interesting environments [19].

Production systems have logical adequacy. They use a formalism that allows the use of variables to express knowledge. They have heuristic power. Along with well defined syntax and semantics, they can reason towards a solution to their problem. Finally, they have notational convenience. They express their knowledge in a way that is understandable [9]. Production systems consist of the productions themselves, and an environment in which to operate on the productions.

A production P(C1, C2, C3, ..., Cn -> A1, A2, A3, ...An) can be thought of as a condition-action pair. For instance:

```
IF
        the light is green
        and no cars are coming towards you
THEN
        cross the street.
```

If the conditions are met, then the actions described can take place. Another term used for the conditions of the pair is the "premise" of the production. The many productions that make up the domain specific knowledge of an expert system are referred to as the "rules" of the system. A typical system contains on the order of hundreds of rules.

The conditions of a rule are usually object-attribute-value triples [9]. For example:

(Peter age 36)

In this case the object called Peter has an attribute called age that has a value of 36. These triples can be used to create rules, such as:

```
IF
        (Peter age 36)  and (Peter employment none)
THEN
        (Peter claim unemployment-benefits).
```

The rule shown above is not very interesting in that is lacks generality. As mentioned above, production systems allow the use of variables to bind values to the conditions of a rule. With this in mind the rule can be rewritten as:

```
            IF
                    (*person age *number) and (*person employment none) and (*number > 15)
and                         (*number < 65)
            THEN
                    (*person claim unemployment)
```

The *person and *number variables of the rules help to bind "entities" to the rule. Notice that the *person variable comes up twice in the premise of the rule. An important principle of production systems is that the two instances of *person refer to the same entity. This property is termed binding an entity to a rule.

The entities of a production system are referred to as "facts". These are the data objects that cause the premise of rules to be made true. This results in the "firing", or "instantiation" of rules.

The rules of a production system are held in the system "production memory" or "knowledge base". The facts of the system are contained in "working memory". As facts cause rules to be instantiated, the contents of working memory change. This could cause the firing of additional rules, thus again changing the contents of working memory.

It is the task of the production system interpreter, or "inference engine," to allow the process of rule instantiation to proceed in an orderly and deterministic manner. The inference engine searches through the production memory looking for rules that can be made to fire with the current contents of working memory. This process has been termed the "Recognize - Act Cycle" by McDermott and Forgy [19]. The inference engine matches patterns from the knowledge base with the facts of working memory. Thus, this procedure has also been termed "pattern matching".

During the "Recognize - Act" cycle the interpreter first attempts to pattern match its knowledge base against the current contents of working memory. In the process of doing this, a set of possible rules, all of which may be instantiated at the current moment, is identified. This is called the "conflict set". The interpreter then used a conflict resolution mechanism to choose a rule from the conflict set to fire. This cycle continues until the system reached a conclusion or is stopped by the user.

Inference engines often match the conditions of the rule against facts, termed "forward chaining", or they can match the action of the rule against a known goal and then check to see if the facts support this hypothesis. This method is termed "backward chaining". Thus the inference engine chains together inferences to solve the problem.

Giarranto and Riley [8] note that chaining can be illustrated using the rule of modus ponens, as illustrated below:

IF P implies Q
AND P
THEN Q

For example, consider the following rules:

An elephant is a mammal
A mammal is an animal.

These rules can be used with forward chaining to form the following chain of inference:

If Dumbo is an Elephant
    and an Elephant is an mammal
        and a mammal is an animal
            Then Dumbo is an animal

Using backward chaining we would begin with the assertion that Dumbo is an animal and then see if the facts are supported by the hypothesis. We are given the fact that Dumbo is an elephant, thus we can apply the rules to prove the hypothesis.

Giarratano and Riley [8] suggest that it is helpful to think of forward chaining as a path through a problem space in which intermediate states are considered intermediate conclusions. In backward chaining these intermediate states represent intermediate hypotheses.

An important point must be made concerning the architecture of the production system. This is the clear separation of the knowledge base from the inference engine that acts upon the knowledge. Barr, Cohen, and Feigenbaum [3] state that it is vital for the inference engine to be separate from the production rules. This separation allows the inference engine to be independent of the problem domain that is currently being addressed. Therefore the inference engine can easily be reused to solve problems in other domains. Only in this way can a production system be constructed as a general tool that may be used to solve a variety of problems.

The process of obtaining knowledge from a domain expert is termed "knowledge acquisition". Several guidelines have be mentioned in order to allow a knowledge base to be developed in an effective manner. I would like to mention three of these guidelines that I feel are important to the AAI Expert System.

First, it important that the rules be expressed in a modular and declarative way. This allows systems to be developed in a scaled manner. The process of building an expert system has been equated to solving a problem by breaking up the problem into several independent subproblems. Each piece of the whole contains its own specific knowledge [3]. This allows the knowledge base to change in an orderly manner over time and to remain flexible to change.

Second, when representing knowledge the objects of the system should be named and as closely as possible match those used by the domain expert [3]. In this way, the expert can better understand the rules, aiding in the troubleshooting of the system. This is important because the process of knowledge acquisition is an iterative one involving the use of prototype systems to help solicit additional knowledge from the human expert [13].

Finally, when developing a knowledge base it is important to interview more than one expert, as many experts have a difficult time expressing their knowledge of the exceptional case. They tend to have a sparse but highly reliable picture of their domain [13]. Thus, interviewing multiple experts will lessen the chance that implicit assumptions concerning the domain are not excluded from the knowledge base [3].

Together, the three pieces of a modern production system are the production memory which contains domain specific knowledge (in the form of rules), the working memory which contains the current state of the system's environment (in the form of facts), and the inference engine which reasons about the current state of the system. The inference engine binds facts, or objects, to rules and thus causes the system to run (I believe this is why the interpreter has be called an inference engine since it causes the system to run or infer actions based on the state of the environment).

2.3 Reasoning Strategies

As mentioned in the previous section, the mode of chaining defines the way in which rules are fired. In this section, the way that a system is organized will be discussed. Brownston, Farrell, Kant, and Martin [4] state that most successful expert systems are organized around the method in which evidence is gathered. This method is referred to as the "reasoning strategy" of the system.

In this proposal I want to focus on two particular reasoning strategies that I feel are appropriate to this Expert System. In fact, a combination of both strategies were be applied to the system. These are "Plan - Generate - Test" and "Match". They both use pattern matching to solve their problem but approach their problems in very different ways.

In "Plan - Generate - Test" the system first uses its domain specific knowledge to limit the problem search space. This process, called planning, uses the knowledge base and inference engine to apply constraints on possible solutions [3]. For instance, in the Dendral system [5] the planner used input from the domain in the form of the rules of mass spectrometry to apply as constraints on the problem. These constraints become the contents of working memory and allow a chain of inference to form. The constraints allow a "filter" to be applied to the problem search space. The result is that only a subset of the possible solutions (those that are not "filtered" out) are passed onto the generator.

The output of the planner is then a subset of the program search space. The Generate process uses the production system to form further chains of inference and generates a possible solution to the problem.

The test phase checks the hypothesis to verify it has met the requirements of solution. If a solution is found then the process stops. If not, then the generate phase is entered again, but with a different set of facts in working memory. This causes further chains of inference to be applied and another hypothesis developed.

As can be seen, "Plan - Generate - Test" tends to reason toward a solution by creating inference chains, checking their validity, and then "back-tracking" to the generate phase if the result is not valid. On the other hand, the "Match" method applies rules of inference to generate a solution without back-tracking.

The "Match" technique is discussed by John McDermott [13] as a generalized form of pattern matching. Therefore, "Match" is analogous to "Plan - Generate - Test" in that it uses a search of the problem solution space, but in a manner that does not require backtracking. If there is at least one solution to a problem then the "Math" technique will find it without generating any false intermediate conclusions.

In "Match" the idea is to match facts against "forms". A form is a set of instantiations of rules that when taken together, form a successful intermediate solution to the problem. The technique requires that all conditions of the form be satisfied before it can move on. Thus, the chain of inference created guarantees a successful conclusion. This eliminates the need for back-tracking. When one form is considered valid, it generates a fact that causes the next logical form to be processed.

"Match" requires that forms be considered in isolation. One step must complete before another step can proceed. This may be accomplished using "context facts". These facts limit the scope of search to the form at hand. Since the rules contain conditions that use context facts, they will only become instantiated if the context fact is present in working memory. This prevents rules from firing out of context as doing so could lead to invalid chains of inference being formed which could invalidate the solution.

Two requirements therefore exist in order to utilize "Match". These are the Correspondence Condition, and the Propagation Condition [13]. The Correspondence Condition states that the elements of a form must be able to take on a locally determined value. That is, local to the process of completing a form. This is not to say that search space of the form is decomposed into a set of independent subtasks. As rules are fired, working memory will be changed and this will have an effect on the final solution. However, these changes in working memory will have no bearing on what has already been matched. This is the Propagation Condition which states that a partial ordering on decisions must exist such that the consequences of applying an operator can only effect the decisions that have not yet been made.

2.4 Conflict Resolution Strategies

As mentioned in Section 2.1.1, the inference engine is responsible for ensuring that rules are fired in an orderly and deterministic manner. This is simple if the contents of working memory are such that only one rule can fire. However, in practical systems conditions will exist in which multiple rules may fire, given the current contents of working memory. As was introduced earlier, the set of possible instantiations is called the conflict set. The interpreter must be sensitive to the conflict set and take the appropriate action. It must choose the appropriate rule from among the conflict set to instantiate if the system is to retain a high level of performance. McDermott and Forgy[19] state that production systems can reasonably be expected to solve their problems only if they utilize a carefully derived conflict resolution strategy. The ability to resolve conflict by choosing the appropriate rule to fire at any given time is therefore a fundamental property of a production system.

In the introduction to this chapter it was stated that expert systems must have the ability to learn, or to alter their knowledge and actions based on their environment. The contents of working memory change as rules are fired. This leads the system towards a solution by generating chains of inference. Equally as important is the ability to discern the properties of the working memory content, as this forms the basis for the principles of conflict resolution. The rules of conflict resolution will now be discussed.

McDermott and Forgy [19] discuss five sets of rules that form the basis of the conflict resolution strategies used in most modern production systems. They are:

- Production Order Rules. These rules place a priority on productions such that the highest priority production is chosen to fire first, followed by the next highest, and so on.
- Special Case Rules. These rules allow the most specific production to be fired first. For instance if one rule contains all of the conditions of another rule, plus additional constraints, it is considered to be the special case and is chosen to fire.
- Recency Rules. The inference engine will place time tags on the contents of working memory. Recency rules use the time tags to chose a rule to fire. Recency rules favor the newest facts in working memory as their presence follows the latest line of reasoning.
- Distinctiveness Rules. These rules chose a rule to fire based on if it has fired previously. If the rule has been fired, the facts causing the instantiation are deleted from working memory.
- Arbitrary Rules. These rules choose a rule at random from the set of rules waiting to fire.

None of these classifications of rules can perform conflict resolution by themselves. The possible strategies discussed used a combination of distinctiveness, recency, and special case rules to provide a comprehensive conflict resolution strategy. As mentioned in the beginning of the section, these rules provide the basis for the conflict resolution strategies used in modern production systems. For instance the OPS5 language, offers a strategy based on distinctiveness, special cases, and recency[4]. This is termed the LEX strategy. The MEA strategy of OPS5 is similar to LEX and will not be discussed in this paper.

The LEX strategy first uses a technique called "refraction" to eliminate all instantiations that have been previously fired from the conflict set. This is an example of the distinctiveness rules. Next, it uses recency to group the remaining elements of the conflict set according to their time tags. The time tag represents the amount of time an instantiation has been in working memory. The groups are considered in decreasing time order. The group with the largest recency number, is kept and the remaining elements are discarded from the conflict set. Finally, the remaining members of the conflict are sorted with regard to their "specificity" according to the special case rules.

2.5 The Dendral Project

The Dendral Project[5] was among the first expert systems. It was developed by Buchanan and Feigenbaum at Stanford University beginning in 1965. The application domain was the structure elucidation of organic compounds. It used knowledge of organic chemistry and mass spectrometry to solve its problem. The project was active until 1983 and provides invaluable insight into the process of developing an expert system.

The goal of the Dendral project as to produce "intelligent agents" to assist in the solution of problems within their program domain that required complex symbolic reasoning [5]. This involved both short term and long term goals. In the short term the system should useful to organic chemists working on the structure elucidation problem. In the long term, the system should provide a platform for further research into the study of expert systems. By all accounts, it succeeded in meeting all of its objectives.

As a bit of background, although I am by no means an organic chemist, may be helpful in understanding the discussion that follows. Dendral approached the elucidation problem by applying constraints from the principles of mass spectrometry. A mass spectrometer bombards a chemical compound with electrons, causing the compound to partly disintegrate into charged particles which are separated and collected by mass. These collections of compound fragments are referred to as spectra. The mass spectrum for a specific compound is often unique, however it is frequently impossible to infer the chemical structure from just the output of a mass spectrometer. Therefore, simply using a mass spectrometer is not sufficient to solve the elucidation problem.

Dendral, or Heuristic Dendral as it is sometimes referred, utilized the "Plan - Generate - Test" reasoning strategy. This approach was chosen since, while it was not the method used by the domain experts, the domain experts understood it. It also complemented the methods used by an organic chemist by supplying a highly reliable group of possible solutions for the chemist to work with.

The Dendral planner contained domain specific knowledge of mass spectrometry as a set of production rules. It used input from a mass spectrometer and applied its knowledge to rule out chemical structures that could not be represented by the input spectra. The output of the planner was then used by the Dendral generator program to produce a set of possible structures. The generator allowed the chemist to input information to the program to aid in the generation process. This input could come from any source, not just mass spectrometry. Thus the chemist was able to assist in the generation process. The test phase used further knowledge of mass spectrometry to rank the possible solutions.

This work had bearing on the expert system I developed in that it demonstrated that a potentially infinite search space can be narrowed down to a manageable level through the use of a "front end filter". The filter can use domain specific knowledge to remove possibilities that are not valid from the search space. This removed unnecessary burden from the inference process allowing it to focus only on potentially correct facts. It also is an example of how production systems can be used to implement expert systems. Lastly, it introduced the concept of an "intelligent assistant" by allowing the domain expert guide the generation process by providing input to the program.

2.6 The XCON Project

The XCON [13] project applied expert system technology to the configuration of VAX 11/780 computer system manufactured by Digital Equipment Company (DEC). It is among the first successful implementations of such a system in a commercial environment. XCON is the cornerstone of the expert knowledge system at DEC and has resulted in a net return to DEC in excess of $40M a year [1]. It also provided for the development of the OPS4 production language, the precursor to OPS5 and many current production systems. XCON was developed due to the complexity involved with configuring a VAX system. The number of variations of hardware configurations was so vast as to make the task difficult, time consuming and error prone. The input to the XCON program was an order for a system configuration. It produced a configuration in the form of drawing that could be used to configure the order for shipment.

XCON differs from DENDRAL in that it utilized "Match" as its resolution strategy. It contained knowledge in the form of production rules, and database entries. Constraint knowledge, or knowledge of how the configuration process worked was contained as productions. Component knowledge, or information on the actual hardware to be configured, was held in the database.

Constraint knowledge consisted of operator rules which helped the system move towards a solution, sequencing rules that determined the firing sequence of rules (these are context rules as discussed earlier), and information rules which were used to access the database. XCON proceeded through a predefined series of states, one having to complete before another could begin, to reach a conclusion. As such it demonstrated that the "Match" method could be successfully implemented.

Over time the scope of XCON expanded due to new product introductions, increases in functionality, and rule revisions. XCON now contains over 10,000 sequencing and operator rules in its knowledge base. The component database contains over 30000 records. XCON demonstrates that a large scale expert system can meet the demands of a large commercial enterprise.

Equally as important was the insights the project gives with respect to knowledge acquisition. McDermott [13] makes the following observations, among others:

- Experts tend to have a sparse but highly reliable picture of their domain.
- They tend to describe the configuration task in terms of subtasks.
- They have a deep understanding of how partial configurations effect future decisions.

The use of prototype systems at frequent points during the development process is necessary to extract knowledge of the exceptional case.

XCON has bearing on this Expert System due to its use of context rules to guide the inference process. The configuration process followed by our domain experts is similar to this approach and therefore I implemented a form of "Match " using context rules for the project. The guidelines for knowledge acquisition are also interesting, especially the use of prototype systems to help solicit implicit knowledge.

2.7 The BON Method of Software Design

The BON [18] method is an object-oriented design method that supports the concepts of "seamlessness", "reversibility", and "software contracting". It is implemented in a commercial software package available from Integrated Software Engineering (ISE), called EiffelCase.

In classical software models, such as the "Waterfall Model", software development proceeds through a series of well defined steps. First requirements are established, followed by the generation of a specification. Only then does implementation begin. However, software development is very often iterative, since requirements change over the course of a large project. The "Waterfall Model" requires that when a change is made to requirements, the process loops back to the specification step. This process of looping back to the beginning of the process when requirements change is unnatural. In fact, with BON, the line between design and implementation is grayed, lending itself to more natural software development.

Seamlessness refers to the ability to easily move from the analysis phase to the design phase and then to a running system. For example, EiffelCase will take the design and create a set of classes, written in Eiffel, that may be compiled and run directly.

Reversibility is the ability to iterate through the design and implementation phases easily. One can modify the Eiffel code and then bring these changes into the design model directly, without user intervention. This feature is also implemented in the EiffelCase product.

Software contracting refers to the method of placing pre- and post-conditions on the features of a class. Preconditions of a class feature are the conditions that the user of the feature must meet in order for the feature to operate correctly. Post-conditions are used to inform the user of how the feature will respond to his request given that the pre-condition has been met. This method is referred to by Meyer [16] as "design by contract". EiffelCase allows pre- and post-conditions to be specified in the design model. In addition to placing conditions on the features of a class, the method also allows class invarients to be specified. These are conditions that must be followed in order to use objects of the class properly.

The BON method includes two models, the "static" and the "dynamic" model. The "static" model shows how <u>classes</u> of the system are organized. This model shows the inheritance structure of the system and the various client-server relationships between classes. As such, the "static" model shows what the system is. It does not attempt to show how objects of the classes interact. This is the diagrammed using the "dynamic" model which shows how <u>objects</u> communicate and use the features of one another. It specified "how" the system operates through the generation of a set of interesting scenarios showing object interaction.

The BON method specifies a number of required deliverables. These are:

- Static Architecture - A set of diagrams showing the relationship between the classes of the system.
- Class Interfaces - A description of each class in the system showing the features of the class, and their contracts. The class interfaces are the heart of the BON method as they represent a detailed view of the classes and form the specification for these classes.
- Scenario Chart - A list of the scenarios that illustrate the dynamic operation of the system.
- Object Scenarios - A set of diagrams illustrating the scenarios listed on the scenario chart.

Each of these deliverables were produced during the development of the system. They will be presented later in this paper.

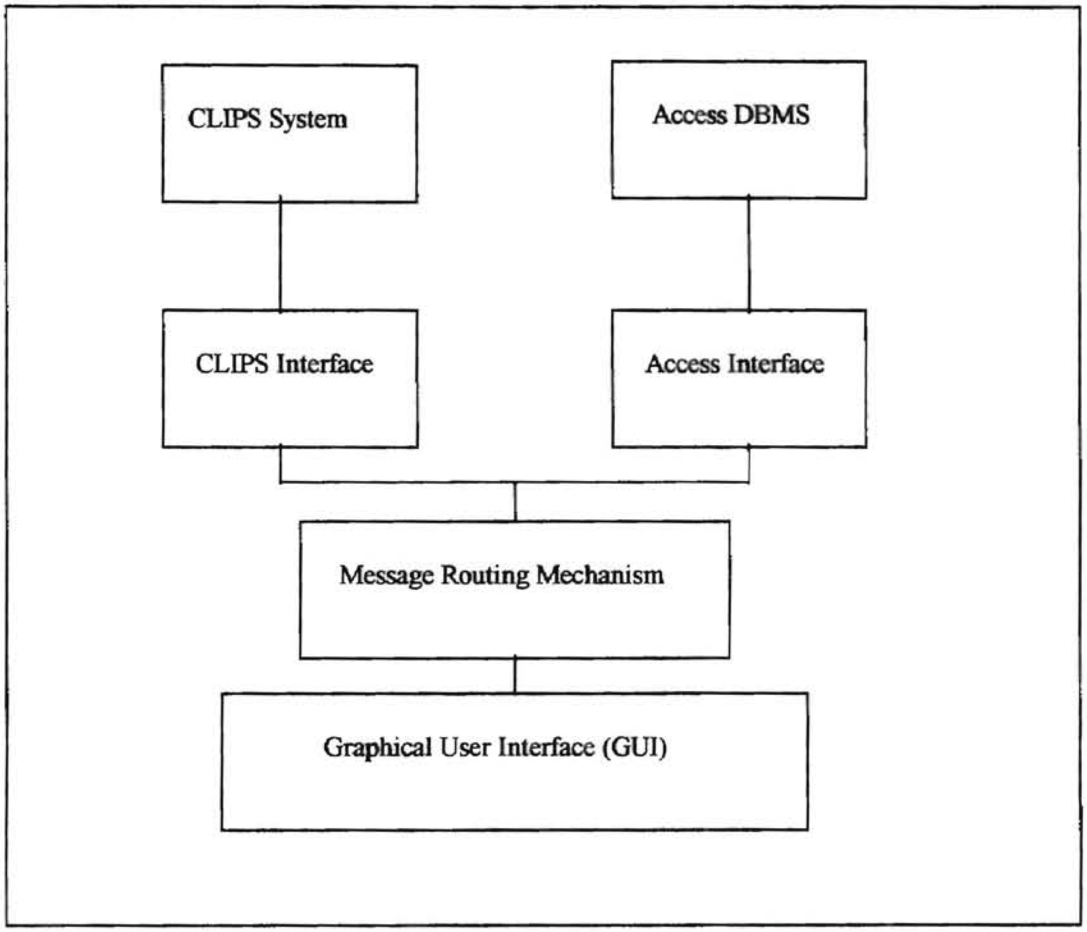
3.0 PROBLEM STATEMENT

The development of the GC Expert System was a software engineering project whose goal was to produce an embedded rule-based expert system. It was implemented using the following commercial software products: ISE Eiffel Version 3.3.9, ISE EiffelCase Version 3.3.9, Microsoft Access, and CLIPS (a production system developed by NASA for use on the Space Shuttle program). The target operating system is Windows 95. These tools were used as follows:

- ISE Eiffel - This is the software development environment for the system. It allows the production of 32-bit applications that can be executed on a target computer running Windows 95 or Windows NT. The Eiffel source code is compiled within this environment to C source code. Microsoft Visual C/C++, Version 2.0 is used to compile the C source code to object files and to link these into an executable application.
- ISE EiffelCase - This tool implements the BON method that was discussed earlier in this paper.
- CLIPS - This is a production system that contains the features discussed in the literature review. It is implemented as a 32-bit Windows Dynamic Link Library (DLL). The DLL is callable from the C language. Therefore, I created an Eiffel "wrapper" that interfaces between the rest of the system and the inference engine of CLIPS.
- Microsoft Access - This is a commercial DBMS that contains the application database that was introduced earlier in this paper. Eiffel can communicate to the DBMS using Open Database Connectivity (ODBC) protocol.

A block diagram showing the configuration of the system is shown on the following page.

- The GUI is the users view of the system. It allows the user to supply process stream information for a customers application. This data is passed to the message routing mechanism for further processing. The GUI will also shows the user the result of the reasoning session, including the configuration chosen to meet these requirements. If no solution to the problem was found, this is also be reported to the user.



- The message router allows information to be exchanged between the GUI, ACCESS and CLIPS. This piece of code provides a gateway to the other parts of the system.
- The CLIPS interface allows messages to be passes between CLIPS and the message router.
- The ACCESS interface allows the DBMS to be queried for data pertinent to this configuration session.

It is always important to specify what a computer program will do. It is equally as important to state what it won't do. Earlier in this paper I discussed the process an application chemist uses to configure a GC.

This is what the expert system does. It automates the existing process. If a solution can be reasoned using data taken from the application database, then the expert system will find this solution. However, if no solution can be reasoned from existing historical data, then the expert system will not be able to find a solution. This will be the case if the system is given a process stream, whose characteristics do not match any given in the database, as input. It is expected that this will happen. In the future, as the knowledge base matures and becomes more sophisticated, this constraint may be lifted. But for the moment it defines the system's capacity to reason.

The next section will present the results of my work in detail. It will discuss each of the components of the system and include:

- Screen captures of the user interface.
- BON static and dynamic architecture diagrams.
- Details of the systems data structures.
- Details of the application database a queries used to perform the Generate step of the reasoning process.
- Details of the knowledge base.

4.0 RESULTS OF PROJECT

4.1 The System Development Cycle

This section of the paper discusses the results of my work done in developing the expert system.

The following steps were performed to develop the GC Expert System:

- Interviews with users of the system were performed in order to obtain their requirements for the GUI. I felt it was important that the system be easy for the user to use. If not, then the system would remain unused regardless of how well it performed it's job.
- As a result of these interviews, a prototype user interface was developed and presented to the users for evaluation of look and feel. This was an iterative process, repeated until the user was satisfied with the interface.
- In addition, development of the user interface led to a model for the systems data structures. These were developed in EiffelCase. The resulting BON models were used to implement the data structure classes. These classes will be discussed in detail later in this paper.
- Interviews with domain experts were performed in order to gather domain knowledge for the rule base. I planned on interviewing at least three different application chemists in order to solicit information with which to generate the knowledge base and on using the initial results to generate a prototype knowledge base for use by the experts. Unfortunately I was unable to complete interviews with more than two experts before I left my position at the GC manufacturer The results of these interviews represents the knowledge based presented in this paper. The development of the knowledge base was an iterative process that was repeated until the knowledge base supplied valid results to sample problems.
- Knowledge Base development led to the enhancement and refinement of system data structures. It also let me begin to understand how the application data base would be queried and allowed the next step to proceed.

- Next, the CLIPS and ACCESS interfaces were designed and implemented. These were tested to verify that they can properly perform their functions using stubs to simulate the communication between them and their targets.
- Design and implemention of the message router.
- Integration and test the system.

The process described above most closely approximates a Spiral Development Process.

4.2 The User Interface

The first part of the expert system developed was the user interface. Interviews with potential system users were conducted in order to determine requirements for the interface. In summary, the following requirements were levied on the user interface:

- It should contain windows that mimic the way in which the sales engineer specifies an application. These windows include: a main application window, a stream properties window, a stream composition window, and a component properties window.
- It should prompt the user for missing information before continuing the input session.
- It should not allow a user to begin a application search until the entire application stream is specified.
- It should allow the input session to be stored on disk.
- It should have a standard Windows look and feel.

Before discussing the user interface components in detail it will be useful to examine the overall architecture of the interface. The BON static architecture of the user interface is shown in the figure below:

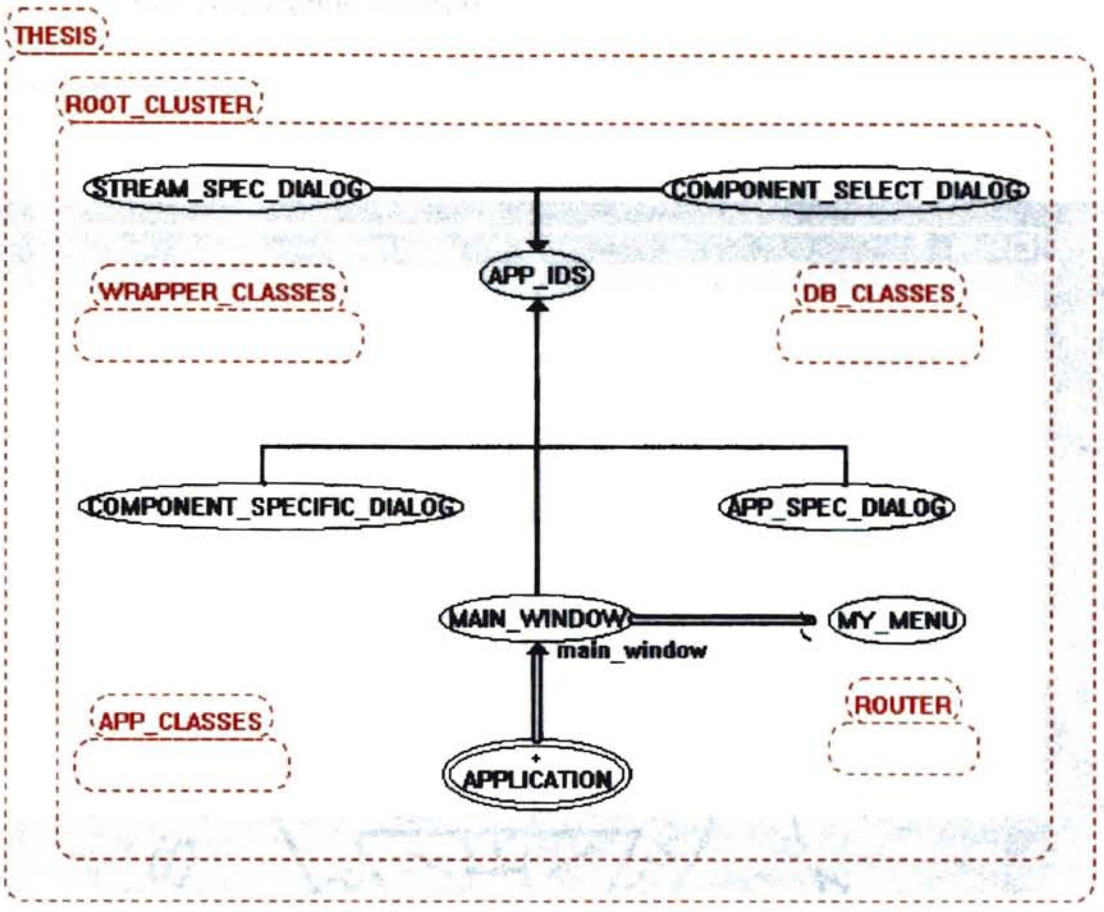


Figure 1 - BON Static Architecture Diagram For the User Interface

The root class of the system is the APPLICATION class. This class is derived from the WEL library class WEL_APPLICATION. The APPLICATION class uses the services of the MAIN_WINDOW class. The MAIN_WINDOW class contains an object of type MY_MENU which is derived from the WEL library class WEL_MENU. The MAIN_WINDOW, COMPONENT_SPECIFIC_DIALOG, COMPONENT_SELECT_DIALOG, STREAM_SPEC_DOALOG, and APP_SPEC_DIALOG classes all inherit from the class APP_IDS. APP_IDS contains the definition of system GUI constants.

Each of the user interface components will now be described in detail to describe how these requirements were implemented as an Eiffel application.

4.2.1 The Main Application Window

The main application window is shown in the figure below.

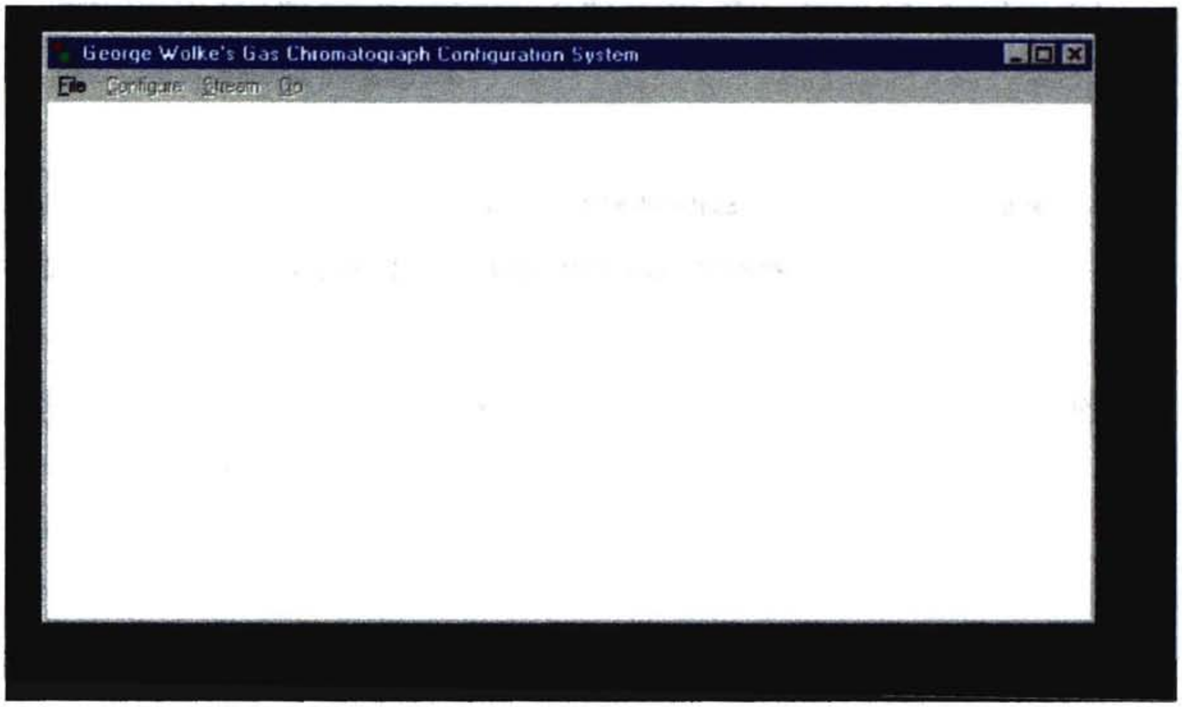


Figure 2 - Main Application Window

This window is created when the user starts the application. It contains a standard Windows menu that is accessed in order to begin an input session. Only the File menu item is active at start up, indicating to the user that he or she must select a File operation to continue. The valid options within the File menu item are:

- New: Used to enter a new application.

- Open: Used to open an existing application that was saved to disk.
- Save: Used to save the current input session to a file on disk.
- Save As: Used to save the current input session to a file under a different file name.
- Print: Used to print the current input session to the printer. This option is not yet implemented.
- Exit: Used to terminate the program.

The main window is an instance of class MAIN_WINDOW which inherits the properties of WEL classes WEL_FRAME_WINDOW, APP_IDS, and WEL_OFN_CONSTANTS.

WEL _FRAME_WINDOW is the library class that abstracts the Windows API for a framed window.

APP_IDS is the class that contains system wide constants (including menu ids).

WEL_OFN_CONTANTS is the library class that contains file operation constants.

The Eiffel source code for the MAIN_WINDOW class is:

```
class
	MAIN_WINDOW
inherit
	WEL_FRAME_WINDOW
		redefine
			class_icon,
			on_menu_command,
			closeable
		end

	APP_IDS
		export
			{NONE} all
		end

	WEL_OFN_CONSTANTS
		export
			{NONE} all
		end
creation
	make

feature {NONE} -- Initialization

	make is
```

```
        do
            make_top (Title)
            set_menu (main_menu)
            current_stream := 0
            app_selected := false
            !! router.make
            !! app_object.make
            !! app_spec.make (Current)
            !! comp_sel.make(Current)
            !! stream_spec.make(Current)
        end

feature {NONE} -- Attributes

    app_spec            :       APP_SPEC_DIALOG
    comp_sel            :       COMPONENT_SELECT_DIALOG
    stream_spec         :       STREAM_SPEC_DIALOG
    current_stream      :       INTEGER
    app_selected        :       BOOLEAN
    app_object          :       GC_APPLICATION
    router              :       MESSAGE_ROUTER

feature {NONE} -- Implementation

    check_configuration is
        local i    : INTEGER
            configured : BOOLEAN
            -- check to see if all streams are configured
        do
            from    i:= 0
                configured := true
            until i = app_object.number_streams

            loop
                if app_object.streams.item(i).get_configured = false then
                    configured := false
                end
                i := i + 1
            end

            if configured then
                main_menu.enable_item(Id_go)
            else
                main_menu.disable_item(Id_go)
            end
            draw_menu
        end
    on_menu_command (menu_id: INTEGER) is
        do
            inspect
                menu_id
            when Id_file_exit then
                if closeable then
                    destroy
```

```
            end
        when Id_file_new then
            if app_selected then
              if save_current then
                  save_system
              end

            app_object.make
            app_selected := false
            reset_menu
            end
            app_spec.activate
        when Id_configure_gaschromatograph_1 ..
        Id_configure_gaschromatograph_15 then
            current_stream := menu_id - Id_configure_gaschromatograph_1
            comp_sel.activate
        when Id_stream_1 .. Id_stream_15 then
            current_stream := menu_id - Id_stream_1
            stream_spec.activate
        when Id_file_open then
          if app_selected then
              if save_current then
                  save_system
              end
            end

            open_system
            app_spec.activate
        when Id_file_sabe then
            save_system
        when Id_file_close then
            if app_selected then
              if save_current then
                  save_system
              end
            end

            set_text(Title)
            app_object.make
            app_selected := false
            reset_menu
        when Id_go then
            -- Debug stuff
            router.start_session(app_object)
        else
            not_implemented
        end
    end

open_system is
        -- open the GC_APPLICATION object
    local   file_name:      STRING
            file_object:    RAW_FILE
            file_window:    WEL_OPEN_FILE_DIALOG
```

```
                temp_app :      GC_APPLICATION

        do
                !!file_window.make
                file_window.set_title("Open Configuration")
                file_window.set_initial_directory_as_current
                file_window.set_default_extension("cfg")
                file_window.set_filter (<<"CFG File">>,<<"*.cfg">>)
                file_window.set_flags(Ofn_pathmustexist +
                                Ofn_filemustexist)
                file_window.activate(Current)

                if file_window.selected = true then
                        !! file_name.make(80)
                        !! temp_app.make
                        file_name.copy(file_window.file_name)
                        !! file_object.make_open_read(file_name.out)
                        app_object ?= temp_app.retrieved(file_object)
                        file_object.close
                        update_title(file_window.file_name)
                        check_configuration
                end
        end

save_system is
                -- save the GC_APPLICATION object
        local   file_name: STRING
                file_object:    RAW_FILE
                file_window:    WEL_SAVE_FILE_DIALOG
        do
                !!file_window.make
                file_window.set_title("Save Configuration")
                file_window.set_initial_directory_as_current
                file_window.set_default_extension("cfg")
                file_window.set_filter (<<"CFG File">>,<<"*.cfg">>)
                file_window.set_flags(Ofn_overwriteprompt)
                file_window.activate(Current)

                if file_window.selected = true then
                        !! file_name.make(80)
                        file_name.copy(file_window.file_name)
                        !! file_object.make_open_write(file_name.out)
                        app_object.general_store(file_object)
                        file_object.close
                        update_title(file_window.file_name)
                end
        end

update_title(file_name : STRING) is
                -- update the project title
        local new_title: STRING
do
        !!new_title.make(80)
        new_title.copy(Title)
```

```
          new_title.append(" - ")
          new_title.append(file_name)
          set_text(new_title)
end

not_implemented is
                    – Message to inform that the feature is not implemented
          do
                    information_message_box("Feature Not Implemented Yet",
                                        "Not Implemented")

          end

closeable: BOOLEAN is
                    – When the user can close the window?
          do
                    Result := message_box ("Do you want to exit?",
                                        "Exit", Mb_yesno + Mb_iconquestion) = Idyes

          end

save_current: BOOLEAN is
                    -- Don't intentionally lose an objects data
          do
                    Result := message_box ("Save current configuration",
                                        "Exit", Mb_yesno + Mb_iconquestion) = Idyes
          end

class_icon: WEL_ICON is
                    – Window's icon
          once
                    !! Result.make_by_id (Id_ico_application)
          end

main_menu: MY_MENU is
                    – Window's menu
          once
                    !! Result.make_by_id (Id_main_menu)
          ensure
                    result_not_void: Result /= Void
          end

setup_menu is
          local i : INTEGER
                    -- enable the stream and configure menus
                    – based on the number of streams attribute
                    -- of the GC_APPLICATION object
do
          main_menu.enable_item_by_position(1)
          main_menu.enable_item_by_position(2)
          main_menu.enable_item(Id_file_sabe)
          main_menu.enable_item(Id_file_saveas)
          main_menu.enable_item(Id_file_print)
          main_menu.enable_item(Id_file_close)
```

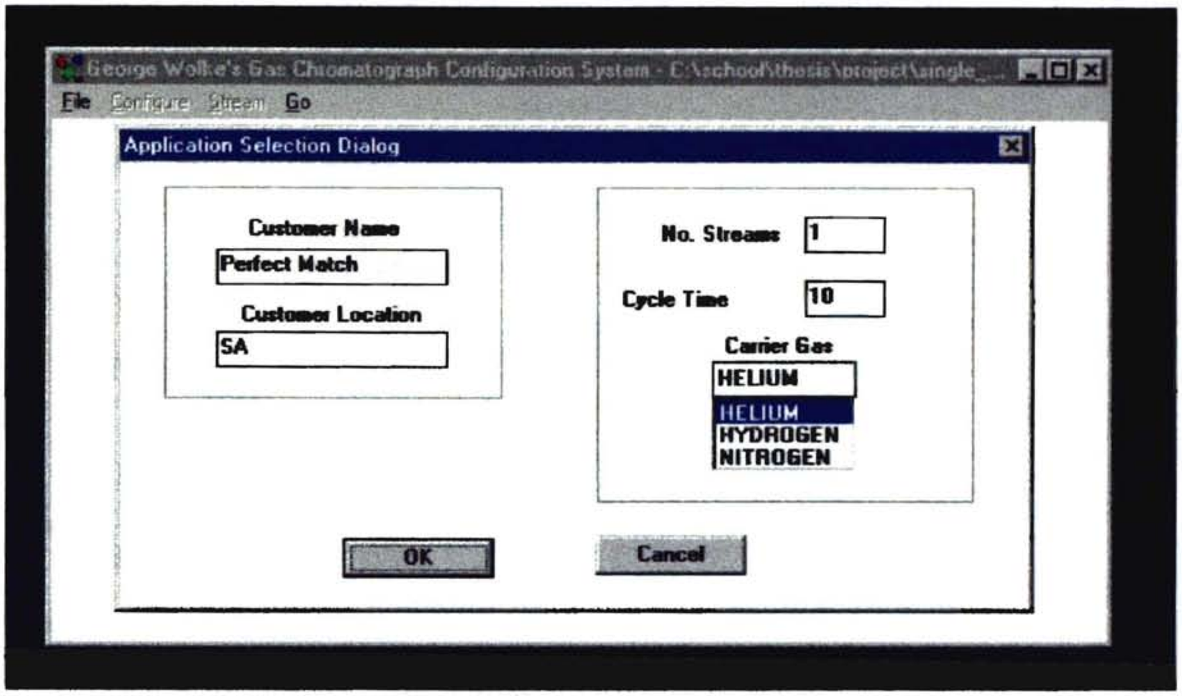
```
                from i := app_object.number_streams
                until i = 15

                loop
                        main_menu.disable_item(Id_configure_gaschromatograph_1 + i)
                        main_menu.disable_item(Id_stream_1 + i)
                        i := i + 1
                end
                draw_menu
        end

        reset_menu is
                local i : INTEGER
                        -- reset the menu to power_up state
        do
                from i := 0
                until i = 15

                loop
                        main_menu.enable_item(Id_configure_gaschromatograph_1 + i)
                        main_menu.enable_item(Id_stream_1 + i)
                        i := i + 1
                end
                main_menu.disable_item_by_position(1)
                main_menu.disable_item_by_position(2)
                main_menu.disable_item(Id_file_sabe)
                main_menu.disable_item(Id_file_saveas)
                main_menu.disable_item(Id_file_print)
                main_menu.disable_item(Id_file_close)
                main_menu.disable_item(Id_go)
                draw_menu
        end

     Title: STRING is "George Wolke's Gas Chromatograph Configuration System"
                        -- Window's title

feature {APP_SPEC_DIALOG} -- Access

        set_app_object(app : GC_APPLICATION) is
                        -- set the GC application object
                        -- finalize the window setup
        do
                app_object := app
                app_selected := true
                setup_menu
        end

        get_app_object : GC_APPLICATION is
                        -- to give access to the GC object
        do
                Result := app_object
        end
```

```
feature {STREAM_SPEC_DIALOG, COMPONENT_SELECT_DIALOG} -- Access

        get_stream_object : STREAM is
                        -- to give access to the current stream
        do
                Result := app_object.get_stream(current_stream)
        end

        set_stream_object(obj : STREAM) is
                        -- give the data to the app object
                        -- look to see if the system is ready
                        -- to run a configuration session
        do
                app_object.set_stream(obj,current_stream)
                check_configuration

        end
end -- class MAIN_WINDOW
```

Notice that the MAIN_WINDOW class contains objects of type GC_APPLICATION and MESSAGE_ROUTER. These are used to implement important system data structures and messaging capabilities and will be discussed later in this section.

4.2.2 The Application Specific Dialog

If the user chooses to enter a new application (by selecting File-New) or to open an existing application (by selecting File-Open) then the following dialog box is displayed:



Customer Name: Perfect Match

Customer Location: SA

No. Streams: 1

Cycle Time: 10

Carrier Gas: HELIUM

Figure 3 - Application Select Dialog Box

This dialog box is an instance of class APP_SPEC_DIALOG, which is derived from the WEL library class WEL_MODAL_DIALOG. WEL_MODAL_DIALOG abstracts the API for a modal dialog. The dialog allows the user to input important information concerning the application. The Customer Name and Customer Location edit boxes are used to store the name and location of the customer. The No. Streams edit box contains the number of streams associated with this application. The Cycle Time edit box stores the time required to perform the analysis. The Carrier Gas combo-box is used to select the carrier gas for the application. After entering this information and selecting the OK pushbutton this information is stored in the GC_APPLICATION class and the dialog box is removed from the screen. At this point the Configure and Stream menu items of the MAIN_WINDOW are enabled.

The Eiffel source code for the dialog box is:

```
class
        APP_SPEC_DIALOG

inherit
        WEL_MODAL_DIALOG
                redefine
                        on_ok,
                        setup_dialog
                end

        APP_IDS
                export
                        {NONE} all
                end

creation
        make

feature {NONE} -- Initialization

        make (a_parent: MAIN_WINDOW) is
                do
                        owner := a_parent

                        make_by_id (a_parent, Idd_appl_select)
                        !! customer.make_by_id (Current, Idc_customer)
                        !! location.make_by_id (Current, Idc_customerlocation)
                        !! no_streams.make_by_id (Current, Idc_streams)
                        !! cycle_time.make_by_id (Current, Idc_cycletime)
                        !! carrier_gas.make_by_id (Current, Idc_carriergas)
                end

feature {NONE} -- Attributes

        customer: WEL_SINGLE_LINE_EDIT
                        -- Customer Edit control

        location: WEL_SINGLE_LINE_EDIT
                        -- Customer Location Edit control

        no_streams: WEL_SINGLE_LINE_EDIT
                        -- Number Streams Edit control

        cycle_time: WEL_SINGLE_LINE_EDIT
                        -- Cycle Time Edit Control
```

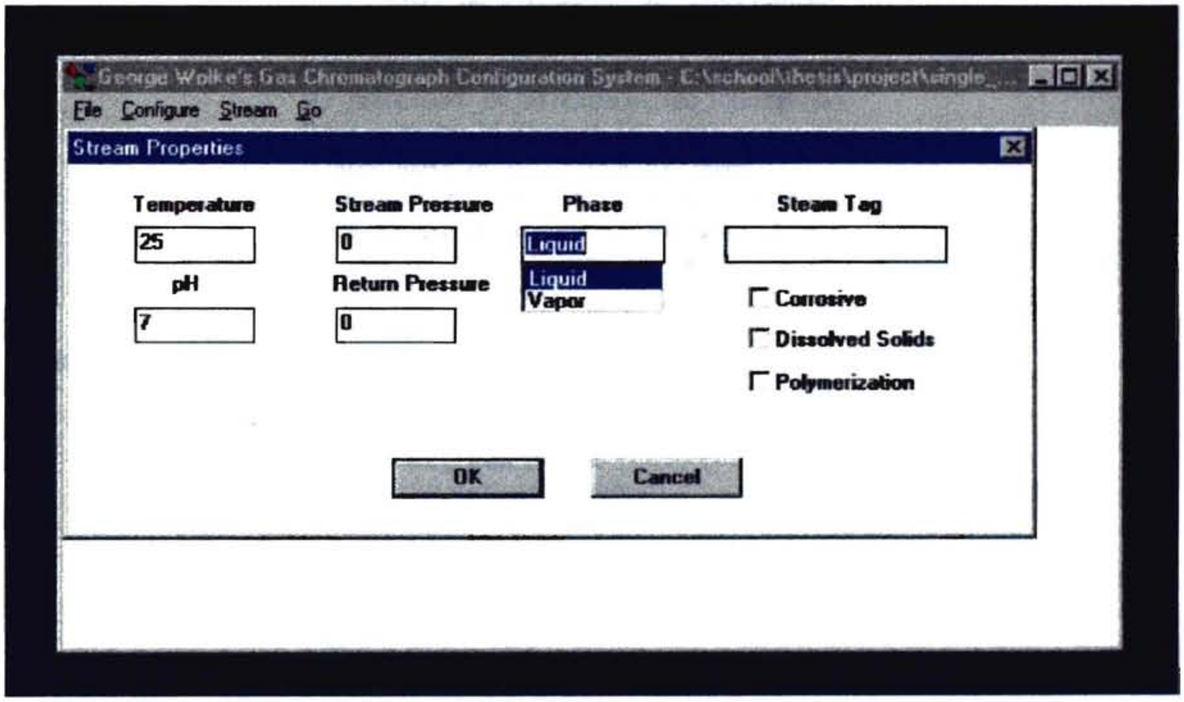
```
        carrier_gas: WEL_SIMPLE_COMBO_BOX
                        -- Carrier Gas List box

        local_application : GC_APPLICATION
                        -- the main application object

        owner: MAIN_WINDOW
                        -- the owner of this object

    feature {NONE} -- Implementation

        setup_dialog is
                local index : INTEGER
                        -- setup the dialog
                do
                        local_application := clone(owner.get_app_object)

                        carrier_gas.add_string("NITROGEN")
                        carrier_gas.add_string("HELIUM")
                        carrier_gas.add_string("HYDROGEN")

                        index := carrier_gas.find_string_exact(0,local_application.carrier_gas)
                        if index /= -1 then
                                carrier_gas.select_item(index)
                        else
                                carrier_gas.select_item(0)
                        end

                        customer.set_text(local_application.customer_name)
                        location.set_text(local_application.customer_location)
                        no_streams.set_text(local_application.number_streams.out)
                        cycle_time.set_text(local_application.cycle_time.out)
                end

        on_ok is
                local stream, c_time : INTEGER
                        -- Save the application data
                do
                        if no_streams.text.is_integer and cycle_time.text.is_integer then
                           stream := no_streams.text.to_integer
                           c_time := cycle_time.text.to_integer
                           if stream > 15 or stream < 1 then
                                information_message_box("Set number of streams (1 - 15)",
                                        "Required Entry")
                                no_streams.select_all
                           elseif c_time <= 0 then
                                information_message_box("Cycle time must be greater than 0",
                                        "Required Entry")
                                cycle_time.select_all
                           else
                                local_application.set_customer_name(customer.text)
                                local_application.set_customer_location(location.text)
                                local_application.set_streams(no_streams.text.to_integer)
                                local_application.set_cycle_time(cycle_time.text.to_integer)
```

```
                    local_application.set_carrier_gas(carrier_gas.selected_string)
                    signal_owner
                    terminate (Idok)
               end
            elseif not no_streams.text.is_integer then
               information_message_box("Streams must be an integer (1-15)",
                         "Entry Error")
               no_streams.select_all
            else
               information_message_box("Cycle time must be greater than 0",
                         "Entry Error")
           cycle_time.select_all
            end
      end

   signal_owner is
         -- inform the owner that the app object is filled
   do
         owner.set_app_object(local_application)
   end

end -- class APP_SPEC_DIALOG
```

The feature of most interest in this class is 'on_ok'. This is called when the user selects the OK push button. This feature verifies that all of the required information has been entered, stores this information in a GC_APPLICATION object and calls the 'signal_owner' feature. The 'signal_owner' feature is used to inform the MAIN_WINDOW that the application specific data has been entered. This is accomplished by calling the 'set_app_object' feature of MAIN_WINDOW, passing as a parameter an object of type GC_APPLICATION.

4.2.3 Stream Select Dialog

After completing the Application Select Dialog, the Stream and Configure menu items are enabled. At this point the user may choose to select the Stream menu item. A sub-menu containing entries for each of the streams specified in the No. Streams edit field will be displayed. If one of the sub-menu items is selected the dialog box shown below will be displayed. This dialog box is an instance of class STREAM_SPEC_DIALOG, which is derived from the WEL library class WEL_MODAL_DIALOG. WEL_MODAL_DIALOG abstracts the API for a modal dialog. The dialog box is used to specify important parameters for the application stream.



Temperature 25
Stream Pressure 0
Phase Liquid
pH 7
Return Pressure 0

Figure 4 - Stream Properties Dialog

The Eiffel source code for this dialog box is:

```
class
        STREAM_SPEC_DIALOG

inherit
        WEL_MODAL_DIALOG
                redefine
```

```
                        on_ok,
                        setup_dialog
                end

        APP_IDS
                export
                        {NONE} all
                end

creation
        make

feature {NONE} -- Initialization

        make (a_parent: MAIN_WINDOW) is
                do
                        owner := a_parent

                        make_by_id (a_parent, Idd_stream_properties)
                        !! temp.make_by_id (Current, Idc_temperature)
                        !! pH.make_by_id (Current, Idc_ph)
                        !! s_press.make_by_id (Current, Idc_streampress)
                        !! r_press.make_by_id (Current, Idc_returnpress)
                        !! tag.make_by_id (Current, Idc_tag)
                        !! phase.make_by_id (Current, Idc_phase)
                        !! corrosive.make_by_id (Current, Idc_corrosive)
                        !! solids.make_by_id (Current, Idc_dissolids)
                        !! poly.make_by_id (Current, Idc_poly)
                end

feature {NONE}-- Attributes

        temp: WEL_SINGLE_LINE_EDIT
                        -- stream temp Edit control

        pH: WEL_SINGLE_LINE_EDIT
                        -- stream pH Edit control

        s_press: WEL_SINGLE_LINE_EDIT
                        -- stream pressure Edit control

        r_press: WEL_SINGLE_LINE_EDIT
                        -- return pressure Edit Control

        tag: WEL_SINGLE_LINE_EDIT
                        -- the app specific name for this stream

        phase: WEL_SIMPLE_COMBO_BOX
                        -- stream phase List box

        corrosive: WEL_CHECK_BOX
                        -- is the stream corrosive

        solids: WEL_CHECK_BOX
```

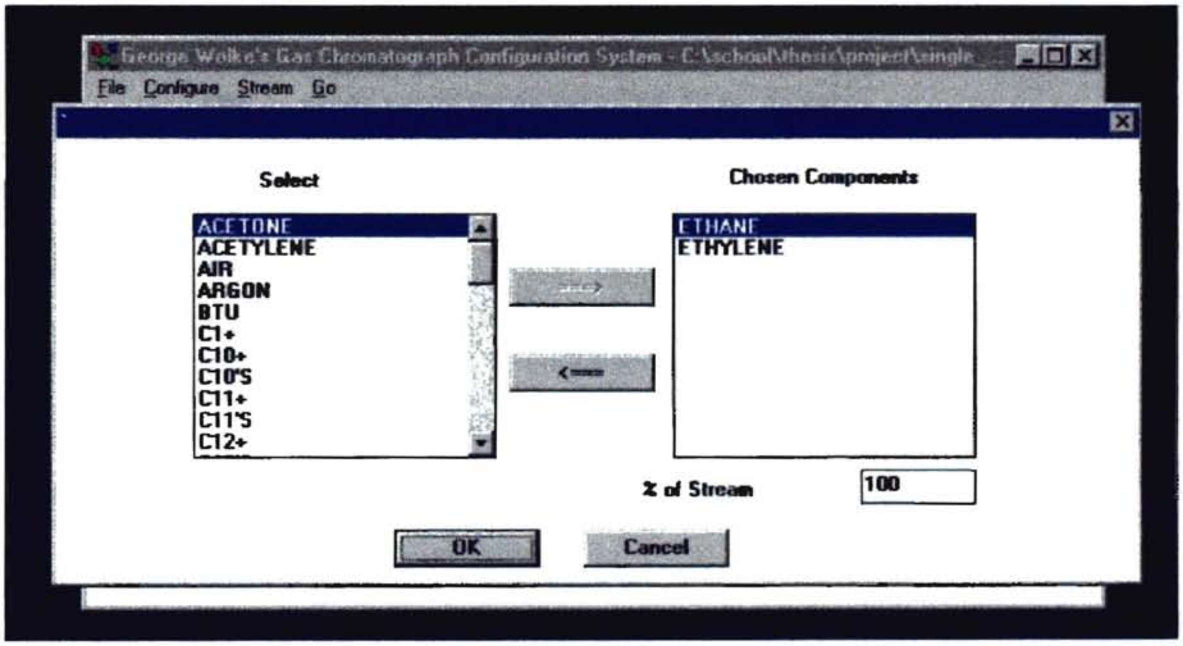
```
                    -- are dissolved solids in the stream

        poly: WEL_CHECK_BOX
                    -- does the stream polimerize

        local_stream : STREAM
                    -- used to store the changes to the stream
                    -- attributes

        owner: MAIN_WINDOW
                    -- the owner of this object


feature {NONE} -- Implementation

        setup_dialog is
                local index : INTEGER
                    -- setup the dialog
                do
                    local_stream := clone(owner.get_stream_object)

                    temp.set_text(local_stream.get_temperature.out)
                    pH.set_text(local_stream.get_pH.out)
                    s_press.set_text(local_stream.get_spress.out)
                    r_press.set_text(local_stream.get_rpress.out)
                    tag.set_text(local_stream.get_tag)

                    phase.add_string("Liquid")
                    phase.add_string("Vapor")

                    index := phase.find_string_exact(0,local_stream.get_phase)
                    if index /= -1 then
                            phase.select_item(index)
                    else
                            phase.select_item(0)
                    end

                    if local_stream.get_corrosive = true then
                            corrosive.set_checked
                    end

                    if local_stream.get_disolids = true then
                            solids.set_checked
                    end

                    if local_stream.get_polimer = true then
                            poly.set_checked
                    end


                end

        on_ok is
                    -- Save the stream data
```

```
        do
            if temp.text.is_real and pH.text.is_integer and
                s_press.text.is_real and  r_press.text.is_real then
                local_stream.set_temperature(temp.text.to_real)
                local_stream.set_pH(pH.text.to_integer)
                local_stream.set_spress(s_press.text.to_real)
                local_stream.set_spress(r_press.text.to_real)
                local_stream.set_tag(tag.text)
                local_stream.set_phase(phase.selected_string)
                local_stream.set_corrosive(corrosive.checked)
                local_stream.set_polimer(poly.checked)
                local_stream.set_disolids(solids.checked)
                signal_owner
                terminate (Idok)
            else
                if not temp.text.is_real then
                   information_message_box("Temperature must be an real number",
                        "Entry Error")
                   temp.select_all
                elseif not pH.text.is_integer then
                   information_message_box("pH must be an integer",
                        "Entry Error")
                   pH.select_all
                elseif not s_press.text.is_real then
                   information_message_box("Stream Pressure must be a real
number",
                        "Entry Error")
                    s_press.select_all
                else
                   information_message_box("Return Pressure must be a real
number",
                        "Entry Error")
                   r_press.select_all
                end
            end
        end

    signal_owner is
        -- inform the owner that the stream object is filled
    do
        owner.set_stream_object(local_stream)
    end
```

The 'on_ok' and 'signal owner' features are used in the same manner as for the APP_SPEC_DIALOG class. The 'on_ok' feature is called when the OK push button is selected. It verifies that important information has been fully defined, stores this information in a STREAM object and then calls 'signal_owner'. The 'signal_owner' feature calls 'set_stream_object' of MAIN_WINDOW to store this STREAM object within the GC_APPLICATION object.

4.2.4 Component Select Dialog

After completing the Application Select Dialog, the Stream and Configure menu items are enabled. At this point the user may choose to select the Configure menu item. A sub-menu containing entries for each of the streams specified in the No. Streams edit field will be displayed. If one of the sub-menu items is selected the dialog box shown below will be displayed. This dialog box is an instance of class COMPONENT_SELECT_DIALOG, which is derived from the WEL library class WEL_MODAL_DIALOG. WEL_MODAL_DIALOG abstracts the API for a modal dialog. The dialog box is used to specify the component make-up of this application stream.



% of Stream 100

Figure 5 - Component Select Dialog

The dialog box contains two list boxes. The 'Select' list box contains the possible components of a stream and the 'Chosen Components' list box contains the components that make up this process stream. The user may select a component by either 'double clicking' on the item or by selecting the item and then pushing the 'right arrow' push button. The 'left arrow' push button is used to remove a component from the 'Chosen Components' list box. The '% of Stream' edit box shows the total percentage of the stream that has been specified. A reasoning session may only be run for a completely specified stream.

The Eiffel source code for the dialog box is:

```
class
        COMPONENT_SELECT_DIALOG

inherit
        WEL_MODAL_DIALOG
                redefine
                        on_ok,
                        setup_dialog,
                        notify
                end

        APP_IDS
                export
                        {NONE} all
                end

        WEL_LBN_CONSTANTS
                export
                        {NONE} all
                end

        WEL_BN_CONSTANTS
                export
                        {NONE} all
                end

creation
        make

feature {NONE} -- Initialization

        make (a_parent: MAIN_WINDOW) is
                do
                        owner := a_parent
                        make_by_id (a_parent, Idd_comp_select)

                        !! comp_list.make_by_id (Current, Idc_chosen)
                        !! sel_list.make_by_id (Current, Idc_select)
                        !! stream_percent.make_by_id (Current, Idc_streampercent)
                        !! insert_button.make_by_id(Current,Idc_tochosen)
                        !! removed_button.make_by_id(Current,Idc_toselect)
                        !!local_component.make
```

```
                end

feature {NONE}-- Attributes

        comp_list: WEL_SINGLE_SELECTION_LIST_BOX
                        -- list of possible components List box

        sel_list: WEL_SINGLE_SELECTION_LIST_BOX
                        -- list of possible components List box

        stream_percent: WEL_SINGLE_LINE_EDIT
                        -- edit showing total % of stream defined

        insert_button: WEL_PUSH_BUTTON
                        -- add items to the selected list

        removed_button: WEL_PUSH_BUTTON
                        -- remove items from the selected list

        comp_dialog: COMPONENT_SPECIFIC_DIALOG
                        -- the popup that configures a component

        local_stream : STREAM
                        -- used to store the changes to the stream

        local_component: COMPONENT
                        -- used to store the component we are working on

        owner: MAIN_WINDOW
                        -- the owner of this object

feature {NONE} -- Implementation

        setup_dialog is
                local  i,j    : INTEGER

                        -- setup the dialog
                do
                        local_stream := clone(owner.get_stream_object)


                        calculate_percent_stream
                        sel_list.add_string("HYDROGEN")
                        sel_list.add_string("HELIUM")
                        sel_list.add_string("NITROGEN")
                        sel_list.add_string("AIR")
                        sel_list.add_string("BTU")
                        sel_list.add_string("ACETONE")
                        sel_list.add_string("ARGON")
                        sel_list.add_string("CARBONDIOXIDE")
                        sel_list.add_string("CARBONMONIXIDE")
                        sel_list.add_string("CARBONTETRACHLORIDE")
                        sel_list.add_string("CARBONTETRAFLORIDE")
                        sel_list.add_string("CHLORINE")
```

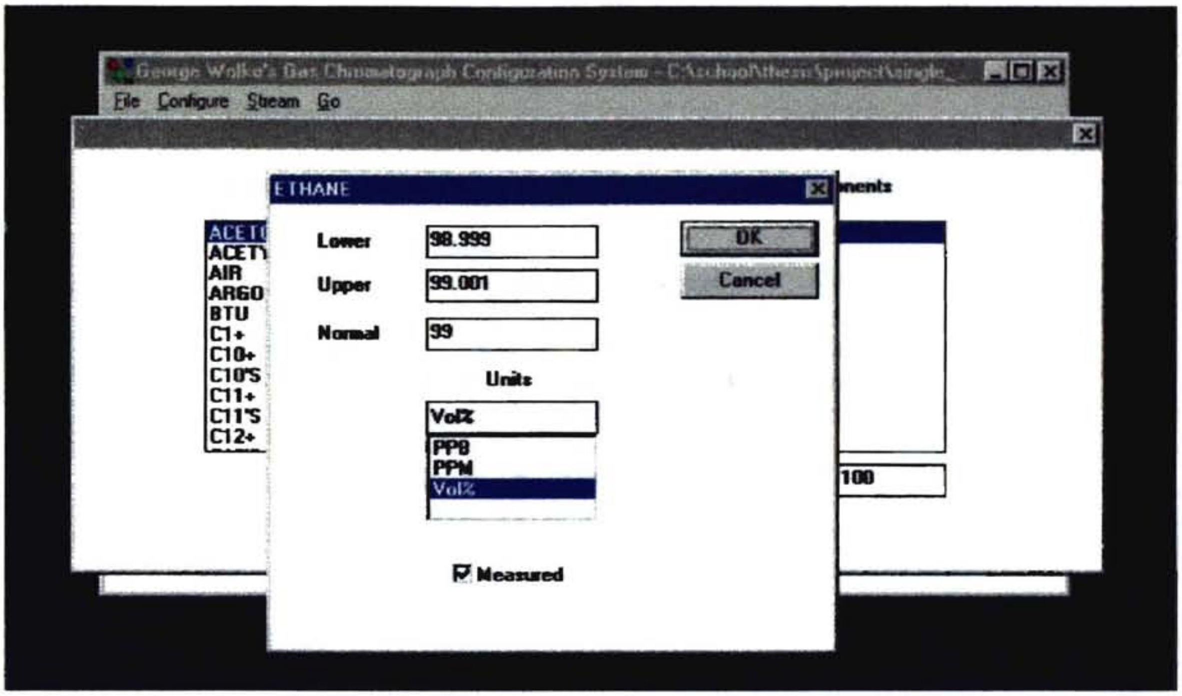
```
                sel_list.add_string("C1+")
                sel_list.add_string("C2+")
                sel_list.add_string("C3+")
                sel_list.add_string("C4+")
                sel_list.add_string("C5=+")
                sel_list.add_string("C6+")
                sel_list.add_string("C7+")
                sel_list.add_string("C8+")
                sel_list.add_string("C9+")
                sel_list.add_string("C10+")
                sel_list.add_string("C11+")
                sel_list.add_string("C12+")
                sel_list.add_string("C2'S")
                sel_list.add_string("C3'S")
                sel_list.add_string("C4'S")
                sel_list.add_string("C5'S")
                sel_list.add_string("C6'S")
                sel_list.add_string("C7'S")
                sel_list.add_string("C8'S")
                sel_list.add_string("C9'S")
                sel_list.add_string("C10'S")
                sel_list.add_string("C11'S")
                sel_list.add_string("C12'S")
                sel_list.add_string("METHANE")
                sel_list.add_string("ETHANE")
                sel_list.add_string("ETHYLENE")
                sel_list.add_string("ACETYLENE")
                sel_list.add_string("PROPANE")
                sel_list.add_string("PROPYLENE")
                sel_list.add_string("CYCLOPROPANE")
                sel_list.add_string("PROPADIANE")
                sel_list.add_string("METHYLACETYLENE")
                sel_list.add_string("ISOBUTANE")
                sel_list.add_string("NORMALBUTANE")
                sel_list.add_string("I-BUTENE")
                sel_list.add_string("TRANS-2-BUTENE")
                sel_list.add_string("ETHYLACETYLENE")
                sel_list.add_string("VINYLACETYLENE")
                sel_list.select_item(0)

                -- update the list boxes
                from i := 1
                until i > local_stream.get_number_of_components
                loop
                   comp_list.add_string(local_stream.get_component_by_index(i).get_name)
                   j := sel_list.find_string(0,
local_stream.get_component_by_index(i).get_name)
                   if j /= -1 then
                          sel_list.delete_string(j)
                   end
                   i := i + 1
                end

                if i = 1 then
```

```
                                        removed_button.disable
                                else
                                        comp_list.select_item(0)
                                end
                        end

                calculate_percent_stream is
                        local percent : REAL
                                -- walk through the components and
                                -- calculate the percentage of the stream
                                -- that has been defined.  Set the configured
                                -- state of the stream
                        do
                                percent := local_stream.get_percent_stream
                                stream_percent.set_text(percent.out)
                                if percent >= 100 then
                                        insert_button.disable
                                else
                                        insert_button.enable
                                end

                                if percent > 100 then
                                        information_message_box("An error exists in the stream.  Stream %%
> 100",
                                                "Stream Composition Error")
                                end

                                if percent = 100 then
                                        local_stream.set_configured
                                else
                                        local_stream.reset_configured
                                end

                        end

                add_component_to_list is
                                -- add a component to the component list
                        local   buffer          : STRING
                                selection       : INTEGER
                        do
                                !!buffer.make(80)
                                buffer := sel_list.selected_string
                                selection := sel_list.selected_item

                                local_stream.add_component(buffer)
                                local_component := local_stream.get_component_by_name(buffer)

                                comp_list.add_string(buffer)
                                sel_list.delete_string(selection)

                                sel_list.select_item(0)
                                comp_list.select_item(0)
                                if comp_list.count = 1 then
                                        removed_button.enable
```

```
                end

                !!comp_dialog.make(Current,buffer)
                comp_dialog.activate

        end

remove_component_from_list is
                – remove a component from the list
        local   buffer  : STRING
                selection       : INTEGER
                comp            : COMPONENT
                percent         : REAL
        do
                !!buffer.make(80)
                !!comp.make
                buffer := comp_list.selected_string
                selection := comp_list.selected_item

                comp := local_stream.get_component_by_name(buffer)
                local_stream.remove_component(comp)

                sel_list.add_string(buffer)
                comp_list.delete_string(selection)

                sel_list.select_item(0)
                if comp_list.count = 0 then
                        removed_button.disable
                else
                        comp_list.select_item(0)
                end

                calculate_percent_stream
        end


notify (control: WEL_CONTROL; notify_code: INTEGER) is
        local buffer: STRING

        do
                if control = sel_list then
                        if notify_code = Lbn_dblclk then
                                -- add the component to the component list
                                add_component_to_list
                        end
                elseif control = insert_button then
                        if notify_code = Bn_clicked then
                                -- add the component to the component list
                                add_component_to_list
                        end
                elseif control = comp_list then
                        if notify_code = Lbn_dblclk then
                                -- Bring up the component properties dialog
                                !!buffer.make(80)
```

```
                                        buffer := comp_list.selected_string
                                        local_component :=
local_stream.get_component_by_name(buffer)
                                        !!comp_dialog.make(Current,buffer)
                                        comp_dialog.activate
                                end
                        elseif control = removed_button then
                                if notify_code = Bn_clicked then
                                        -- remove the component from the component list
                                        remove_component_from_list
                                end

                        end
                end

        on_ok is
                        -- Save the application data
                do
                        signal_owner
                        terminate (Idok)
                end

        signal_owner is
                -- inform the owner that the stream object is filled
        do
                owner.set_stream_object(local_stream)
        end

feature {COMPONENT_SPECIFIC_DIALOG} -- ACCESS

        get_component_object : COMPONENT is
                -- get the current component_object
        do
                Result := local_component
        end

        update_component_list(comp : COMPONENT) is
                -- update the list with new component specific information
                -- update the stream percent dialog
        do
                local_stream.replace_component(comp)
                calculate_percent_stream
        end


end -- class COMPONENT_SELECT_DIALOG
```

When the user selects a component by double-clicking or selecting the 'Right Arrow' push button, the 'add_component_to_list' feature is called and the following dialog box is displayed:



Lower: 98.999
Upper: 99.001
Normal: 99
Units: Vol%
☑ Measured

Figure 6 - Component Specific Dialog Box

This dialog allows the user to specify the concentration of this component within the stream. The 'Upper', 'Lower', and 'Normal' edit fields contains the upper, lower, and normal concentrations of the component within the process stream. The 'measured' check box specifies if this component is to be analyzed or not. Only measured components are considered by the knowledge base.

The Eiffel Source code for this class is:

```
class
        COMPONENT_SPECIFIC_DIALOG

inherit
        WEL_MODAL_DIALOG
                redefine
                        on_ok,
                        setup_dialog
                end

        APP_IDS
                export
                        {NONE} all
                end
```

```
creation
        make

feature {NONE} -- Initialization

        make (a_parent: COMPONENT_SELECT_DIALOG; component: STRING) is
                do
                        owner := a_parent
                        make_by_id (a_parent,Idd_comp_prop)

                        !! upper_limit.make_by_id (Current, Idc_upperlimit)
                        !! lower_limit.make_by_id (Current, Idc_lowerlimit)
                        !! measured.make_by_id (Current, Idc_measured)
                        !! normal.make_by_id(Current, Idc_normal)
                        !! units.make_by_id(Current,Idc_units)
                        !!my_title.make(80)
                        my_title.copy(component)
                end

feature {NONE}-- Attributes


        upper_limit: WEL_SINGLE_LINE_EDIT
                        -- edit showing upper concentration limit

        lower_limit: WEL_SINGLE_LINE_EDIT
                        -- edit showing upper concentration limit

        normal: WEL_SINGLE_LINE_EDIT
                        -- edit showing normal concentration

        units: WEL_SIMPLE_COMBO_BOX
                        -- combo box listing the possible units

        measured: WEL_CHECK_BOX
                        -- Is this component analyzed?

        my_title: STRING

        owner : COMPONENT_SELECT_DIALOG
                        -- the owner of this window

        local_component : COMPONENT
                        -- this component

feature {NONE} -- Implementation

        setup_dialog is
                local index : INTEGER
                        -- setup the dialog
                do
                        local_component := clone(owner.get_component_object)

                        units.add_string("PPM")
```

```
                        units.add_string("PPB")
                        units.add_string("Vol%%")

                        lower_limit.set_text(local_component.get_minimum.out)
                        upper_limit.set_text(local_component.get_maximum.out)
                        normal.set_text(local_component.get_normal.out)

                        if local_component.get_measured = true then
                                measured.set_checked
                        end

                        index := units.find_string_exact(0,local_component.get_units)
                        if index /= -1 then
                                units.select_item(index)
                        else
                                units.select_item(0)
                        end
                        set_text(my_title)
                end

        on_ok is
                        -- Save the application data
                do
                        if not normal.text.is_real or not upper_limit.text.is_real
                                or not lower_limit.text.is_real then
                          if not normal.text.is_real then
                            information_message_box("Normal concentration must be a real number",
                                        "Entry Error")
                                normal.select_all
                          elseif not upper_limit.text.is_real then
                                 information_message_box("Upper limit concentration must be a real
number",
                                        "Entry Error")
                                upper_limit.select_all
                          else
                                information_message_box("Lower limit concentration must be a real
number",
                                        "Entry Error")
                                lower_limit.select_all
                          end
                        elseif upper_limit.text.to_real < lower_limit.text.to_real then
                          information_message_box("Upper Limit must be greater than lower limit",
                                        "Entry Error")
                        elseif upper_limit.text.to_real < 0 then
                          information_message_box("Upper Limit must be greater than or equal to 0",
                                        "Entry Error")
                        elseif lower_limit.text.to_real < 0 then
                          information_message_box("Lower Limit must be greater than or equal to 0",
                                        "Entry Error")
                        elseif normal.text.to_real <= 0 then
                          information_message_box("Normal concentration must be greater than 0",
                                        "Entry Error")
                        elseif lower_limit.text.to_real >= normal.text.to_real then
```

```
                        information_message_box("Lower Limit must be less than normal
concentration",
                                     "Entry Error")
                    elseif upper_limit.text.to_real <= normal.text.to_real then
                        information_message_box("Upper limit must be greater than normal
concentration",
                                     "Entry Error")
                    else
                        local_component.set_normal(normal.text.to_real)
                        local_component.set_maximum(upper_limit.text.to_real)
                        local_component.set_minimum(lower_limit.text.to_real)
                        local_component.set_measured(measured.checked)
                        local_component.set_units(units.selected_string)
                        signal_owner
                        terminate (Idok)
                    end
            end


        signal_owner is
                    -- update the component object
        do
            owner.update_component_list(local_component)
        end
end -- class COMPONENT_SPECIFIC_DIALOG
```

The 'on_ok' feature is used in the same manner as described earlier. It verifies that data has been properly entered, copies the data into a COMPONENT object, and then calls the 'signal_owner' feature. The 'signal_owner' feature calls 'update_component_list' of COMPONENT_SELECT_DIALOG to add the component to the STREAM object.

When 'update_component_list' is called the COMPONENT_SELECT_DIALOG object will call the 'get_percent_stream' feature of class STREAM to update the percentage of the stream that has been specified. When the entire stream has been specified then the OK push button of the COMPONENT_SELECT_DIALOG window is used to signal the MAIN_WINDOW that the stream has been configured. At this point the updated STREAM object is inserted into the GC_APPLICATION object and the GO menu item is enabled. This signifies that a reasoning session can now take place.

4.3 The Application Classes

The Application classes are the main data structures for the expert system. Together they model the structure of a GC and are used to create queries of the application data base in order to perform the 'generate' phase of the reasoning session. The BON static architecture of the Application classes is shown in the figure below.

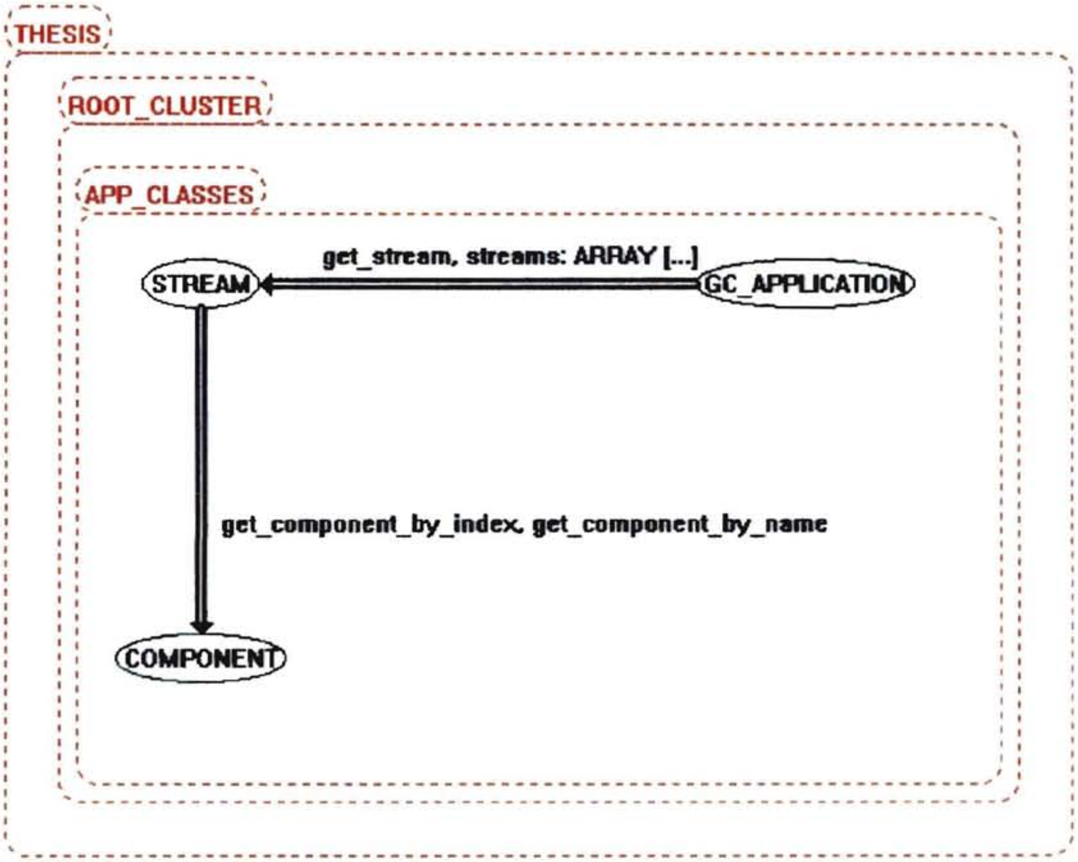


Figure 7 - Application Class Static Architecture

The three classes that comprise the Application classes will now be discussed in more detail.

GC_APPLICATION

This is the root of the application class and models a GC. It contains an array of STREAM objects. An array representation was chosen instead of a linked-list because the maximum number of streams a GC may contain is fixed. When a GC_APPLICATION object is created, an array of STREAM objects is also created and initalized to a known, empty, state. The STREAM in turn creates an empty linked list of COMPONENT objects. The GC_APPLICATION class inherits from the library class STORABLE. This allows the object and its components to be stored in a platform independent manner. Exactly one GC_APPLICATION object exists within the system. It is created within the MAIN_WINDOW class and accessed from the APP_SPEC_DIALOG. The Eiffel source code for the GC_APPLICATION class is:

```
indexing
description: "The main application class"

class GC_APPLICATION

inherit

        STORABLE

creation make

feature  -- creation

        make is
                -- initialize the object
        do
                !!customer_name.make(80)
                !!customer_location.make(80)
                !!carrier_gas.make(80)
                !!streams.make(0, 14)
                fill_streams

                cycle_time := 0
                number_streams := 0
        end

feature

        customer_name: STRING
                        -- The customer name

        customer_location: STRING
```

```
                -- The customer location

number_streams: INTEGER
                -- The number of streams in this application

cycle_time: INTEGER
                -- The application cycle time

carrier_gas: STRING
                -- The carrier gas for the application

streams : ARRAY[STREAM]
                -- The stream aggregate attribute

set_customer_name (name: STRING) is
                -- Set the customer name
        require
          exists: name /= void
        do
                customer_name := name
        ensure
          configured: customer_name = name
        end

set_customer_location (location: STRING) is
                -- set the customer location
        require
          exists: location /= void
        do
                customer_location := location
        ensure
          configured: customer_location = location
        end

set_streams (number: INTEGER) is
                -- set the number of streams
        require
          exists: number /= void
        do
                number_streams := number
        ensure
          configured: number_streams = number
        end

set_cycle_time (time: INTEGER) is
                -- set the cycle time in seconds
        require
          exists: time /= void
        do
                cycle_time := time
        ensure
          configured: cycle_time = time
        end
```

```
    set_carrier_gas (gas: STRING) is
                -- set the carrier gas for the application
            require
              exists: gas /= void
            do
                carrier_gas := gas
            ensure
              configured: carrier_gas = gas
            end

    get_stream(index : INTEGER) : STREAM is
                -- get the indexed stream object
            require
              valid_index: index >= 0
              exists: index /= void
            do
                Result := streams.item(index)
            ensure
              stream_obtained: result = streams.item(index)
            end

    set_stream(s : STREAM ; index : INTEGER) is
                -- put the stream into the array
            require
              stream_exists: s /= void
              index_exists: index /= void
              index_valid: index >= 0
            do
                streams.put(s, index)
            ensure
              stream_set: streams.item(index) = s
            end

    fill_streams is
            local s : STREAM
                i : INTEGER

                -- fill the stream array
            do
                from i := 0
                until i = 15
                loop
                        !!s.make
                        streams.put(s,i)
                        i := i + 1
                end
            end

end -- class GC_APPLICATION
```

STREAM

The STREAM class models a GC process stream. A process stream is a unique set of components for which an analysis is performed. Up to 15 STREAM objects may be defined for the GC_APPLICATION object and the number of components within a stream is variable. Therefore, the components are contained within a linked-list of COMPONENT objects. The attributes of this class are updated from the STREAM_SPEC_DIALOG class described earlier. The Eiffel source code for the class is:

```
indexing
        description: "The application stream"

class STREAM

creation make

feature -- creation

        make is
                – initialize the object
        do
                !! components.make
                configured := false
                temperature := 25.00
                pH := 7
                s_pressure := 0.00
                r_pressure := 0.00
                corrosive := false
                dis_solids := false
                polimer := false
                !!phase.make(20)
                !!tag.make(80)
        end

feature {NONE} -- attributes

        components : LINKED_LIST[COMPONENT]
                        -- the list of components for this stream

        corrosive : BOOLEAN
                        -- is the stream corrosive

        dis_solids : BOOLEAN
                        -- does the stream have disolved solids

        polimer : BOOLEAN
```

```
                        -- does the stream polimerize

        temperature: REAL
                        -- The stream temperature

        pH: INTEGER
                        -- The stream pH

        s_pressure: REAL
                        -- The stream pressure

        r_pressure: REAL
                        -- The return pressure

        phase: STRING
                        -- The phase of the gas in this stream

        tag: STRING
                        -- The customer specific tag of this stream

        configured : BOOLEAN
                        -- set true if 100% of the components are
                        -- specified

feature -- ACCESS

        set_configured is
                        -- set the configured state
                do
                        configured := true
                ensure
                   configured: configured = true
                end

        reset_configured is
                        -- reset the configured state
                do
                        configured := false
                ensure
                   configured: configured = false
                end

        set_polimer (state : BOOLEAN) is
                        -- set the polimer state
                require
                   exists: state /= void
                do
                        polimer := state
                ensure
                   configured: polimer = state
                end

        set_corrosive (state : BOOLEAN) is
                        -- set the corrosive state
```

```
        require
           exists: state /= void
        do
                corrosive := state
        ensure
           configured: corrosive = state
        end

set_disolids (state : BOOLEAN) is
                -- set the solids state
        require
           exists: state /= void
        do
                dis_solids := state
        ensure
           configured: dis_solids = state
        end
set_temperature (t : REAL) is
                -- Set the temperature
        require
           exists: t /= void
        do
                temperature := t
        ensure
           configured: temperature = t
        end

set_pH (p : INTEGER) is
                -- Set the pH
        require
           exists: p /= void
        do
                pH := p
        ensure
           configured: pH = p
        end

set_spress (number: REAL) is
                -- set the stream pressure
        require
           exists: number /= void
        do
                s_pressure := number
        ensure
           configured: s_pressure = number
        end

set_rpress (number: REAL) is
                -- set the return pressure
        require
           exists: number /= void
        do
                r_pressure := number
        ensure
```

```
            configured: r_pressure = number
        end

    set_phase (p: STRING) is
                -- set the phase for the stream
        require
            exists: p /= void
        do
                phase := p
        ensure
            configured: phase = p
        end

    set_tag (t: STRING) is
                -- set the tag for the stream
        require
            exists: t /= void
        do
                tag := t
        ensure
            configured: tag = t
        end

    get_configured : BOOLEAN is
                -- get the configured state
        do
                Result := configured
        ensure
            result = configured
        end

    get_temperature : REAL is
                -- get the temperature
        do
                Result := temperature
        ensure
            result = temperature
        end

    get_pH : INTEGER is
                -- get the pH
        do
                Result := pH
        ensure
            result = pH
        end

    get_spress : REAL is
                -- get the stream pressure
        do
                Result := s_pressure
        ensure
            result = s_pressure
        end
```

```
get_rpress : REAL is
                -- get the return pressure
        do
                Result := r_pressure
        ensure
           result = r_pressure
        end

get_phase : STRING is
                -- get the phase for the stream
        do
                Result := phase
        ensure
           result = phase
        end

get_tag : STRING is
                -- get the tag for the stream
        do
                Result := tag
        ensure
           result = tag
        end

get_polimer : BOOLEAN is
                -- get the polimer state
        do
                Result := polimer
        ensure
           result = polimer
        end

get_corrosive : BOOLEAN is
                -- get the corrosive state
        do
                Result := corrosive
        ensure
           result = corrosive
        end

get_disolids : BOOLEAN is
                -- get the solids state
        do
                Result := dis_solids
        ensure
           result = dis_solids
        end

add_component(name : STRING) is
                -- add a component to the
                -- list of components
        require
           exists: name /= void
```

```
        local comp : COMPONENT
        do
                !!comp.make
                comp.set_name(name)
                components.extend(comp)
        ensure
          one_more_comp: components.count = 1 +
                                old components.count
        end

get_component_by_name(name : STRING) : COMPONENT is
                -- get a component from the
                -- list having the name attribute
                -- given in name
        require
          exists: name /= void
        local position : INTEGER

        do
                from components.start
                until components.item.get_name.is_equal(name)
                loop
                        components.forth
                end

                Result := components.item
        end

remove_component(comp : COMPONENT) is
                -- remove a component from the list
        require
          exists: comp /= void
        do
                components.search(comp)
                components.remove
        ensure
          one_less_comp: components.count = old components.count - 1
        end

replace_component(comp : COMPONENT) is
                -- replace the previous comp with this name
                -- with the new comp
        require
          exists: comp /= void
        do
                from components.start
                until components.item.get_name.is_equal(comp.get_name)
                loop
                        components.forth
                end

                components.replace(comp)
        ensure
          item_changed: components.item = comp
```

```
        end

    get_percent_stream : REAL is
        local  percent, value : REAL
          units   : STRING

            -- calculate the percentage of the stream
            -- that has been defined
        do
            !!units.make(20)

            from     components.start
                     percent := 0.0

            until    components.after
            loop
                     units.copy(components.item.get_units)
                     value := components.item.get_normal

                     if units.is_equal("Vol%%") = true then
                        percent := percent + value
                     elseif units.is_equal("PPM") = true then
                        percent := percent + (value/10000)
                     elseif units.is_equal("PPB") = true then
                        percent := percent + (value/10000000)
                     end

                     components.forth
            end

            Result := percent
        end

    get_component_by_index(i : INTEGER) : COMPONENT is
            -- get a component from the list by its index
        require
          valid_index: i >= 0
          exists: i /= void
        do
            components.go_i_th(i)
            Result := components.item
        end

    get_number_of_components : INTEGER is
            -- get the number of components in the stream
        do
            Result := components.count
        ensure
          valid_count: result >= 0
        end

end -- class STREAM
```

COMPONENT

The Component class models a component of a GC process stream. The attributes of this class are accessed from the COMPONENT_SELECT_DIALOG and COMPONENT_SPEC_DIALOG classes described earlier. The Eiffel source code for this class is:

```
indexing
        description: "The component specific class"

class COMPONENT

creation make

feature -- creation

        make is
                -- initialize the object
        do
                normal := 50.00
                minimum := 0.00
                maximum := 100.00
                measured := true
                !!name.make(80)
                !!units.make(20)

        end
feature {NONE} -- attributes

        name :STRING
                        -- the component name

        units : STRING
                        -- the units for this component

        normal : REAL
                        -- normal concentration of this component

        minimum : REAL
                        -- min concentration of this component

        maximum : REAL
                        -- max concentration of this component

        measured : BOOLEAN
                        -- Is the component measured?


feature -- ACCESS

        set_measured (state : BOOLEAN) is
```

```
                – set the measured state
        require
          exists: state /= void
        do
                measured := state
        ensure
          configured: measured = state
        end

set_normal (t : REAL) is
                -- Set the normal concentration
        require
          exists: t /= void
        do
                normal := t
        ensure
          configured: normal = t
        end

set_minimum (t : REAL) is
                -- Set the minimum concentration
        require
          exists: t /= void
        do
                minimum := t
        ensure
          configured: minimum = t
        end

set_maximum (t : REAL) is
                -- Set the maximum concentration
        require
          exists: t /= void
        do
                maximum := t
        ensure
          configured: maximum = t
        end

set_name (t: STRING) is
                -- set the name for the stream
        require
          exists: t /= void
        do
          name := t
        ensure
          configured: name = t
        end

set_units (t: STRING) is
                -- set the units for the stream
        require
          exists: t /= void
        do
```

```
                units := t
              ensure
                configured: units = t
              end


get_measured : BOOLEAN is
                    -- get the measured state
              do
                    Result := measured
              ensure
                    result = measured
              end

   get_normal : REAL is
                    -- get the normal concentration
              do
                Result := normal
              ensure
                    result = normal
              end

   get_minimum : REAL is
                    -- get the minimum concentration
              do
                    Result := minimum
              ensure
                    result = minimum
              end

   get_maximum : REAL is
                    -- get the maximum concentration
              do
                    Result := maximum
              ensure
                    result = maximum
              end


   get_name : STRING is
                    -- get the name for the stream
              do
                  Result := name
              ensure
                    result = name
              end

   get_units : STRING is
                    -- get the units for the stream
              do
                    Result := units
              ensure
                    result = units
              end
```

```
end -- class COMPONENT
```

4.4 User Interface & Application Object Scenarios

This section presents interesting scenarios that occur between the user interface and application objects. These scenarios illustrate the message passing that takes place between the objects of these classes and are components of the BON Dynamic Model of the expert system.

4.4.1 Creation of the Application Objects

This scenario takes place when a MAIN_WINDOW object is created. The message passing sequence is:

1. MAIN_WINDOW creates the GC_APPLICATION object.
2. The GC_APPLICATION object creates an array of STREAM objects.
3. Each STREAM object creates a linked list of COMPONENT objects.

The BON dynamic diagram for this scenario is:

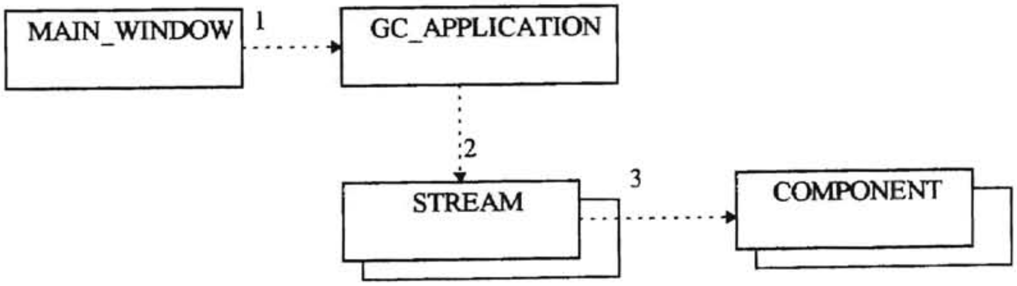


Figure 8 - Creating the Application Objects

4.4.2 Populating the GC_APPLICATION Object

This scenario takes place after the GC_APPLICATION object is created. The message passing sequence is:

1. User chooses New or Open from the File menu.
2. MAIN_WINDOW sends 'Activate' message to APP_SPEC_DIALOG.
3. APP_SPEC_DIALOG sends 'get_app_object' message to MAIN_WINDOW in order to get a copy of the GC_APPLICATION object.
4. APP_SPEC_DIALOG sends 'get_xx' messages to GC_APPLICATION to get its current contents. The current contents are placed in the dialog box edit fields.
5. User sends 'on_ok' message to APP_SPEC_DIALOG.
6. APP_SPEC_DIALOG sends 'set_xx' messages to GC_APPLICATION to populate the object.
7. APP_SPEC_DIALOG sends 'signal_owner' message to itself.
8. APP_SPEC_DIALOG sends 'set_app_object' message to MAIN_WINDOW, passing its copy of the updated GC_APPLICATION object.

The BON dynamic diagram for this scenario is:

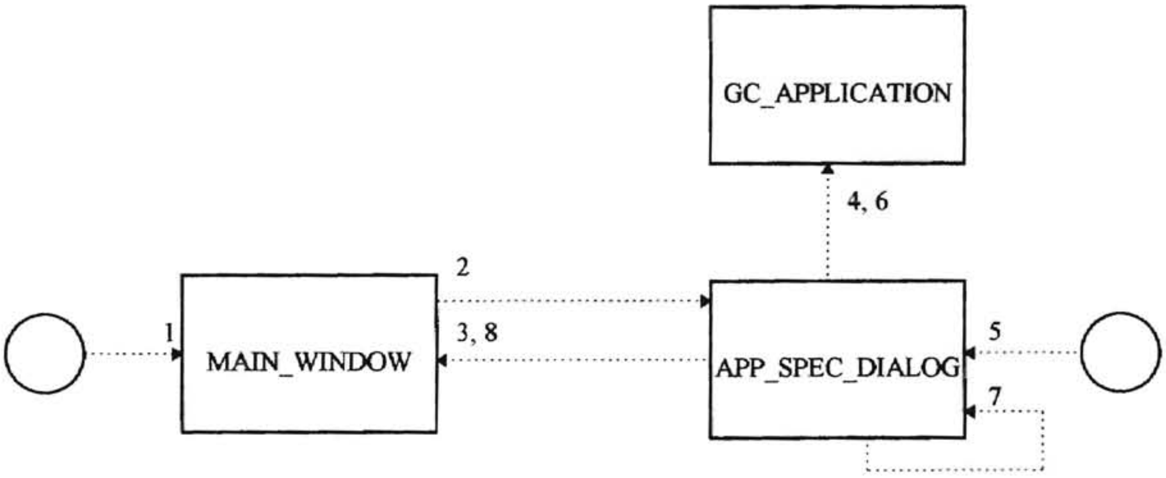


Figure 9 - Populating the GC_APPLICATION Object

4.4.3 Populating a STREAM Object

This scenario can take place at any time after the GC_APPLICATION object is populated. The message passing sequence is:

1. User chooses Stream from the menu.
2. MAIN_WINDOW sends 'Activate' message to STREAM_SPEC_DIALOG.
3. STREAM_SPEC_DIALOG sends 'get_stream_object' message to MAIN_WINDOW in order to get a copy of the STREAM object.
4. MAIN_WINDOW sends a 'get_stream' message to the GC_APPLICATION object, passing the currently accessed stream number to retrieve the STREAM object for STREAM_SPEC_DIALOG.
5. STREAM_SPEC_DIALOG sends 'get_xx' messages to STREAM to get its current contents. The current contents are placed in the dialog edit fields.
6. User sends 'on_ok' message to STREAM_SPEC_DIALOG.
7. STREAM_SPEC_DIALOG sends 'set_xx' messages to STREAM to populate the object.
8. STREAM_SPEC_DIALOG sends 'signal_owner' message to itself.
9. STREAM_SPEC_DIALOG sends 'set_stream_object' message to MAIN_WINDOW, passing its copy of the updated STREAM object.
10. MAIN_WINDOW sends 'set_stream' message to GC_APPLICATION, passing the updated STREAM object.

The BON dynamic diagram for this scenario is:

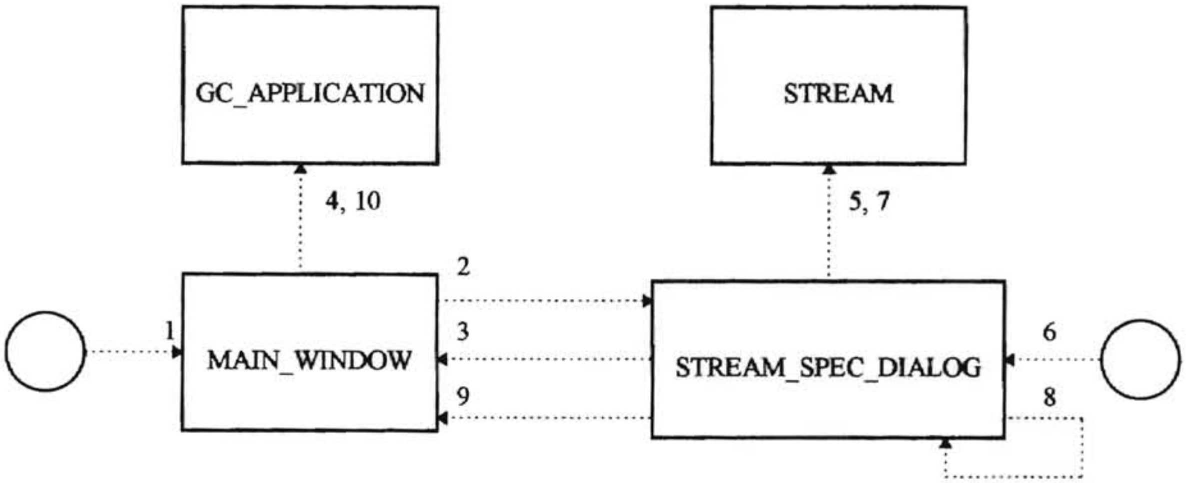


Figure 10 - Populating a STREAM Object

4.4.4 Populating a COMPONENT Object

This scenario can take place at any time after the GC_APPLICATION object is populated. It is the most complex message passing scenario that occurs between the user interface and application objects. The message passing sequence is:

1. User chooses CONFIGURE from the menu.
2. MAIN_WINDOW sends 'Activate' message to COMPONENT_SELECT_DIALOG.
3. COMPONENT_SELECT_DIALOG sends 'get_stream_object' message to MAIN_WINDOW in order to get a copy of the STREAM object.
4. MAIN_WINDOW sends a 'get_stream' message to the GC_APPLICATION object, passing the currently accessed stream number to retrieve the STREAM object for STREAM_SPEC_DIALOG.
5. COMPONENT_SELECT_DIALOG sends 'get_percent_stream' message to STREAM to get its current percentage defined value. This is placed in the % Stream edit field
6. COMPONENT_SELECT_DIALOG sends 'get_component_by_index' messages to STREAM in order to get the names of all of the COMPONENTS of the STREAM. These are placed in the Chosen Components list box.
7. User selects a component name from the Select list box.
8. COMPONENT_SELECT_DIALOG sends 'get_component_by_name' messages to STREAM in order to get a copy of the selected COMPONENT of the STREAM. The COMPONENT is stored in the local component feature.
9. COMPONENT_SELECT_DIALOG sends 'Activate' message to COMPONENT_SPEC_DIALOG.
10. COMPONENT_SPEC_DIALOG sends 'get_component_object' to COMPONENT_SELECT_DIALOG to get a copy its local COMPONENT object
11. COMPONENT_SPEC_DIALOG sends 'get_xx' messages to COMPONENT in order to get its current contents. These are placed in the dialog box edit fields.
12. User sends 'on_ok' message to COMPONENT_SPEC_DIALOG.

13. COMPONENT_SPEC_DIALOG sends 'set_xx' messages to COMPONENT to populate the object.
14. COMPONENT_SPEC_DIALOG sends 'signal_owner' message to itself.
15. COMPONENT_SPEC_DIALOG sends 'update_component_list ' message to COMPONENT_SELECT_DIALOG, passing its copy of the updated COMPONENT object.
16. COMPONENT_SELECT_DIALOG sends 'replace_component' message to STREAM, passing the updated COMPONENT object.
17. User sends 'on_ok' message to COMPONENT_SELECT_DIALOG
18. COMPONENT_SELECT_DIALOG sends 'signal_owner' message to itself.
19. COMPONENT_SELECT_DIALOG sends 'set_stream_object' message to MAIN_WINDOW, passing its copy of the updated STREAM object.
20. MAIN_WINDOW sends 'set_stream' message to GC_APPLICATION, passing the updated STREAM object.

The BON dynamic diagram for this scenario is:

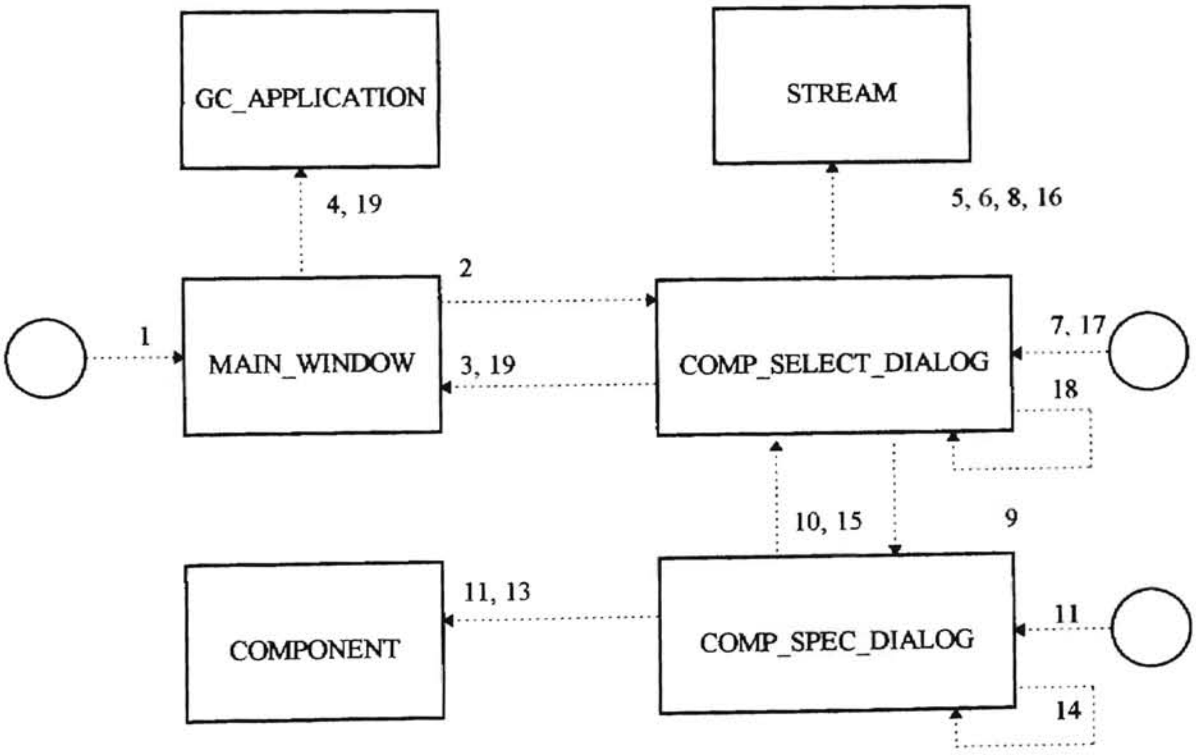


Figure 11 - Populating a COMPONENT Object

4.5 The Legacy Data Base and Supporting Classes

The physical makeup of a GC is modeled by the Application classes which were discussed in the previous section. These data structures contain the properties of the GC a customer wishes to purchase and are used to query a legacy database, called GC_DATABASE, containing historical data on GC configurations that have been successfully fielded by the GC manufacturer. This is a Microsoft Access database and as such can be queried via any ODBC compliant application. By querying the database, a set of possible solutions to the configuration problem can be obtained. The set of possible solutions are then asserted as facts to the inference engine to determine the optimal solution to the configuration problem. This set of possible solutions represents the 'Generate' portion of the reasoning strategy used in the project.

This section discusses the legacy data base used by the expert system. It will describe the relations of the database and discuss the mapping of relational tables to Eiffel objects.

4.5.1 The Relational Schema

The database contains two relations. They are the *Analyzer* and *Components* tables. The *Analyzer* table contains data relating to the configuration of the GC. This table contains attributes such as oven configuration, column type, sample valve type, and carrier gas type. The *Components* table contains attributes related to a chemical component such as component name, and measured concentration. The *Component* table contains an attribute, AnalyzerSerialNumber, which is a foreign key reference to the *Analyzer* table.

The SQL data definition statements for the schema of the database are as follows:

```
CREATE TABLE ANALYZER
(
        ProjectNumber               CHAR,
        AnalyzerSerialNumber        CHAR,
        OVCONFIG                    CHAR,
        SV1TYPE                     CHAR,
        SV1MODEL                    CHAR,
        SV1SIZE                     CHAR,
        SPLITTER                    CHAR,
        METHANATOR                  CHAR,
        VORTEX                      CHAR,
        PROGTEMP                    CHAR,
        ISOTHERZO                   CHAR,
        INITIALTEMP                 CHAR,
        FINALTEMP                   CHAR,
        TEMPRATE                    CHAR,
        DETCT1                      CHAR,
        CARRIERA                    CHAR,
        CARRIERB                    CHAR,
        CYCLETIME                   CHAR,
        APPLNUMB                    CHAR,
        TYPECOLUMN                  CHAR,
        PRIMARY                     CHAR,
        PROCESS                     CHAR,
        COMMENT                     CHAR
) PRIMARY KEY (ProjectNumber)

CREATE TABLE COMPONENTS
(
        AnalyzerSerialNumber        CHAR,
        StreamNumber                INT,
        ComponentName               CHAR,
        Concentration               DOUBLE
) PRIMARY KEY (AnalyzerSerialNumber),
 FOREIGN KEY(AnalyzerSerialNumber) References ANALYZER
```

4.5.2 Mapping the Database to Objects

As mentioned earlier, the Application classes are used to perform queries on the GC_DATABASE in order to generate a set of possible solutions to the configuration problem. The results of these queries are relations that are mapped to Eiffel objects via the facilities of the EiffelStore library. This library is a set of classes that allow an Eiffel application to access an ODBC compliant database. Within the context of the expert system, the DB_WRAPPER classes are used to perform the database access operations.

The BON static architecture for the DB_WRAPPER classes is shown in the following figure.

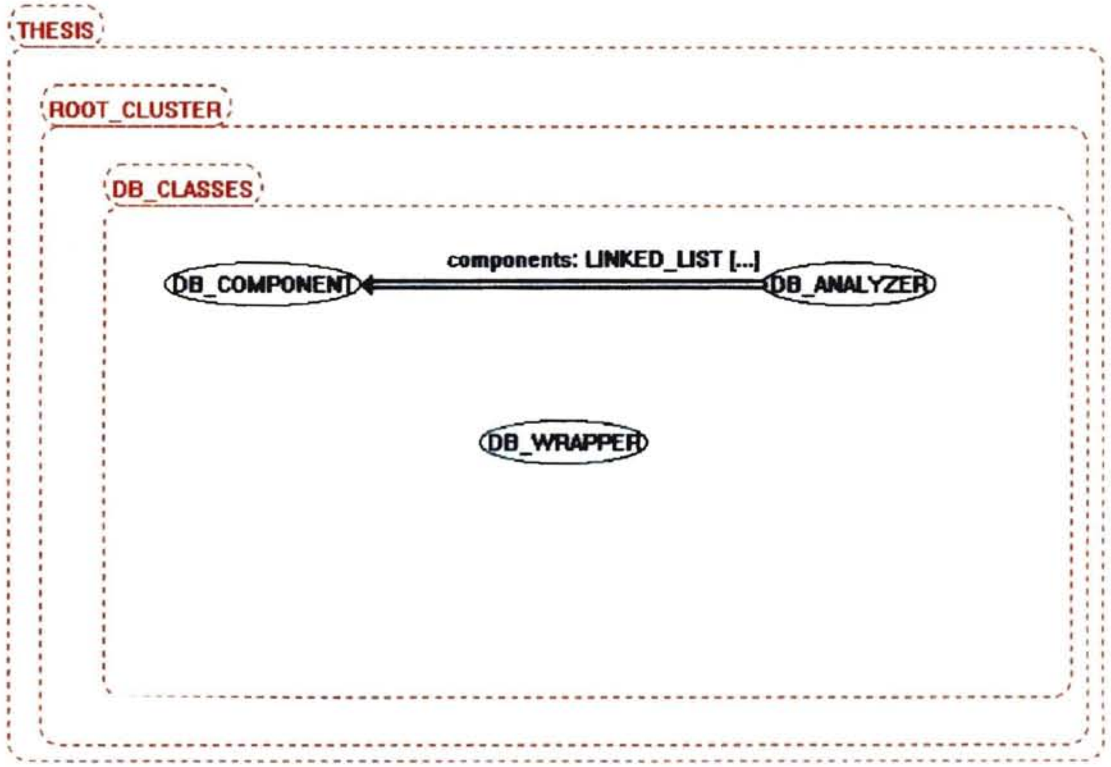


Figure 12 - DB_WRAPPER Classes

The DB_WRAPPER class uses the services of the EiffelStore library classes to perform SELECT queries on the database. The DB_ANALYZER class contains attributes which match those of the *Analyzer* table. It is used to store the results of selection queries on that table. The DB_COMPONENTS class is used to store the results of selection queries on the *Components* table. It has attributes which mimic those of the Components table. The DB_ANALYZER class also contains an attribute of type LINKED_LIST[DB_COMPONENT]. This mirrors the Application classes, where class GC_APPLICATION contains an attribute of type STREAM. STREAM in turn contains a LINKED_LIST[COMPONENT].

The Eiffel source code for the DB_WRAPPER class is:

```
class DB_WRAPPER

inherit

        ACTION

                redefine
                        execute
                end


        RDB_HANDLE

creation

        make

feature {NONE}

        base_selection : DB_SELECTION

        session_control : DB_CONTROL

        myOwner        : MESSAGE_ROUTER

        query_mode     : INTEGER

        AX_QUERY       : INTEGER is 1

        COMP_QUERY     : INTEGER is 2

        index          : INTEGER

feature {MESSAGE_ROUTER}

        make(owner :  MESSAGE_ROUTER) is
                local
                        tmp_string: STRING
                do
                        myOwner := owner
```

```
                set_data_source("GC_DATA_BASE")
                login("admin", "")

                -- Initialization of the Relational Database:
                -- This will set various informations to perform a correct
                -- connection to the Relational database
                set_base

                -- Create useful classes
                -- 'session_control' provides information control access and
                --  the status of the database.
                -- 'base_selection' provides a SELECT query mechanism.
                !! session_control.make
                !! base_selection.make

                -- Start session: establishes connection to database
                session_control.connect
                if not session_control.is_connected then

                        session_control.raise_error
                        -- Something went wrong, and the connection failed
                        io.putstring ("Can't connect to database.%N")

                end
        end

make_analyzer_selection(ax_string: STRING) is
        require
                exists: ax_string /= void
        do
                        -- Query database.
                        base_selection.set_action(Current)
                        base_selection.query (ax_string);
                        query_mode := AX_QUERY
                        -- Iterate through resulting data, and display them
                        base_selection.load_result
        end

make_component_selection(comp_string : STRING) is
        require
                exists: comp_string /= void
        do
                --Query the database
                base_selection.set_action(Current)
                base_selection.query (comp_string)
                query_mode := COMP_QUERY

                -- Iterate through resulting data, and display them
                base_selection.load_result
        end

execute is
        -- gather the query results
        local analyzer : DB_ANALYZER
```

```
                         component: DB_COMPONENT
                    do
                              io.put_string(".")
                              index := index + 1
                              if (index  \\ 60) = 0 then
                                        io.new_line
                              end

                              if query_mode = AX_QUERY then
                                        !! analyzer.make
                                        base_selection.object_convert(analyzer)
                                        base_selection.cursor_to_object
                                        myOwner.add_candidate(analyzer)
                              else
                                        !! component.make
                                        base_selection.object_convert(component)
                                        base_selection.cursor_to_object
                                        myOwner.add_component(component)
                              end
                    end

          analyzer_query(application: GC_APPLICATION) is
                    -- test the interface
                    require
                              exists: application /= void
                    local s             : STREAM
                                c       : COMPONENT
                                n       : REAL
                                i       : INTEGER
                                query   : STRING
                                multiplier: REAL
                    do

                      !!query.make(1024)

                      --begin making the select query
                      s := application.get_stream(0)
                      query.copy("SELECT Analyzer.* FROM Analyzer INNER JOIN components ON
Analyzer.AnalyzerSerialNumber = components.AnalyzerSerialNumber WHERE ")

                      --add the component names/concentrations to the query
                      from  i := 0
                      until i = s.get_number_of_components
                      loop
                                        -- component name stuff
                                        query.append("((components.ComponentName= '")
                                        c := s.get_component_by_index(i + 1)
                                        query.append(c.get_name)
                                        query.append("') AND " )

                                        -- component concentration stuff
                                        -- scale concentration based upon unit
                                        if c.get_units.is_equal("PPM") then
                                                  multiplier := .000001
```

```
                                elseif c.get_units.is_equal("PPB") then
                                        multiplier := .000000001
                                else
                                        multiplier := 1.0
                                end

                                query.append("(components.Concentration > ")
                                query.append(((c.get_minimum) * multiplier).out)
                                query.append(" AND components.Concentration < ")
                                query.append(((c.get_maximum) * multiplier).out)
                                query.append(")) ")

                                if i < s.get_number_of_components - 1 then
                                        query.append(" OR ");
                                end

                                i := i + 1
                        end

                        query.append(";")
                        make_analyzer_selection(query)
                end

        component_query(ax_string : string) is
                        -- create a query for each candiate to fill its components
                        require
                        exists: ax_string /= void

                        local query : STRING
                        do
                                !!query.make(120)
                                query.copy("Select * from components where analyzerserialnumber =
'")

                                query.append(ax_string)
                                query.append("';");
                                make_component_selection(query)
                        end


        close_session is
        -- terminate the session
        do
                        session_control.disconnect
        end

end -- class DB_WRAPPER
```

DB_WRAPPER contains attributes of type DB_CONTROL and DB_SELECTION. These are EiffelStore library classes that allows an application to connect to an query an ODBC database. A more detailed discussion of EiffelStore is given by Meyer in [22].

DB_WRAPPER is used to implement a two phase approach to querying the GC_DATABASE. The approach is:

1. Determine which *Analyzer* tuples are referenced by *Component* tuples which have a concentration within the range of the Component objects of the Application classes. Map the results of this query to DB_ANALYZER objects. The sql statement for this query is of the form:

   SELECT Analyzer. * *FROM Analyzer INNER JOIN components ON*
   Analyzer.AnalyzerSerialNumber = components.AnalyzerSerialNumber WHERE
   components.componentname = 'name of first component of application' OR
   components.componentname = 'name of second component of application' ...

2. For each DB_ANALYZER object obtained in step 1), select all *Components* of the *Analyzer*. Map the results to DB_COMPONENT objects. The sql statement for this query is of the form:

   *SELECT * FROM components WHERE analyzerserialnumber where analyzerserialnumber = 'my serial number'*

The feature 'analyzer_query' is used to create the sql statement which implements step 1) while 'component_query' creates the sql for step 2). These features create the sql string, register the query with EiffelStore, and then initiate the query. EiffelStore will then call the feature 'execute' for each returned tuple of the query and map the resultant tuple to an Eiffel object. These features are called by the MESSAGE_ROUTER object. MESSAGE_ROUTER will be discussed later in the paper. The resultant object is then added to the list of candidate solutions through the 'add_candidate' and 'add_conponent' features of MESSAGE_ROUTER.

The Eiffel source code for DB_ANALYZER is:

```
indexing
        description: "The root of a candidate solution"

class DB_ANALYZER
inherit

        ANY
                redefine
                        out
                end


creation make

feature -- creation

        make is
                -- initialize the db_object
        do
                !!components.make
                !!projectnumber.make(80)
                !!analyzerserialnumber.make(80)
                !!ovconfig.make(80)
                !!sv1type.make(80)
                !!sv1model.make(80)
                !!sv1size.make(80)
                !!isotherzo.make(80)
                !!initaltemp.make(80)
                !!finaltemp.make(80)
                !!temprate.make(80)
                !!detct1.make(80)
                !!carriera.make(80)
                !!carrierb.make(80)
                !!cycletime.make(80)
                !!typecolumn.make(80)
                !!applnumb.make(80)
                !!process.make(80)
                !!comment.make(80)
        end

feature {NONE}

        projectnumber: STRING

        analyzerserialnumber: STRING

        ovconfig: STRING

        sv1type: STRING
```

```
	sv1model: STRING

	sv1size: STRING

	splitter: BOOLEAN

	methanator: BOOLEAN

	vortex: BOOLEAN

	progtemp: BOOLEAN

	isotherzo: STRING

	initaltemp: STRING

	finaltemp: STRING

	temprate: STRING

	detct1: STRING

	carriera: STRING

	carrierb: STRING

	cycletime: STRING

	applnumb: STRING

	typecolumn: STRING

	primary: CHARACTER

	process: STRING

	comment: STRING

	components : LINKED_LIST[DB_COMPONENT]

feature

	out: STRING is
			-- Display contents
		do
			!! Result.make (100)
			if projectnumber /= Void then
				Result.append ("project:")
				Result.append (projectnumber)
				Result.extend ('%N')
			end
			if analyzerserialnumber /= Void then
				Result.append ("SerialNumber:")
				Result.append (analyzerserialnumber)
```

```
                        Result.extend ('%N')
                end
                Result.extend ('%N')
        end


    set_projectnumber (t: STRING) is
                -- Set `projectnumber' with `t'
        require
                argument_exists: not (t = Void)
        do
                projectnumber := t
        ensure
                projectnumber = t
        end

    set_analyzerserialnumber (t: STRING) is
                -- Set `analyzerserialnumber' with `t'
        require
                argument_exists: not (t = Void)
        do
                analyzerserialnumber := t
        ensure
                analyzerserialnumber = t
        end

    set_ovconfig (t: STRING) is
                -- Set `ovconfig' with `t'
        require
                argument_exists: not (t = Void)
        do
                ovconfig := t
        ensure
                ovconfig = t
        end

set_sv1type (t: STRING) is
                -- Set `sv1type' with `t'
        require
                argument_exists: not (t = Void)
        do
                sv1type := t
        ensure
                sv1type = t
        end

      set_sv1model (t: STRING) is
                -- Set `sv1model' with `t'
        require
                argument_exists: not (t = Void)
        do
                sv1model := t
        ensure
                sv1model = t
```

```
        end

set_svlsize (t: STRING) is
                – Set `svlsize' with `t'
        require
                argument_exists: not (t = Void)
        do
                svlsize := t
        ensure
                svlsize = t
        end


        set_splitter (t: BOOLEAN) is
                -- Set `splitter' with `t'
        require
                argument_exists: not (t = Void)
        do
                splitter := t
        ensure
                splitter = t
        end

set_methanator(t: BOOLEAN) is
                – set `methanator' with `t'
        require
                argument_exists: not (t = Void)
        do
                methanator := t
        ensure
                methanator = t
        end

set_vortex(t: BOOLEAN) is
                -- set `vortex' with `t'
        require
                argument_exists: not (t = Void)
        do
                vortex := t
        ensure
                vortex = t
        end

set_progtemp(t: BOOLEAN) is
                – set `progtemp' with `t'
        require
                argument_exists: not (t = Void)
        do
                progtemp := t
        ensure
                progtemp = t
        end

set_isotherzo (t: STRING) is
```

```
                        -- Set `isotherzo' with `t'
            require
                        argument_exists: not (t = Void)
            do
                        isotherzo := t
            ensure
                        isotherzo = t
            end

    set_initaltemp (t: STRING) is
                        -- Set `initialtemp' with `t'
            require
                        argument_exists: not (t = Void)
            do
                        initaltemp := t
            ensure
                        initaltemp = t
            end

    set_finaltemp (t: STRING) is
                        -- Set `finaltemp' with `t'
            require
                        argument_exists: not (t = Void)
            do
                        finaltemp := t
            ensure
                        finaltemp = t
            end

    set_temprate (t: STRING) is
                        -- Set `temprate' with `t'
            require
                        argument_exists: not (t = Void)
            do
                        temprate := t
            ensure
                        temprate = t
            end

    set_detct1 (t: STRING) is
                        -- Set `detct1' with `t'
            require
                        argument_exists: not (t = Void)
            do
                        detct1 := t
            ensure
                        detct1 = t
            end

    set_carriera (t: STRING) is
                        -- Set `carriera' with `t'
            require
                        argument_exists: not (t = Void)
            do
```

```
                        carriera := t
            ensure
                        carriera = t
            end

set_carrierb (t: STRING) is
                        – Set `carrierb' with `t'
            require
                        argument_exists: not (t = Void)
            do
                        carrierb := t
            ensure
                        carrierb = t
            end

set_cycletime (t: STRING) is
                        -- Set `cycletime' with `t'
            require
                        argument_exists: not (t = Void)
            do
               cycletime := t
            ensure
                        cycletime = t
            end

set_applnumb (t: STRING) is
                        -- Set `applnumb' with `t'
            require
                        argument_exists: not (t = Void)
            do
               applnumb := t
            ensure
                        applnumb = t
            end


set_typecolumn (t: STRING) is
                        -- Set `typecolumn' with `t'
            require
                        argument_exists: not (t = Void)
            do
               typecolumn := t
            ensure
               typecolumn = t
            end

set_primary (t: CHARACTER) is
                        -- Set `primary' with `t'
            require
                        argument_exists: not (t = Void)
            do
               primary := t
            ensure
               primary = t
```

```
                end

        set_process (t: STRING) is
                        -- Set `process' with `t'
                require
                        argument_exists: not (t = Void)
                do
                   process := t
                ensure
                   process = t
                end

        set_comment (t: STRING) is
                        -- Set `comment' with `t'
                require
                        argument_exists: not (t = Void)
                do
                   comment := t
                ensure
                   comment = t
                end



        add_component(comp : DB_COMPONENT) is
                        -- add a component to the list
                do

                        if not components.has(comp) then
                                components.extend(comp)
                        end
                end


end -- class DB_ANALYZER
```

The Eiffel source for DB_COMPONENT is:

```
class DB_COMPONENT

creation make

feature  -- creation

        make is
                -- initialize the db_object
        do
                !!analyzerserialnumber.make(80)
                !!componentname.make(80)
                init_strings
        end
```

```
feature  {NONE}

         analyzerserialnumber: STRING

         streamnumber: INTEGER

         componentname: STRING

         concentration: DOUBLE

         measurement: DOUBLE

feature


         set_streamnumber (t: INTEGER) is
                          -- Set `streamnumber' with `t'
                 require
                          argument_exists: not (t = Void)
                 do
                          streamnumber := t
                 ensure
                          streamnumber = t
                 end

         set_concentration (t: DOUBLE) is
                          -- Set `concentration' with `t'
                 require
                          argument_exists: not (t = Void)
                 do
                          concentration := t
                 ensure
                          concentration = t
                 end

         set_measurement (t: DOUBLE) is
                          -- Set `measurement' with `t'
                 require
                          argument_exists: not (t = Void)
                 do
                          measurement := t
                 ensure
                          measurement = t
                 end


         set_analyzerserialnumber (t: STRING) is
                          -- Set `analyzerserialnumber' with `t'
                 require
                          argument_exists: not (t = Void)
                 do
                          analyzerserialnumber := t
                 ensure
                          analyzerserialnumber = t
```

```
		end

	set_componentname (t: STRING) is
				-- Set `componentname' with `t'
		require
				argument_exists: not (t = Void)
		do
				componentname := t
		ensure
				componentname = t
		end

	init_strings is
	do
		analyzerserialnumber.copy("None")
		componentname.copy("None")
	end

end -- class DB_COMPONENT
```

4.6 *The CLIPS Wrapper Classes*

Once the legacy database has been queried, generating a set of possible solutions to the configuration problem, the results must be asserted as facts to the CLIPS inference engine for the reasoning session to continue. The mechanisms for asserting the facts, running the inference engine, and reporting results of the reasoning session reside in the CLIPS Wrapper Classes. These are Eiffel classes which interface to the CLIPS DLL.

The BON static architecture for these classes is shown in the figure below.

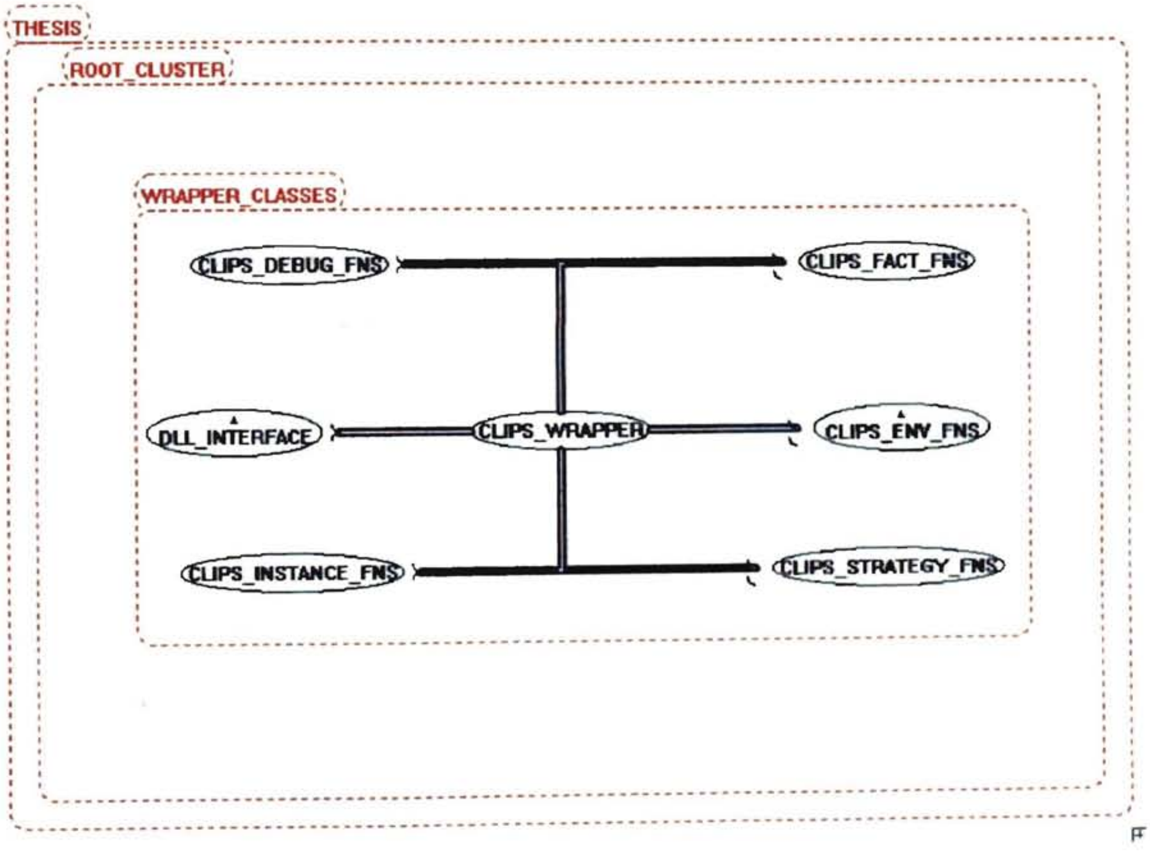


Figure 13 - CLIPS Wrapper Classes

The CLIPS_WRAPPER class is the main interface to the CLIPS DLL. It contains an instance of each of the other wrapper classes and other features that implement high level operations on the inference engine. The features of CLIPS_WRAPPER are used by the MESSAGE_ROUTER.

The Eiffel source code for the CLIPS_WRAPPER is:

```
indexing
        description:
                "Provides the interface to the CLIPSDLL"

class CLIPS_WRAPPER

inherit

        SHARED_LIBRARY_CONSTANTS

creation
  make

feature -- Initialization

        make is
                -- create the wrapper objects
          do
                        !!interface.make
                        !!env.make
                        !!deb.make
                        !!facts.make
                        !!ins.make
                        !!strat.make
                        initialized := false
                end

feature {NONE} -- Implementation

        deb : CLIPS_DEBUG_FNS
        facts : CLIPS_FACT_FNS
        env: CLIPS_ENV_FNS
        ins: CLIPS_INSTANCE_FNS
        strat : CLIPS_STRATEGY_FNS
        interface : DLL_INTERFACE
        initialized : BOOLEAN
        rule_file : STRING is "c:\school\thesis\project\system.clp"
        fact_file : STRING is "c:\school\thesis\project\facts.clp"
```

```
feature {MESSAGE_ROUTER}

        clear is
                -- clear the CLIPS environment
                do
                        env.clear
                end

        load_rule_file is
                -- load the rule file
                do
                        env.load(rule_file)
                end


        start_clips is
                -- start up clips
                do
                        if initialized /= true then
                                env.initialize
                                env.add_route
                                initialized := true
                        end
                end

        load_fact_file is
                -- load the facts file
                do
                        env.load(fact_file)
                end

        run is
                -- run a inference session
                local r : INTEGER
                do
                 env.reset
                 env.run
          end

end -- class CLIPS_WRAPPER
```

The features of CLIPS_WRAPPER represent the minimum requirements necessary to control a reasoning session. At the present time, only the facilities of the CLIPS_ENV_FNS class is utilized by the CLIPS_WRAPPER. The CLIPS_ENV_FNS class contains features that configure the CLIPS DLL. These include:

1. Loading rules into CLIPS.
2. Loading or asserting facts.

3. Clearing the contents of working memory and the rule base.
4. Resetting the contents of working memory.
5. Running the inference engine.

The other wrapper classes each implement a subset of the CLIPS API. The Eiffel source code for these classes will not be presented now. Instead the Eiffel short form for each of these classes is presented in the appendices of this paper.

CLIPS_WRAPPER also contains an attribute of type DLL_INTERFACE. This class is used to implement a callback procedure used by CLIPS to report the results of the reasoning session. The manner in which the callback is implemented is very interesting in that the DLL_INTERFACE is called indirectly by an external C routine, not by CLIPS. The C routine implements the callback and is called by CLIPS. This routine in turn, sends a message to the DLL_INTERFACE. This scenario will be discussed in detail in a moment.

The Eiffel source code for the DLL_INTERFACE is:

```
indexing
        description:
                "Provides the interface to the CLIPSDLL%
                %print router."

class DLL_INTERFACE


creation
   make

feature -- Initialization

        make is
          do
                        !!message.make(80)

                        get_obj(current)
                        get_proc($CLIPS_PRINTER)
          end
```

```
		message : STRING

		get_proc(function : POINTER) is
				external
						"C"
				end

		get_obj(obj : ANY) is
				external
						"C"
				end

		CLIPS_PRINTER(char_pointer : POINTER) is
				-- this feature is called indirectly by the clips dll
		do
				message.from_c(char_pointer)
				io.put_string(message.out)
		end

end -- class DLL_INTERFACE
```

The external C routines referenced in DLL_INTERFACE are implemented as follows:

```
/***********************************************
*                                              *
*                                              *
*            C Calls Used to interface         *
*            with the CLIPS DLL                *
*                                              *
***********************************************/


#include <stdlib.h>
#include "c:\clips\source\clips\clips.h"
#include "c:\eiffel3\bench\spec\w32msc\include\eiffel.h"

int _declspec (dllexport) printFunction(char *, char *);
int _declspec (dllexport) queryFunction(char *);
int add_route(void);
void get_proc(EIF_PROC);
void get_obj(EIF_REFERENCE);
void run(void);

int add_route()
{
        AddRouter("print", 20, queryFunction, printFunction,
                                NULL, NULL, NULL);
        return (CLIPS_TRUE);
}

void run()
{
        Run(-1);
}

int _declspec (dllexport) queryFunction(char * logicalName)
{
        if (strcmp(logicalName, "stdout") == 0)
                return (CLIPS_TRUE);

        return (CLIPS_FALSE);
}

EIF_REFERENCE eiffel_messenger;
EIF_PROC        c_clips_printer;

void get_proc(EIF_PROC p)
{
        c_clips_printer = p;
}

void get_obj(EIF_REFERENCE e)
{
        eiffel_messenger = eif_adopt(e);
```

```
}

int _declspec (dllexport) printFunction(char * logicalName, char * str)
{
        if (strlen(str) > 0)
        {
                str[strlen(str) + 1] = '\0';
                (c_clips_printer)(eif_access(eiffel_messenger), (EIF_REFERENCE) str);
        }

        return (CLIPS_TRUE);
}
```

When a DLL_INTERFACE object is created, it passes a reference to itself and it's CLIPS_PRINTER feature to the external C code. The printFunction() of the external C code uses these references to signal the DLL_INTERFACE object when CLIPS needs to report the results of a reasoning session. The BON dynamic model for the callback mechanism is:

1. CLIPS_WRAPPER creates DLL_INTERFACE
2. DLL_INTERFACE sends 'get_obj' and 'get_proc' message to external c code.
3. CLIPS_WRAPPER sends 'run' message to CLIPS_ENV_FNS.
4. CLIPS_DLL sends 'printFunction' message to external c code whenever it needs to perform an output operation.
5. External c code sends 'CLIPS_PRINTER' message to DLL_INTERFACE.

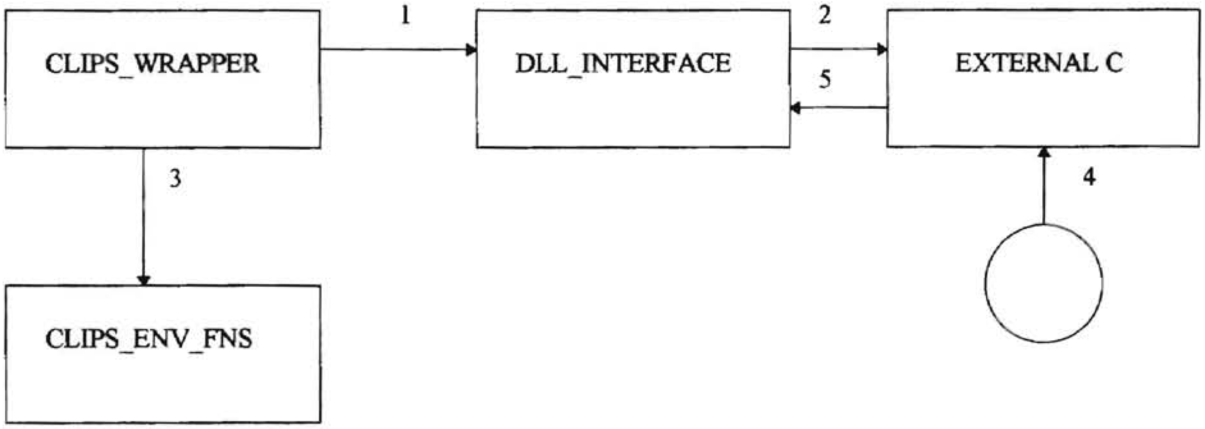


Figure 14 - BON Dynamic Model For Callback Mechanism

4.7 The Knowledge Base

The results of the database queries are asserted as facts to the CLIPS inference engine. These facts are then caused to fire according to the rules defined in the system's knowledge base. The knowledge base contains a set of rules that implement the "match" reasoning strategy discussed earlier in the paper as implemented to solve the GC configuration problem.

The knowledge base consists of rules to solve the various stages of the configuration process. These stages are:

1. Determining the detector make up of the solution.
2. Determining the carrier gas type of the solution.
3. Determining the analyzer that most closely matches the target application.
4. Reporting the results of the reasoning session.

Each of these steps will now be discussed in detail.

4.7.1 Flow Control

The knowledge base contains a set of rules that are responsible for allowing the reasoning session to proceed in an orderly and deterministic manner. These rules help to implement the flow control mechanism that is characteristic of the "Match" reasoning strategy.

As an example, here is one of the flow control rules that is present in the knowledge base:

```
(defrule determine-detectors
        (declare (salience -10))
        ?phase <- (phase detectors)
        =>
        (retract ?phase)
        (assert (phase carrier_gas)))
```

This particular rule will file only if the current reasoning phase is the detector phase and no other rules are ready to fire. This is due to the use of the 'salience' operator which controls the priority of a rule. A salience of -10 is used to set the flow control rules to the lowest priority of the rules on the firing stack. The rule then fires and changes the execution mode to the carrier_gas determination phase. Flow control rules like this exist in the knowledge base to control execution through each of the following phases of reasoning.

4.7.2 Determining The Detectors

The first phase of the reasoning session is concerned with determining the correct detectors for the target application. The following hueristics are implemented in the knowledge base:

- If the target application contains component concentrations in the percent level, then a TCD detector is appropriate.
- If the target application contains inert components, then a TCD detector is appropriate.
- If the target application contains trace level (concentrations in the ppm range) components that are not inert, and not Sulfer, then a FID detector is appropriate.
- If the target application contains a trace level (concentrations in the ppm range) component that is Sulfer, then a FPD detector is appropriate.
- If the target application contains trace level (concentrations in the ppm range) components that are inert, then a TCD detector is appropriate.

In the form of CLIPS rules, these huristics are represented as:

```
; If an inert component is present
; and its concentration level is trace
; assert a trace-inert fact

(defrule trace-inert-present
        (phase detectors)
        (components (name ?n&Helium|Nitrogen) (normal ?c&:(< ?c .000009)) (measured true))
=>
        (assert (trace-inert)))

; If an inert component is present
; and its concentration level is trace
; assert an inert fact

(defrule inert-present
        (phase detectors)
        (components (name ?n&Helium|Nitrogen) (normal ?c&:(> ?c .000009)) (measured true))
```

```
=>
        (assert (inert)))


; If all components are not trace level
; or an insert fact is present
; or a trace-insert fact is present
; choose a TCD detector

(defrule TCD
        (phase detectors)
        (or
                (forall
                        (components (normal ?concentration) (measured true))
                        (test (> ?concentration .00009))
                )
                (inert)
                (trace-inert)
        )
=>
        (assert (chosen-detector TCD))
        (printout t "detector is TCD" crlf))

; If all components are trace level
; and an inert fact is not present
; and a trace-insert fact is not present
; choose a FID detector

(defrule FID
        (phase detectors)
        (forall
                (components (normal ?concentration) (measured true))
                (test (< ?concentration .0001))
        )
        (not (trace))
        (not (trace-inert))
 =>
        (assert (chosen-detector FID))
        (printout t "detector is FID" crlf))

; If a component with the name Sulfer is present
; and the concentration is trace level
; choose a FPD detector

(defrule FPD
        (phase detectors)
        (exists
                (components (normal ?concentration) (name Sulfer) (measured true))
                        (test (< ?concentration .0001))
        )
=>
        (assert (chosen-detector FPD))
        (printout t "detector is FPD" crlf))
```

4.7.3 Determining The Carrier Gas

The following hueristics are used to determine the proper carrier gas for the target application:

- If the solution contains a TCD or FID detector and the customer is in North America, then use Helium as the carrier gas.
- If the solution contains a TCD or FID detector and the customer is not in North America, then use Hydrogen as the carrier gas.
- If the target application contains Hydrogen or Helium as a measured component, then use Nitrogen as the carrier gas.
- If the solution contains a FPD detector and the target application contains no measured Hydrogen or Helium, then use Helium as the carrier gas.
- If the solution contains a FPD detector and the target application contains measured Hydrogen or Helium, then use Nitrogen as the carrier gas.

In the form of CLIPS rules, these huristics are represented as:

```
; If a component with the name Hydrogen exists
; assert a has-hydrogen face

(defrule has-hydrogen
        (phase carrier_gas)
        (exists
                (components (name ?n&Hydrogen))
        )
=>
        (assert (has-hydrogen)))

; If a component with the name Helium exists
; assert a has-helium face

(defrule has-helium
        (phase carrier_gas)
        (exists
                (components (name ?n&Helium))
        )
=>
        (assert (has-helium)))

; If the chosen detector is FPD
; and no helium or hydrogen is present
```

```
; or
; If the customer is located in North America
; and the chosen detector is FID or TCD
; and no helium or hydrogen is present
; select Helium as the carrier gas

(defrule Carrier-Gas-Helium
        (phase carrier_gas)
        (or
                (and
                        (applications (customer_location NA))
                        (or (chosen-detector FID) (chosen-detector TCD))
                        (not (has-helium))
                        (not (has-hydrogen))
                )
                (and
                        (chosen-detector FPD)
                        (not (has-helium))
                        (not (has-hydrogen))
                )
        )
=>
        (assert (chosen-carrier-gas Helium))
        (printout t "carrier gas is Helium" crlf))

; If the customer is not located in North America
; and the chosen detector is FID or TCD
; and no helium or hydrogen is present
; select Hydrogen as the carrier gas

(defrule Carrier-Gas-Hydrogen
        (phase carrier_gas)
        (and
                (applications (customer_location ?loc&~NA))
                (or (chosen-detector FID) (chosen-detector TCD))
                (not (has-helium))
                (not (has-hydrogen))
        )
=>
        (assert (chosen-carrier-gas Hydrogen))
        (printout t "carrier gas is Hydrogen" crlf))

; If helium or hydrogen is present
; select Nitrogen as the carrier gas

(defrule Carrier-Gas-Nitrogen
        (phase carrier_gas)
        (or
                (has-helium)
                (has-hydrogen)
        )
=>
        (assert (chosen-carrier-gas Nitrogen))
        (printout t "carrier gas is Nitrogen" crlf))
```

4.7.4 Choosing the Correct Analyzer from the Candidate Solutions

The choice of the candidate solution to the target application is highly dependent on the concentration of the components in the process stream. The hueristics used to select the analyzer are:

- Find the smallest component in the target application.
- Find the largest component in the target application.
- If the set of candidate solutions contains an analyzer with same smallest and largest components as the target application, and these are at the same concentration, then this candidate solution is a perfect match for the application.
- If the set of candidate solutions contains an analyzer with same smallest and largest components as the target application, and their concentrations are within a factor of two of the target concentrations, then this candidate solution is a match for the application.
- If the set of candidate solutions contains an analyzer with same smallest component, and another analyzer with the same largest component as the target application, and the concentrations are within a factor of two of the target concentrations, then a combination of these candidate solutions may be a match for the application.
- If the set of candidate solutions contains an analyzer with same smallest component as the target application, and the concentration is within a factor of two of the target concentration, then this candidate solution is a partial match for the application.
- Otherwise, no matches exist.

In the form of CLIPS rules, these huristics are represented as:

```
; Find the component of the target application with
; the smallest concentration. Store this into a
; 'smallest-component' fact

(defrule find-smallest-component
        (phase component)
        ?ac <- (components (name ?n) (normal ?c2) (measured true))
        ?sc <- (smallest-component (concentration ?c))
        (test (> ?c ?c2))
=>
        (modify ?sc (name ?n) (concentration ?c2))
)
```

```
; Find the component of the target application with
; the largest concentration.  Store this into a
; 'largest-component' fact

(defrule find-largest-component
        (phase component)
        ?ac <- (components (name ?n) (normal ?c2) (measured true))
        ?lc <- (largest-component (concentration ?c))

        (test (< ?c ?c2))
=>
        (modify ?lc (name ?n) (concentration ?c2))
)

; Check for a perfect match

(defrule perfect_match
        (declare (salience 10))
        (phase component)
        (largest-component (name ?lc_name) (concentration ?lc_conc))
        (smallest-component (name ?sc_name) (concentration ?sc_conc))
        ?acl <- (actual_component (analyzerserialnumber ?sn) (componentname ?lc_name)
                        (concentration ?lc_conc) (measurement ~0))
        ?acs <- (actual_component (analyzerserialnumber ?sn) (componentname ?sc_name)
                        (concentration ?sc_conc) (measurement ~0))
        ?ccs <- (closest-component-small)
        ?ccl <- (closest-component-large)
=>
        (modify ?ccs (sn ?sn) (name ?sc_name) (delta 0.0))
        (modify ?ccl (sn ?sn) (name ?lc_name) (delta 0.0))
        (retract ?acl)
        (retract ?acs)
)

; Find the component within the solution space with
; the same name as the 'smallest-component' such that
; it's concentration is closest to that of the
; 'smallest-component'.  Store this into a 'closest-component-small'
; fact.
(defrule find-closest-analyzer-small
        (phase component)
        (smallest-component (name ?n) (concentration ?c))
        ?ac <- (actual_component (analyzerserialnumber ?serial)(componentname ?n)
                        (concentration ?c2) (measurement ~0))
        ?cc <- (closest-component-small (concentration ?c3) (delta ?c4))
        (test (> ?c4 (abs(- ?c ?c2))))
=>
        (modify ?cc (sn ?serial) (name ?n) (concentration ?c2) (delta (abs(- ?c ?c2))))
)

; Find the component within the solution space with
; the same name as the 'largest-component' such that
; it's concentration is closest to that of the
```

```
; 'largest-component'.  Store this into a 'closest-component-large'
; fact.
(defrule find-closest-analyzer-large
        (phase component)
        (largest-component (name ?n) (concentration ?c))
        ?ac <- (actual_component (analyzerserialnumber ?serial)(componentname ?n)
                        (concentration ?c2) (measurement ~0))
        ?cc <- (closest-component-large (concentration ?c3) (delta ?c4))
        (test (> ?c4 (abs(- ?c ?c2))))
=>
        (modify ?cc (sn ?serial) (name ?n) (concentration ?c2) (delta (abs(- ?c ?c2))))
)

;If the 'closest-component-small' is within a factor of 2
; of the smallest component then we have a match.

(defrule within-range-small
        (phase component)
        (closest-component-small (sn ?s) (concentration ?c))
        (smallest-component (concentration ?c2))
        ?ax <- (analyzer (analyzerserialnumber ?s))
        (and
                (test (> ?c (/ ?c2 2)))
          (test (< ?c (* ?c2 2)))
        )
=>
        (assert (within-range-small))
)

; If the 'closest-component-large' is within a factor of 2
; of the largest component then we have a match.


(defrule within-range-large
        (phase component)
        (closest-component-large (sn ?s) (concentration ?c))
        (largest-component (concentration ?c2))
        ?ax <- (analyzer (analyzerserialnumber ?s))
        (and
                (test (> ?c (/ ?c2 2)))
          (test (< ?c (* ?c2 2)))
        )
=>
        (assert (within-range-large))
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
; Solution rules
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; If the closest components are within range and
; the closest components are from the same analyzer
; then we have a perfect match

(defrule finish-up-with-perfect-solution
        ?p <-  (phase solution)
        (closest-component-small (sn ?s) (delta 0.0))
        (closest-component-large (sn ?s) (delta 0.0))
=>
        (retract ?p)
        (printout t "Analyzer SN: " ?s " matches application perfectly" crlf))

; If the closest components are within range but
; the closest components are from the different analyzers
; then we have a imperfect match

(defrule finish-up-with-partial-solution
        ?p <-  (phase solution)
        (closest-component-small (sn ?s))
        (closest-component-large (sn ?s2&~?s))
        (within-range-large)
        (within-range-small)
=>
        (retract ?p)
        (printout t "Analyzer SN: " ?s " matches smallest component" crlf)
        (printout t "Analyzer SN: " ?s2 " matches largest component" crlf)
        (printout t "A combination of these analyzers may match the application" crlf))

; If one of the components matches

(defrule finish-up-with-partial-solution-small
        ?p <-  (phase solution)
        (closest-component-small (sn ?s))
        (not (exists (within-range-large)))
        (within-range-small)
=>
        (retract ?p)
        (printout t "Analyzer SN: " ?s " matches smallest component" crlf)
        (printout t "No match for the largest component" crlf)
        (printout t "Application is only partially matched" crlf))

(defrule finish-up-with-partial-solution-large
        ?p <-  (phase solution)
        (closest-component-large (sn ?s))
        (not (exists (within-range-small)))
```

```
        (within-range-large)
=>
        (retract ?p)
        (printout t "Analyzer SN: " ?s " matches largest component" crlf)
        (printout t "No match for the smallest component" crlf)
        (printout t "Application is only partially matched" crlf))

; If an analyzer does not exist that contains the
; closest component, then we do not have a match.

(defrule finish-up-with-nosolution
        ?p <-  (phase solution)
        (not (exists(within-range-large)))
        (not (exists(within-range-small)))
=>
        (retract ?p)
        (printout t "Application has no match." crlf))
```

4.8 The Message Router, Rules File, and Facts File

The Message Router class is responsible for interfacing between the other classes of the system in order to:

1. Pass data from the GC_APPLICATION class to CLIPS.
2. Query the database.
3. Pass data from the DB_WRAPPER classes to CLIPS.

This is accomplished by storing the contents of the GC_APPLICATION and DB_ANALYZER objects in an ASCII file called the 'facts file'. This representation was chosen for two reasons. First, it allowed to file to be run from within the CLIPS interactive environment. Second, having the data stored persistently aided in the debugging of the application. The 'assert_application' feature is used to create the file and write the GC_APPLICATION data to the file and the 'assert_candidates' feature appends the DB_ANALYZER data to the file.

The Eiffel source code for this class is:

```
indexing
        description:
                "Routes Messages to ODBC and CLIPS"

class MESSAGE_ROUTER

creation
   make

feature {MAIN_WINDOW} -- Initialization

        make is
                -- create the router objects
          do
                        !!clips.make
                        !!db.make(Current)
                end

        start_session(application : GC_APPLICATION) is
                -- begin a CLIPS session
                require
                        app_exists : application /= void
```

```
        do

            !!candidates.make           -- reinitialize data structures
            clips.start_clips
            clips.load_rule_file

            io.put_string("Querying Database")
            io.new_line
             db.analyzer_query(application)

            from candidates.start
            until candidates.after
            loop
                db.component_query(candidates.item.analyzerserialnumber)
                candidates.forth
            end

            io.put_string("Query Complete")
            io.new_line

            io.put_string("Building Facts File")
            io.new_line

            assert_application(application)
            assert_candidates
            io.put_string("Asserting Facts")
            io.new_line

            clips.load_fact_file

            io.put_string("Running CLIPS")
            io.new_line

            clips.run
            clips.clear
        end


feature {NONE} -- Attributes and private features

    clips       : CLIPS_WRAPPER
    db              : DB_WRAPPER
    candidates  : LINKED_LIST[DB_ANALYZER]
    file        : FACTFILE
    fact_file   : STRING is "c:\school\thesis\project\facts.clp"

    assert_application(application: GC_APPLICATION) is
            -- the target application facts are written to the facts file
    require
            exists : application /= void
            local fact_string               : STRING
                comp                            : COMPONENT
                str                             : STREAM
                str_index, comp_index           : INTEGER
```

```
                        multiplier                    : REAL
do
                !!fact_string.make(255)
                !!comp.make
                !!str.make
                !!file.make_open_write(fact_file)

                -- construct the application facts
                fact_string.copy("(deffacts applications ")
                file.to_fact_file(fact_string, FALSE)

                -- construct the single application fact
                fact_string.copy("(applications (customer_name ")
                fact_string.append(application.customer_name)
                fact_string.append(")")
                file.to_fact_file(fact_string, FALSE)

                fact_string.copy("(customer_location ")
                fact_string.append(application.customer_location)
                fact_string.append(")")
                file.to_fact_file(fact_string, FALSE)

                fact_string.copy("(number_streams ")
                fact_string.append(application.number_streams.out)
                fact_string.append(")")
                file.to_fact_file(fact_string, FALSE)

                fact_string.copy("(cycle_time ")
                fact_string.append(application.cycle_time.out)
                fact_string.append(")")
                file.to_fact_file(fact_string, FALSE)

                fact_string.copy("(carrier_gas ")
                fact_string.append(application.carrier_gas)
                fact_string.append("))")
                file.to_fact_file(fact_string, FALSE)

                -- create the component and stream facts
                from str_index     := 0
                until str_index = application.number_streams
                loop
                        -- create the stream facts
                        str := application.get_stream(str_index)
                        fact_string.copy("(stream (tag ")
                        fact_string.append(str_index.out)
                        fact_string.append(")")
                        file.to_fact_file(fact_string, FALSE)

                        fact_string.copy("(corrosive ")
                        fact_string.append(str.get_corrosive.out)
                        fact_string.append(")")
                        file.to_fact_file(fact_string, FALSE)

                        fact_string.copy("(dis_solids ")
```

```
fact_string.append(str.get_disolids.out)
fact_string.append(")")
file.to_fact_file(fact_string, FALSE)

fact_string.copy("(polimer ")
fact_string.append(str.get_polimer.out)
fact_string.append(")")
file.to_fact_file(fact_string, FALSE)

fact_string.copy("(temperature ")
fact_string.append(str.get_temperature.out)
fact_string.append(")")
file.to_fact_file(fact_string, FALSE)

fact_string.copy("(pH ")
fact_string.append(str.get_ph.out)
fact_string.append(")")
file.to_fact_file(fact_string, FALSE)

fact_string.copy("(s_pressure ")
fact_string.append(str.get_spress.out)
fact_string.append(")")
file.to_fact_file(fact_string, FALSE)

fact_string.copy("(r_pressure ")
fact_string.append(str.get_rpress.out)
fact_string.append(")")
file.to_fact_file(fact_string, FALSE)

fact_string.copy("(phase ")
fact_string.append(str.get_phase)
fact_string.append("))")
file.to_fact_file(fact_string, FALSE)

str_index := str_index + 1

from  comp_index := 1
until comp_index > str.get_number_of_components
loop
        comp := str.get_component_by_index(comp_index)
        fact_string.copy("(components (tag ")
        fact_string.append(str_index.out)
        fact_string.append(")")
        file.to_fact_file(fact_string, FALSE)

  fact_string.copy("(name ")
        fact_string.append(comp.get_name)
        fact_string.append(")")
        file.to_fact_file(fact_string, FALSE)

        -- scale concentration based upon unit
        if comp.get_units.is_equal("PPM") then
                multiplier := .000001
        elseif comp.get_units.is_equal("PPB") then
```

```
                    multiplier := .000000001
                else
                    multiplier := 1.0
                end

                fact_string.copy("(normal ")
                fact_string.append((comp.get_normal * multiplier).out)
                fact_string.append(")")
                file.to_fact_file(fact_string, FALSE)

                fact_string.copy("(measured ")
                fact_string.append(comp.get_measured.out)
                fact_string.append("))")
                file.to_fact_file(fact_string, FALSE)

                comp_index := comp_index + 1
            end
        end

        -- terminate the deffacts
        fact_string.copy(")")
        file.to_fact_file(fact_string, FALSE)
    end

assert_candidates       is
        -- the candidate facts are written to the facts file
        local fact_string           : STRING
        comp                        : LINKED_LIST[DB_COMPONENT]
        do
        !!fact_string.make(255)

        -- construct the application facts
        fact_string.copy("(deffacts candidates ")
        file.to_fact_file(fact_string, FALSE)

        -- create the db_analyzer and db_component facts
        from  candidates.start
        until candidates.after
        loop
            -- create the analyzer facts
            fact_string.copy("(analyzer (projectnumber ")
            fact_string.append(candidates.item.projectnumber)
            fact_string.append(")")
            file.to_fact_file(fact_string, FALSE)

            fact_string.copy("(analyzerserialnumber ")
            fact_string.append(candidates.item.analyzerserialnumber)
            fact_string.append("))")
            file.to_fact_file(fact_string, FALSE)

            comp := candidates.item.components
            from  comp.start
            until comp.after
            loop
```

```
                    fact_string.copy("(actual_component (analyzerserialnumber
")
                    fact_string.append(comp.item.analyzerserialnumber)
                    fact_string.append(")")
                    file.to_fact_file(fact_string, FALSE)

                    fact_string.copy("(componentname ")
                    fact_string.append(comp.item.componentname)
                    fact_string.append(")")
                    file.to_fact_file(fact_string, FALSE)

                    fact_string.copy("(concentration ")
                    fact_string.append(comp.item.concentration.out)
                    fact_string.append(")")
                    file.to_fact_file(fact_string, FALSE)

                    fact_string.copy("(measurement ")
                    fact_string.append(comp.item.measurement.out)
                    fact_string.append("))")
                    file.to_fact_file(fact_string, FALSE)

                    comp.forth
                end
                candidates.forth
            end

            -- terminate the deffacts
            fact_string.copy(")")
            file.to_fact_file(fact_string, TRUE)
        end
```

The 'start_session' feature is called when the user selects 'GO' from the MAIN_WINDOW menu. This is the feature that starts a reasoning session by calling features of the DB_WRAPPER class to:

1. Query the database, taking the results of the query and writing them to an ASCII file called the 'facts' file.
2. Call features of the CLIPS_WRAPPER to assert the knowledge base and facts file.
3. Start the reasoning session, and report back the results.

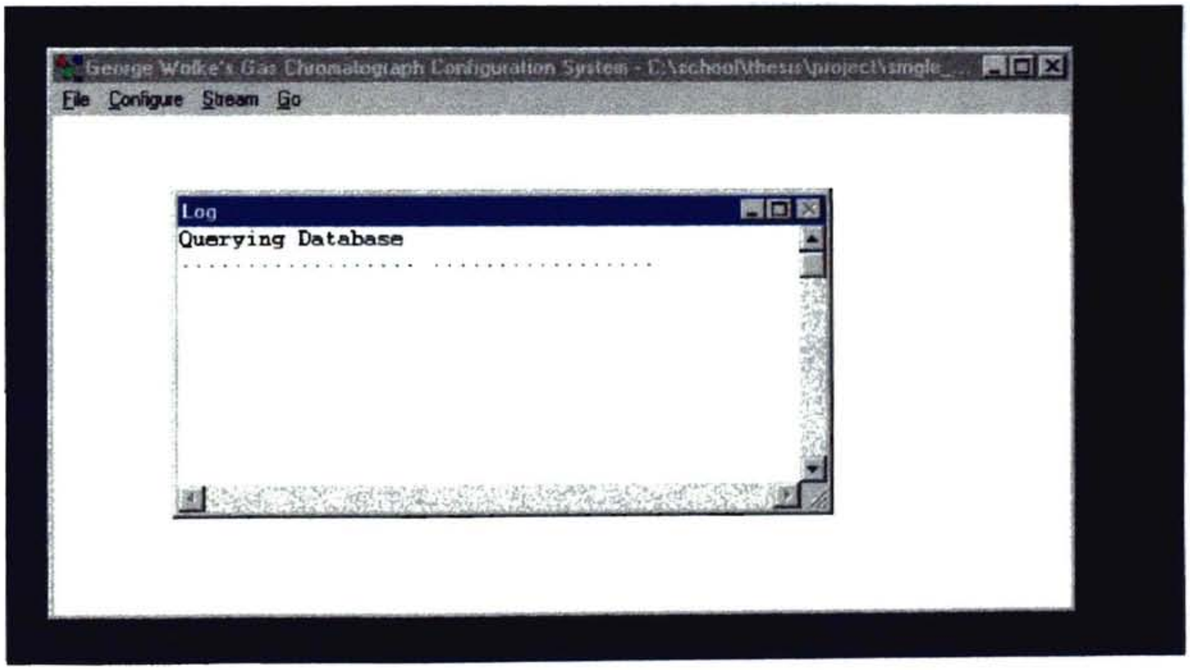
The facts file contains the results of the database queries and the target application data in a format that can be understood by the CLIPS inference engine. The facts are stored according to the following fact templates:

```
; Target application (what we are solving for)

(deftemplate applications "The target application"
        (multislot customer_name)           ; the customers name
        (slot customer_location)            ; the customers country
        (slot number_streams)               ; the number of streams
        (slot     cycle_time)               ; application cycle time
        (slot carrier_gas)                  ; the carrier gas desired
)

(deftemplate stream "Applications contain streams"
        (slot tag)                          ; the tag for this stream
        (slot corrosive)                    ; is the stream corrosive
        (slot dis_solids)                   ; does the stream have disolved solids
        (slot polimer)                      ; does the stream polimerize
        (slot temperature)                  ; The stream temperature
        (slot pH)                           ; The stream pH
        (slot s_pressure)                   ; The stream pressure
        (slot     r_pressure)               ; The return pressure
        (slot phase)                        ; The phase of the gas in this stream
)

(deftemplate components "Streams contain components"
        (slot tag)                          ; the tag for this stream
        (slot name)                         ; the component name
        (slot normal)                       ; normnal concentration of this component
        (slot measured)                     ; Is the component measured?
)

; These templates are for the generated solution set

(deftemplate analyzer       "An actual GC"
        (slot projectnumber)                ; Where this analyzer was used
        (slot analyzerserialnumber)         ; Identifier for this analyzer
        (slot ovconfig)                     ; oven configuration
        (slot sv1type)                      ; type of first sample valve
        (slot sv1model)                     ; model of first sample valve
        (slot sv1size)                      ; size of first sample valve
        (slot detct1)                       ; type of first detector
        (multislot carriera)                ; type of first carrier gas
        (multislot cycletime)               ; actual cycle time
        (multislot typecolumn)              ; the type of column
)

(deftemplate actual_component      "A component of an analyzer"
        (slot analyzerserialnumber)         ; binds the component to an analyzer
        (slot     streamnumber)             ; analyzers contain more than 1 stream
        (slot componentname)                ; the name of this component
        (slot concentration)                ; normal concentration for this component
        (slot measurement)                  ; is this component measured
)
```

Again, the 'assert_application' feature creates the facts file and writes templates of type application, stream, and component to the file. These templates are filled with GC_APPLICATION, STREAM, and COMPONENT data respectively. The 'assert_candidates' feature writes templates of type analyzer, and actual_component. These templates are filled with DB_ANALYZER and DB_COMPONENT objects respectively.

4.9 Running the System

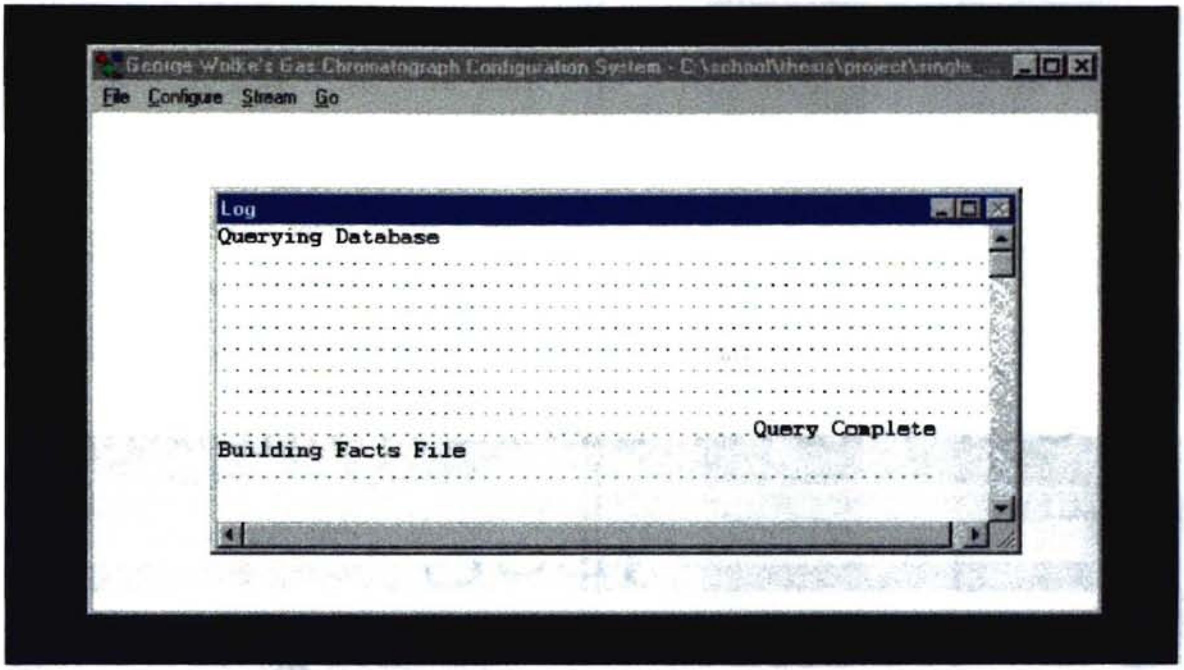
We are finally in a position to examine the output of running the system. In order to run the system, the user will select the 'Go' menu item from the main menu. At this point the system will query the database and display the output shown in the figure below.



```
Querying Database
.................. ..................
```

Figure 15 - Database Query Output

When the database query is complete the system will begin building the facts file and the output will be as shown in the figure on the following page.



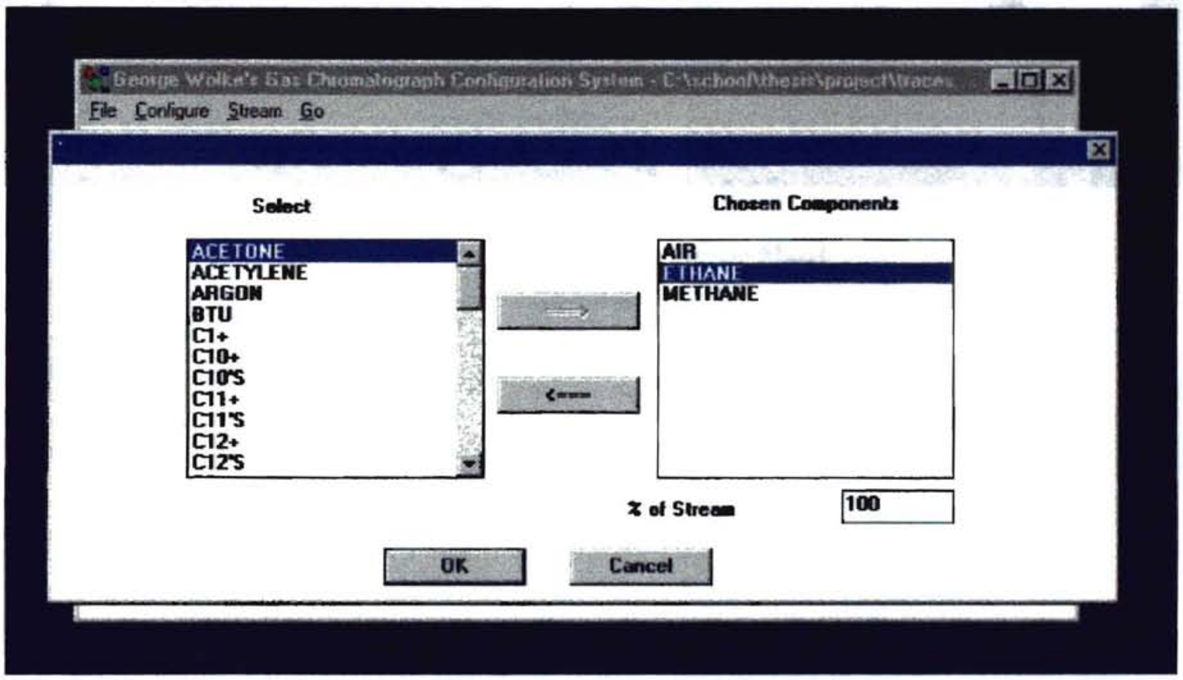
```
Querying Database
........................................................................
........................................................................
........................................................................
........................................................................
........................................................................
........................................................................
........................................................................
........................................................................
..................................................Query Complete
Building Facts File
...................................................................
```

Figure 16 - Database Query Output

After the facts file is created, the facts are asserted to CLIPS, and a reasoning session started. Consider the process stream shown in the figure below, with normal concentrations:

- Air: 99.999%
- Ethane: 4.5 ppm
- Methane: 5.5 ppm

This process stream represents a case where no match may be found.



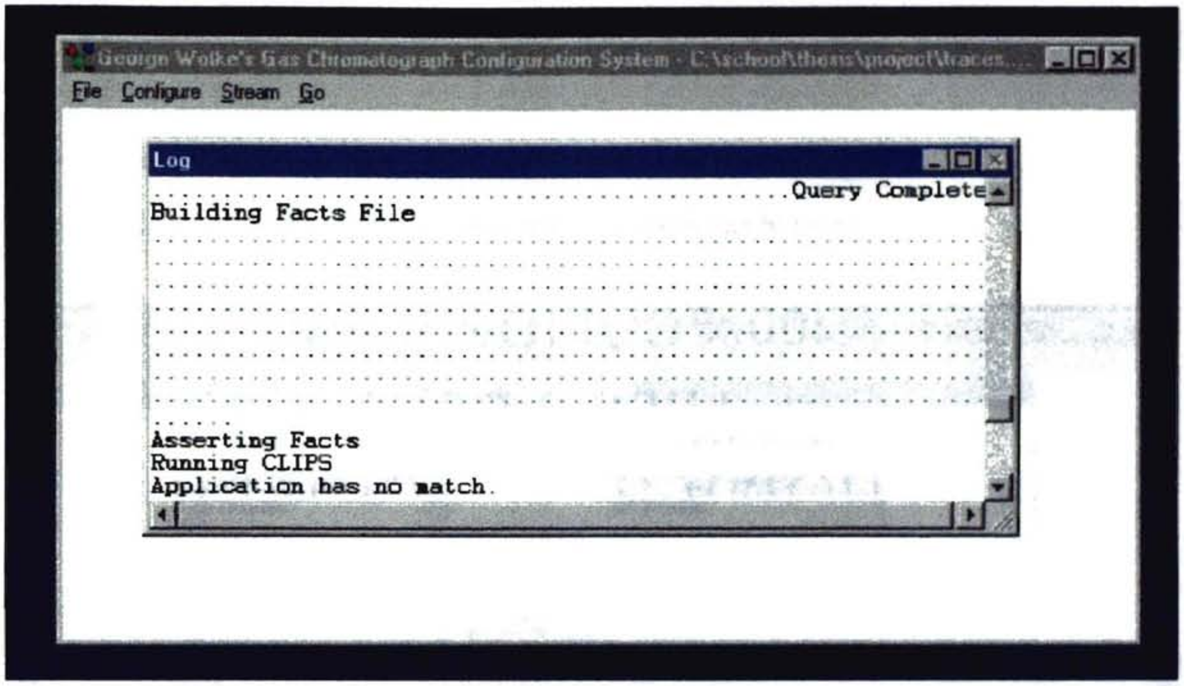
ACETONE
ACETYLENE
ARGON
BTU
C1+
C10+
C10'S
C11+
C11'S
C12+
C12'S

AIR
ETHANE
METHANE

% of Stream 100

Figure 16 - Process Stream Yielding No Match

The system output using the process stream shown above is given in the following figure.



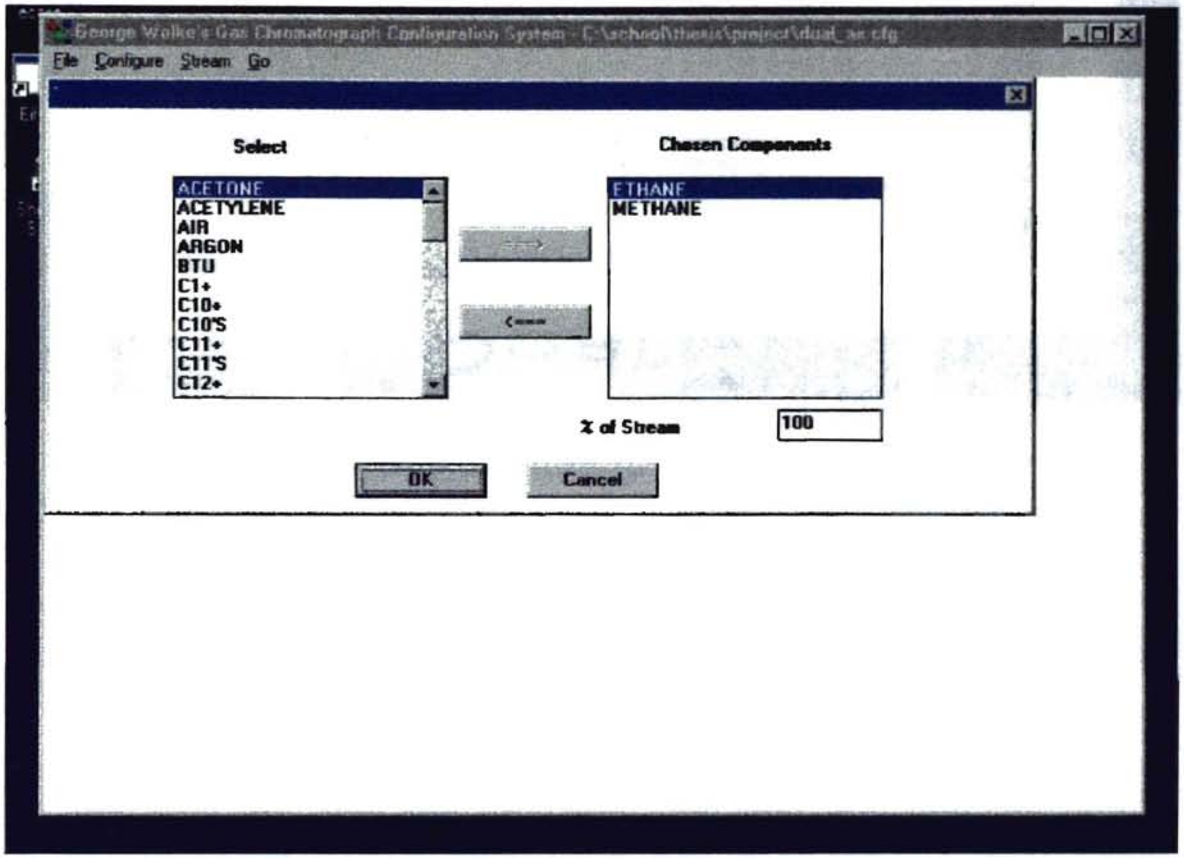
```
...........................................Query Complete
Building Facts File
.........................................................
.........................................................
.........................................................
.........................................................
.........................................................
.........................................................
.........................................................
.........................................................
......
Asserting Facts
Running CLIPS
Application has no match.
```

Figure 17 - System Output For No Match

Consider the process stream shown in the figure below, with normal concentrations:

- Ethane = 91%
- Methane = 9%

This process stream represent a case where an imperfect match may be found.

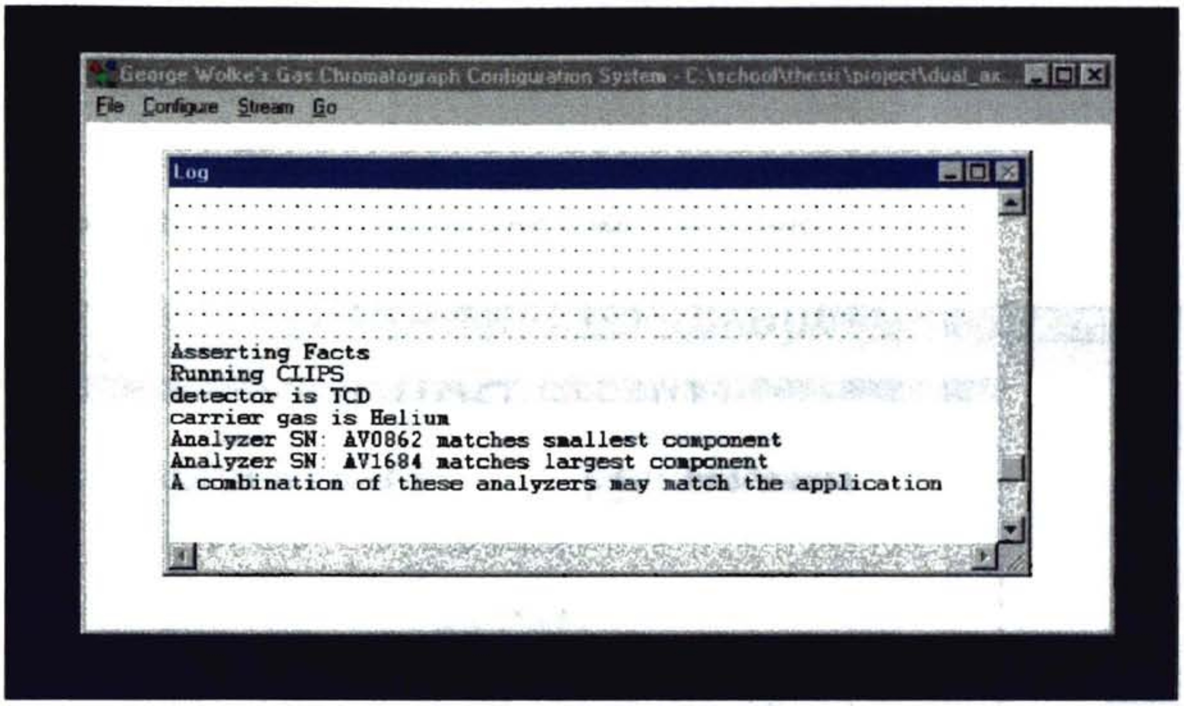


Chosen Components
ETHANE
METHANE

% of Stream 100

Figure 18 - Process Stream Yielding an Imperfect Match

The system output using the process stream shown above is given in the following figure.



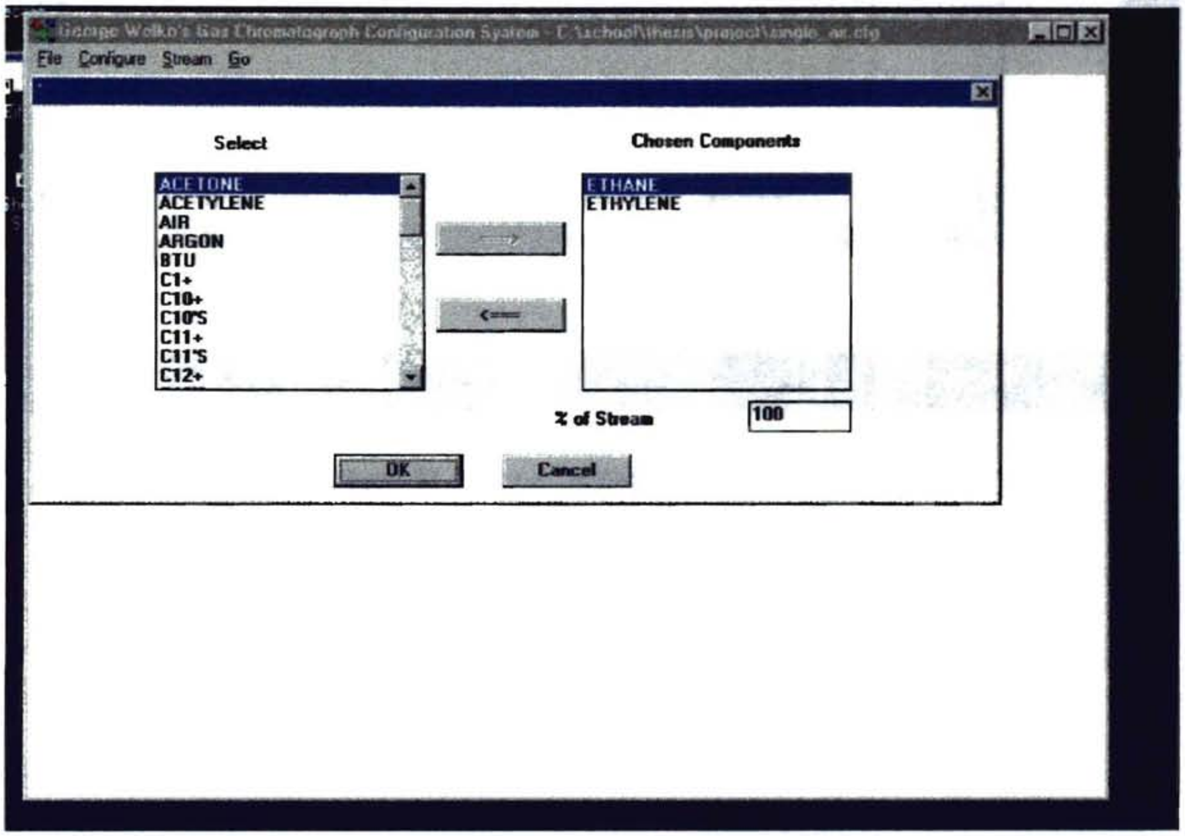
```
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
Asserting Facts
Running CLIPS
detector is TCD
carrier gas is Helium
Analyzer SN: AV0862 matches smallest component
Analyzer SN: AV1684 matches largest component
A combination of these analyzers may match the application
```

Figure 19 - System Output For an Imperfect Match

Finally, consider the process stream shown in the figure below, with normal concentrations:

- Ethane = 99%
- Ethylene = 1%

This process stream represent a case where a perfect match may be found.

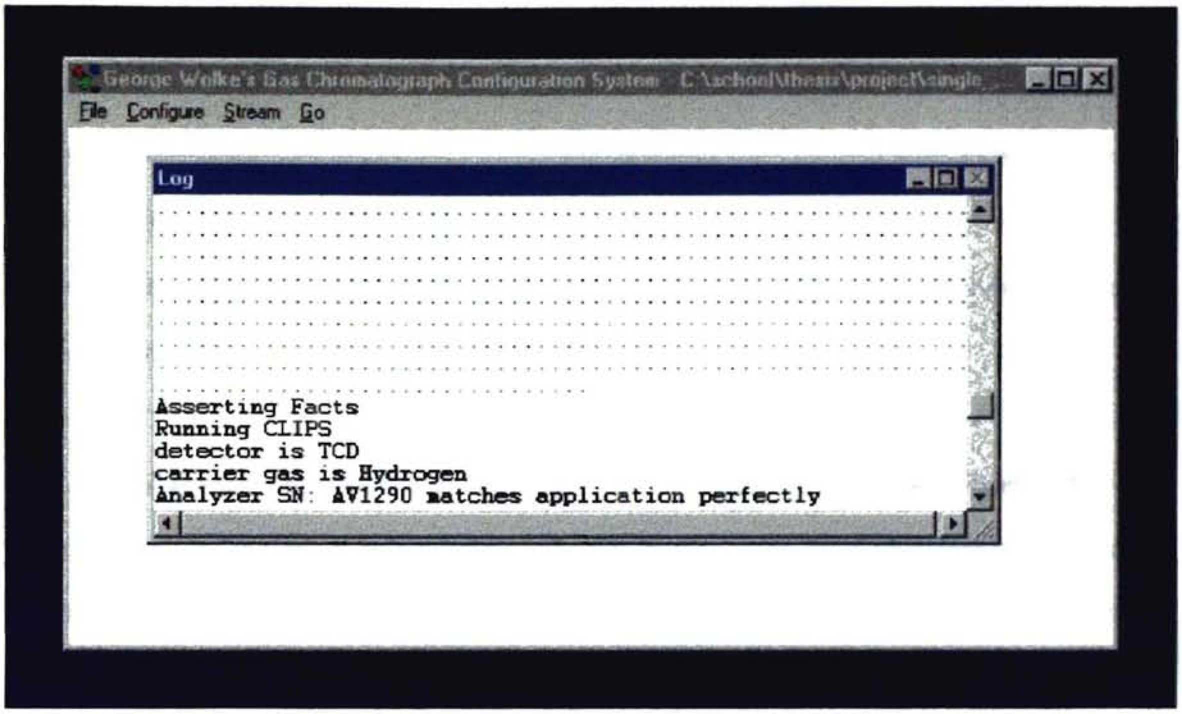


Chosen Components
ETHANE
ETHYLENE

% of Stream 100

Figure 20 - Process Stream Yielding a Perfect Match

The system output using the process stream shown above is given in the following figure.



```
....................................................................
....................................................................
....................................................................
....................................................................
....................................................................
....................................................................
....................................................................
....................................................................
....................................
Asserting Facts
Running CLIPS
detector is TCD
carrier gas is Hydrogen
Analyzer SN: AV1290 matches application perfectly
```

Figure 21 - System Output For Perfect Match

5.0 Conclusions

This section will discuss the conclusions I've reached concerning the work that made up this project. To begin, I believe that the results of my project have met the goals described in my thesis proposal. I believe that I have succeeded in designing and implementing a rule based expert system that solves the configuration problem, as was detailed earlier in this paper.

Next, I would like to discuss the positive impact that the Eiffel programming language (and the BON design method) made in the success of the project. I began work on the program in May of 1996 and completed work in November of 1996. This equates to a 6 + calendar month development cycle. The project consists of 28 application specific classes. During final testing of the project I was unable to make the system crash. For a project of this complexity, that says a lot.

I made extensive use of the BON method throughout the development process. I believe that the program's robustness is due to the use of BON and the robust static architecture it allowed me to develop through 'design by contract'. As is evident in my source listings, I made use of pre- and post-conditions whenever they made sense. These contracts allowed me to discover logical flaws in my program early in the development process. This most certainly decreased the overall development time of the project. Eiffel's strong type checking made it close to impossible to introduce the type of bugs that would cause system crashes due to type mismatches and misplaced polymorphism. This limited my bugs to those that were semantic in nature, rather than syntactic. I'm sure that if C++ or Smalltalk had been used as the target language, I would not have seen the benefits listed above and the resulting program would not be as robust.

Eiffel's ability to interface to C in a clear and efficient manner can not be overstated. For this project I used software from three different vendors. These pieces of the puzzle had a single common denominator, an interface to C. Because I was able to write Eiffel wrappers that easily interfaced to these different software packages I was able to glue them together into a single solid software solution to the configuration problem.. For these reasons I would recommend Eiffel as a great choice for a course in programming languages and object oriented programming.

The reasoning strategy used in the project is hybrid in nature. The data persented earlier in the paper show that this approach is indeed sound and can be used to implement an expert system. Relational database queries are used to 'Generate' a set of possible solutions to the problem. These are passed onto an inference engine to determine if a viable solution to the problem exists. Therefore, the overall performance of the system is tied to the ability to perform these queries in a timely manner. I must admit that the performance of the EiffelStore libraries is less than ideal. For this reason it is important to trade off the generality of the database queries versus the execution time of the program.

Lastly, I want to comment on the development of my knowledge base. This was by far the most difficult part of my project as it required me to interface with persons not familiar with software development, or software in general. Also, it required me to enter a problem domain that I was not familiar with. A great deal of time was spent up front, between the Chemists and myself, discussing and clarifying jargon so that we had a set of terminology all understood .

I feel as though I underestimated the level of effort required for this task. The good news is that my experience has shown me that it is easy to 'scale up' a knowledge base if it is designed with 'scaleability' in mind. The knowledge base used for this project was designed to be scaleable and as a result I was easily able to create a base knowledge base that solved the simple cases of this project. The knowledge base may be enhanced in the future to handle more sophisticated cases without having to start again from scratch. I would recommend that a knowledge base developer pay particular attention to its' design in order to ensure it can be scaled-up as required.

In closing I want to acknowledge that this was a very challenging project. It certainly wasn't easy and, for the most part, occupied all of my off work time for 6+ months. But I wouldn't trade it for any other project of lesser scope. This project allowed me to design and build an industrial strength application. This is an experience that any graduate level student should certainly have prior to entering the real world of software development.

Bibliography

1. Barker, V.E., O'Connor D.

Expert Systems for Configuration at Digital: XCON and Beyond,

Communications of the ACM, 32 (3), 298 - 318, March 1989

2. Barr, A., Cohen, P.A., Feigenbaum, E.A.

The Handbook of Artifical Intelligence, Volume II.

Kaufmann, Los Angeles, CA, 1981, 1982

3. Barr, A., Cohen, P.A., Feigenbaum, E.A.

The Handbook of Artifical Intelligence, Volume IV.

Addision-Wesley, Reading, MA, 1989

4. Brownston, L., Farrell, R., Kant, E., Martin, N.

Programming Expert Systems in OP5.

Addison-Wesley, Reading, MA,1985

5. Buchanan, B.G., Feigenbaum, E.A.

DENDRAL and Meta-DENDRAL: Their Applications Dimension. Artifical Intelligence, 11, 5-24, 1978

6. Forgy, C.L

Rete: A Fast Algorithm For the Many Pattern, Many Object Pattern Match Problem. Artificial Intelligence, 19, 17-37, 1982

7. Forsyth, R. (1984)

Expert Systems - Principles and Case Studies

Chapman and Hill Computing

8. Giarratano, J., Riley, G.

Expert Systems Principles and Programming

PWS-Kent Publishing Company, Boston 1989

9. Jackson, Peter

Introduction to Expert Systems

Addison Wesley Co, Reading, MA., 1986

10. Johnson Space Center.

CLIPS Reference Manual: Volume 1 Basic Programming Guide

JSC-25012, 1993

11. Johnson Space Center.

CLIPS Reference Manual: Volume 2 Advanced Programming Guide

JSC-25012, 1993

12. Kline, P., Dolins, S.

Designing Expert Systems - A Guide to Implementation Techniques

John Wiley and Sons, NY 1989

13. McDermott,J.

R1: A Rule Based Configurer of Computer Systems. Artificial Intelligence, 19, 39-88, 1982

14. Meyer, B.

Reusable Software: The Base Object-Oriented Component Libraries.

Interactive Software Engineering, Goltea, CA, 1995

15. Meyer, B.

ISE Eiffel: The Environment.

Interactive Software Engineering, Goltea, CA, 1993

16. Meyer, B.

Object-oriented Software Construction.

Prentice Hall, NY, NY, 1988

17. Schlatter, U. Real-Time Knowledge-Based Support For Air Traffic Management.

IEEE Expert, 2(3), 21 -24 1994

18. Walden, K., Nerson, Jean-Marc.

Seamless Object-oriented Software Architecture.

Prentice Hall, NY, NY, 1995

19. Waterman, D.A, Hayes-Roth, F.

Pattern Directed Inference Systems.

Academic Press, NY, 177-199, 1978

20. Poole, Colin F., Schuette, Sheila A.

Contemporary Practice of Chromatography.

Elsevier, Amsterdam, 1984

21. Sund, W.E.

Process Gas Chromatography Fundamentals.

Applied Automation Inc., Bartlesville, OK, 1987

22. Meyer, B.

ISE EiffelStore Library Manual.

Interactive Software Engineering, Goltea, CA, 1996

Appendices

Appendix A - BON System Chart

system_chart *APP*
cluster *APP_CLASSES*
cluster *DB_CLASSES*
cluster *ROOT_CLUSTER*
cluster *ROUTER*
cluster *THESIS*
cluster *WRAPPER_CLASSES*

Appendix B - BON Cluster Charts

cluster_chart *APP*
cluster *THESIS*

cluster_chart *APP_CLASSES*
class *GC_APPLICATION*
description
"The main application class"

class *COMPONENT*
description
"The component specific class"

class *STREAM*
description
"The application stream"

cluster_chart *DB_CLASSES*
class *DB_ANALYZER*
description
"The root of a candidate solution"

class *DB_COMPONENT*
description
"The components of a candidate solution"

class *DB_WRAPPER*
description
"The main interface to the Eiffel ODBC classes"

cluster_chart *ROOT_CLUSTER*
class *COMPONENT_SELECT_DIALOG*
description
"Allows the user to select components from a list box."

class *COMPONENT_SPECIFIC_DIALOG*
description
"Allows the user to specify component properties."

class *STREAM_SPEC_DIALOG*

description
"Allows the user to specify stream properties."

class *APP_IDS*
description
"Contains user interface constants."

class *MAIN_WINDOW*
description
"The main application window."

class *MY_MENU*
description
"The application menu."

class *APP_SPEC_DIALOG*
description
"Allows the user to specify application properties."

class *APPLICATION*
description
"The root class."

cluster *APP_CLASSES*
cluster *WRAPPER_CLASSES*
cluster *DB_CLASSES*
cluster *ROUTER*

cluster_chart *ROUTER*
class *MESSAGE_ROUTER*
description
"Routes Messages to ODBC and CLIPS"

class *FACTFILE*
description
"Manages the CLIPS deffacts table."

cluster_chart *THESIS*
cluster *ROOT_CLUSTER*

Appendix C - BON Class Charts

class_chart *APPLICATION*
cluster *ROOT_CLUSTER*
description
"The root class."
queries
Create the application's main window

class_chart *APP_IDS*
cluster *ROOT_CLUSTER*
description
"User interface constants."

class_chart *APP_SPEC_DIALOG*
cluster *ROOT_CLUSTER*
description
" Allows the user to specify application properties."
inherits *APP_IDS*
queries
Carrier Gas List box

Customer Edit control

Cycle Time Edit Control

the main application object

Customer Location Edit control

Number Streams Edit control

the owner of this object
commands
inform the owner that the app object is filled

class_chart *CLIPS_DEBUG_FNS*
cluster *WRAPPER_CLASSES*
description
"Provides the interface to the CLIPSDLLdebug functions."
queries
Gets the current Dribble State

Gets the state of a watch item in the
CLIPS system

The DLL object that provides access to the
clips environment functions

Accessor function to the CLIPS DribbleActive command

Accessor function to the CLIPS DribbleOff command

Accessor function to the CLIPS DribbleOn command

Accessor function to the CLIPS GetWatchItem command

Accessor function to the CLIPS Unwatch command

Accessor function to the CLIPS watch command

commands

Commands the CLIPS Dribble state to off

Commands the CLIPS Dribble state to on

Commands the CLIPS Watch off for the item

Commands the CLIPS Watch on for the item

setup the DESC objects

class_chart *CLIPS_ENV_FNS*
cluster *WRAPPER_CLASSES*
description

"Provides the interface to the CLIPSDLLevironment functions."

queries

Remove the CLIPS environment

Run the system

Accessor function to the CLIPS Bload command

Accessor function to the CLIPS BSave command

Accessor function to the CLIPS Clear command

The DLL object that provides access to the clips environment functions

Accessor function to the CLIPS exit command

Accessor function to the CLIPS Init command

Accessor function to the CLIPS load command

Accessor function to the CLIPS reset command

Accessor function to the CLIPS Run command

Accessor function to the CLIPS save command

commands

Adds a router to the system

Load the binary rule file

save the knowledge base to a binary file

Clears the CLIPS environment

Initializes the clips system

Load the rule file

Resets the CLIPS system

Run a CLIPS session

save the knowledge base to a file

setup the DESC objects

class_chart *CLIPS_FACT_FNS*
cluster *WRAPPER_CLASSES*
description

"Provides the interface to the CLIPSDLLfact functions."

queries

Assert a fact given by a string

Creates a fact pointer from a deftemplate

Decrements the fact count for the given fact gets the fact duplication behavior flag

gets the fact list changed flag

gets the fact in pretty print form
uses an ANY object as a parameter since
Eiffel won't allow parameters of calls to
be changed

gets a fact from the fact list

gets the number of facts in the fact list

Retract a fact

Accessor function to the CLIPS Assert command

Accessor function to the CLIPS AssertString command

Accessor function to the CLIPS AssignFactStlotDefaults command

Accessor function to the CLIPS CreateFact command

Accessor function to the CLIPS DecrementFactCount command

The DLL object that provides access to the
clips environment functions

Accessor function to the CLIPS FactIndex command

Accessor function to the CLIPS Facts command

Accessor function to the CLIPS GetFactDuplication query

Accessor function to the CLIPS GetFactListChanged query

Accessor function to the CLIPS GetFactDuplication query

Accessor function to the CLIPS GetNextFact query

Accessor function to the CLIPS GetNumberOfFacts query

Accessor function to the CLIPS IncrementFactCount command

Accessor function to the CLIPS LoadFacts command

Accessor function to the CLIPS RemoveAllFacts command

Accessor function to the CLIPS Retract command

Accessor function to the CLIPS SaveFacts command

Accessor function to the CLIPS SetFactDuplication command

Accessor function to the CLIPS SetFactListChanged command

commands

Assert a fact given by a fact pointer object

Assigns defaults to a fact

Decrements the fact count for the given fact

increases the fact count for this fact by 1

Loads the fact file

Remove all facts that are in the WM

Save the facts

sets the fact duplication behavior flag

sets the fact list changed flag

setup the DESC objects

class_chart *CLIPS_INSTANCE_FNS*
cluster *WRAPPER_CLASSES*
description

"Provides the interface to the CLIPSDLLdeftemplate functions."

queries

Creates an empty instance of a class

Deletes an instance

Find an instance in a class

gets the total number of instances in all modules

gets the class reference for this instance gets the class name for this instance

gets the instance in pretty print form
uses an ANY object as a parameter since
Eiffel won't allow parameters of calls to
be changed

gets a instance from the instance list

gets a instance from the class

Make an instance using a command string

Determines if the instance is still vaild

Accessor function to the CLIPS CreateRawInstance command

Accessor function to the CLIPS DeleteInstance command

The DLL object that provides access to the
clips environment functions

Accessor function to the CLIPS FindInstance command

Accessor function to the CLIPS GetGlobalNumberOfInstances query

Accessor function to the CLIPS GetInstanceClass query

Accessor function to the CLIPS GetInstanceName query

Accessor function to the CLIPS GetInstancePPForm query

Accessor function to the CLIPS GetNextInstance query

Accessor function to the CLIPS GetNextInstanceInClass query

Accessor function to the CLIPS LoadInstances query

Accessor function to the CLIPS MakeInstance command

Accessor function to the CLIPS SaveInstances query

Accessor function to the CLIPS ValidInstanceAddress query

commands

Loads the instance file

Save the instances

setup the DESC objects

class_chart *CLIPS_STRATEGY_FNS*
cluster *WRAPPER_CLASSES*
description
"Provides the interface to the CLIPSDLLstrategy functions."
queries
return the current reasoning strategy

gets the number of memory requests

gets the amount of memory used

sets the current reasoning strategy

The DLL object that provides access to the
clips environment functions

Accessor function to the CLIPS GetStrategy command

Accessor function to the CLIPS MemRequests command

Accessor function to the CLIPS MemUsed command

Accessor function to the CLIPS SetStrategy command
commands
setup the DESC objects

class_chart *CLIPS_WRAPPER*
cluster *WRAPPER_CLASSES*
description
"Provides the interface to the CLIPSDLL"
commands
create the wrapper objects

clear the CLIPS environment

load the facts file

load the rule file

run a inference session

start up clips

class_chart *COMPONENT*
cluster *APP_CLASSES*
description
"The component specific class"
queries
get the maximum concentration

get the measured state

get the minimum concentration

get the name for the stream

get the normal concentration

get the units for the stream

max concentration of this component

Is the component measured?

min concentration of this component

the component name

normnal concentration of this component

the units for this component
commands
Set the maximum concentration

set the measured state

Set the minimum concentration

set the name for the stream

Set the normal concentration

set the units for the stream

initialize the object

class_chart *COMPONENT_SELECT_DIALOG*
cluster *ROOT_CLUSTER*
description
" Allows the user to select components from a list box. "
inherits *APP_IDS*
queries

get the current component_object

the popup that configures a component

list of possible components List box

add items to the selected list

used to store the component we are working on

used to store the changes to the stream

the owner of this object

remove items from the selected list

list of possible components List box

edit showing total % of stream defined

commands

update the list with new component specific information
update the stream percent dialog

add a component to the component list

Save the application data

remove a component from the list

inform the owner that the stream object is filled

class_chart *COMPONENT_SPECIFIC_DIALOG*
cluster *ROOT_CLUSTER*
description
" Allows the user to select components specific properties. "
inherits *APP_IDS*
queries
this component

edit showing upper concentration limit

Is this component analyzed?

edit showing normal concentration

the owner of this window

combo box listing the possible units

edit showing upper concentration limit
commands
Save the application data

update the component object

class_chart *DB_ANALYZER*
cluster *DB_CLASSES*
description
"The root of a candidate solution"
queries
Display contents
commands
add a component to the list

Set `analyzerserialnumber' with `t'

Set `applnumb' with `t'

Set `carriera' with `t'

Set `carrierb' with `t'

Set `comment' with `t'

Set `cycletime' with `t'

```
Set `detct1' with `t'
Set `finaltemp' with `t'
Set `initialtemp' with `t'
Set `isotherzo' with `t'
set `methanator' with `t'
Set `ovconfig' with `t'
Set `primary' with `t'
Set `process' with `t'
set `progtemp' with `t'
Set `projectnumber' with `t'
Set `splitter' with `t'
Set `sv1model' with `t'
Set `sv1size' with `t'
Set `sv1type' with `t'
Set `temprate' with `t'
Set `typecolumn' with `t'
set `vortex' with `t'
initialize the db_object
```

class_chart *DB_COMPONENT*
cluster *DB_CLASSES*
description
"The components of a candidate solution"
commands
Set `analyzerserialnumber' with `t'

Set `componentname' with `t'

Set `concentration' with `t'

Set `measurement' with `t'

Set `streamnumber' with `t'

initialize the db_object

class_chart *DB_WRAPPER*
cluster *DB_CLASSES*
description
" The main interface to the Eiffel ODBC classes "
commands
test the interface

terminate the session

create a query for each candiate to fill its components

gather the query results

class_chart *DLL_INTERFACE*
cluster *WRAPPER_CLASSES*
description
"Provides the interface to the CLIPSDLLprint router."
commands
this feature is called indirectly by the clips dll

class_chart *FACTFILE*
cluster *ROUTER*
description
"Manages the CLIPS deffacts table."
commands
operate on the fact file

class_chart *GC_APPLICATION*
cluster *APP_CLASSES*
description
"The main application class"
queries
The carrier gas for the application

The customer location

The customer name

The application cycle time

get the indexed stream object

The number of streams in this application

The stream aggregate attribute
commands
set the carrier gas for the application

set the customer location

Set the customer name

set the cycle time in seconds

put the stream into the array

set the number of streams

initialize the object

class_chart *MAIN_WINDOW*
cluster *ROOT_CLUSTER*
description
"The main application window."
inherits *APP_IDS*
queries
to give access to the current stream

to give access to the GC object

Window's icon

When the user can close the window?

Window's menu

Don't intentionally lose an objects data

Window's title

commands

give the data to the app object
look to see if the system is ready
to run a configuration session

set the GC application object
finalize the window setup

Message to inform that the feature is not implemented

open the GC_APPLICATION object

save the GC_APPLICATION object

update the project title

class_chart *MESSAGE_ROUTER*
cluster *ROUTER*
description
"Routes Messages to ODBC and CLIPS"
commands

add a new analyzer to the candidates list

add a new component to the current candidate

the target application facts are written to the facts file

the candidate facts are written to the facts file

create the router objects

begin a CLIPS session

class_chart *MY_MENU*
cluster *ROOT_CLUSTER*
description
"The main application menu."
commands
disable 'position

Enable 'position

class_chart *STREAM*
cluster *APP_CLASSES*
description
"The application stream"
queries
get a component from the list by its index

get a component from the
list having the name attribute
given in name

get the configured state

get the corrosive state

get the solids state

get the number of components in the stream

get the pH

get the phase for the stream

get the polimer state

get the return pressure

get the stream pressure

get the tag for the stream

get the temperature

the list of components for this stream

set true if 100% of the components are
specified

is the stream corrosive

does the stream have disolved solids

The stream pH

The phase of the gas in this stream

does the stream polimerize

The return pressure

The stream pressure

The customer specific tag of this stream

The stream temperature

commands

add a component to the
list of components

remove a component from the list

replace the previous comp with this name
with the new comp

reset the configured state

set the configured state

set the corrosive state

set the solids state

Set the pH

set the phase for the stream

set the polimer state

set the return pressure

set the stream pressure

set the tag for the stream

Set the temperature

initialize the object

class_chart *STREAM_SPEC_DIALOG*
cluster *ROOT_CLUSTER*
description
"Allows the user to specify stream properties."
inherits *APP_IDS*
queries
is the stream corrosive

used to store the changes to the stream attributes

the owner of this object

stream pH Edit control

stream phase List box

does the stream polimerize

return pressure Edit Control

stream pressure Edit control

are dissolved solids in the stream

the app specific name for this stream

stream temp Edit control
commands
Save the stream data

inform the owner that the stream object is filled

cluster_chart *WRAPPER_CLASSES*
class *CLIPS_INSTANCE_FNS*
description
"Provides the interface to the CLIPSDLLdeftemplate functions."

class *CLIPS_STRATEGY_FNS*
description
"Provides the interface to the CLIPSDLLstrategy functions."

class *CLIPS_WRAPPER*
description
"Provides the interface to the CLIPSDLL"

class *DLL_INTERFACE*
description
"Provides the interface to the CLIPSDLLprint router."

class *CLIPS_ENV_FNS*
description
"Provides the interface to the CLIPSDLLevironment functions."

class *CLIPS_FACT_FNS*
description
"Provides the interface to the CLIPSDLLfact functions."

class *CLIPS_DEBUG_FNS*
description
"Provides the interface to the CLIPSDLLdebug functions."

Appendix D - BON Class Dictionary

class_dictionary *APP*
class *APPLICATION* ***cluster*** *ROOT_CLUSTER*
description
"The root class."

class *APP_IDS* ***cluster*** *ROOT_CLUSTER*
description
"User interface constants."

class *APP_SPEC_DIALOG* ***cluster*** *ROOT_CLUSTER*
description
" Allows the user to specify application properties."

class *CLIPS_DEBUG_FNS* ***cluster*** *WRAPPER_CLASSES*
description
"Provides the interface to the CLIPSDLLdebug functions."

class *CLIPS_ENV_FNS* ***cluster*** *WRAPPER_CLASSES*
description
"Provides the interface to the CLIPSDLLevironment functions."

class *CLIPS_FACT_FNS* ***cluster*** *WRAPPER_CLASSES*
description
"Provides the interface to the CLIPSDLLfact functions."

class *CLIPS_INSTANCE_FNS* ***cluster*** *WRAPPER_CLASSES*
description
"Provides the interface to the CLIPSDLLdeftemplate functions."

class *CLIPS_STRATEGY_FNS* ***cluster*** *WRAPPER_CLASSES*
description
"Provides the interface to the CLIPSDLLstrategy functions."

class *CLIPS_WRAPPER* ***cluster*** *WRAPPER_CLASSES*
description
"Provides the interface to the CLIPSDLL"

class *COMPONENT* ***cluster*** *APP_CLASSES*
description
"The component specific class"

class *COMPONENT_SELECT_DIALOG* ***cluster*** *ROOT_CLUSTER*
description
" Allows the user to select components from a list box. "

class *COMPONENT_SPECIFIC_DIALOG* ***cluster*** *ROOT_CLUSTER*
description
" Allows the user to select components specific properties. "

class *DB_ANALYZER* ***cluster*** *DB_CLASSES*
description
"The root of a candidate solution"

class *DB_COMPONENT* ***cluster*** *DB_CLASSES*
description
"The components of a candidate solution"

class *DB_WRAPPER* ***cluster*** *DB_CLASSES*
description
" The main interface to the Eiffel ODBC classes "

class *DLL_INTERFACE* ***cluster*** *WRAPPER_CLASSES*
description
"Provides the interface to the CLIPSDLLprint router."

class *FACTFILE* ***cluster*** *ROUTER*
description
"Manages the CLIPS deffacts table."

class *GC_APPLICATION* ***cluster*** *APP_CLASSES*
description
"The main application class"

class *MAIN_WINDOW* ***cluster*** *ROOT_CLUSTER*
description
"The main application window."

class *MESSAGE_ROUTER* ***cluster*** *ROUTER*
description
"Routes Messages to ODBC and CLIPS"

class *MY_MENU* ***cluster*** *ROOT_CLUSTER*
description
"The main application menu."

class *STREAM* ***cluster*** *APP_CLASSES*
description
"The application stream"

class *STREAM_SPEC_DIALOG* ***cluster*** *ROOT_CLUSTER*
description
"Allows the user to specify stream properties."

Appendix E - BON Class Interfaces

```
indexing
        description: " The root class . "
implemented class APPLICATION

inherit

        WEL

        WEL_SUPPORT

        WEL_APPLICATION

feature

        redefined idle_action

        effective main_window: MAIN_WINDOW
                        -- Create the application's main window

end -- class APPLICATION

indexing
        description: " User interface constants . "
class APP_IDS

feature

        id_configure_gaschromatograph: INTEGER

        id_configure_gaschromatograph_0: INTEGER

        id_configure_gaschromatograph_1: INTEGER

        id_configure_gaschromatograph_11: INTEGER

        id_configure_gaschromatograph_12: INTEGER

        id_configure_gaschromatograph_13: INTEGER

        id_configure_gaschromatograph_14: INTEGER
```

id_configure_gaschromatograph_15: INTEGER

id_configure_gaschromatograph_2: INTEGER

id_configure_gaschromatograph_3: INTEGER

id_configure_gaschromatograph_4: INTEGER

id_configure_gaschromatograph_5: INTEGER

id_configure_gaschromatograph_6: INTEGER

id_configure_gaschromatograph_7: INTEGER

id_configure_gaschromatograph_8: INTEGER

id_configure_gaschromatograph_9: INTEGER

id_configure_peripheral: INTEGER

id_file_close: INTEGER

id_file_exit: INTEGER

id_file_new: INTEGER

id_file_open: INTEGER

id_file_print: INTEGER

id_file_printsetup: INTEGER

id_file_sabe: INTEGER

id_file_saveas: INTEGER

id_go: INTEGER

id_ico_application: INTEGER

id_main_menu: INTEGER

id_stream_1: INTEGER

id_stream_10: INTEGER

id_stream_11: INTEGER

id_stream_12: INTEGER

id_stream_13: INTEGER

id_stream_14: INTEGER

id_stream_15: INTEGER

id_stream_2: INTEGER

id_stream_3: INTEGER

id_stream_4: INTEGER

id_stream_5: INTEGER

id_stream_6: INTEGER

id_stream_7: INTEGER

id_stream_8: INTEGER

id_stream_9: INTEGER

idc_carriergas: INTEGER

idc_chosen: INTEGER

idc_corrosive: INTEGER

idc_customer: INTEGER

idc_customerlocation: INTEGER

idc_cycletime: INTEGER

idc_dissolids: INTEGER

idc_lowerlimit: INTEGER

idc_measured: INTEGER

```
        idc_normal: INTEGER
        idc_ph: INTEGER
        idc_phase: INTEGER
        idc_poly: INTEGER
        idc_returnpress: INTEGER
        idc_select: INTEGER
        idc_streampercent: INTEGER
        idc_streampress: INTEGER
        idc_streams: INTEGER
        idc_tag: INTEGER
        idc_temperature: INTEGER
        idc_tochosen: INTEGER
        idc_toselect: INTEGER
        idc_units: INTEGER
        idc_upperlimit: INTEGER
        idd_appl_select: INTEGER
        idd_comp_prop: INTEGER
        idd_comp_select: INTEGER
        idd_stream_properties: INTEGER
end -- class APP_IDS
```

```
indexing
        description: " Allows the user to specify application properties. "
class APP_SPEC_DIALOG

inherit

        WEL

        WEL_WINDOWS

        APP_IDS

        WEL_MODAL_DIALOG

end -- class APP_SPEC_DIALOG

indexing
        description: "Provides the interface to the CLIPSDLLdebug functions."
class CLIPS_DEBUG_FNS

inherit

        BASE

        DESC

        DESC_GENERAL

        SHARED_LIBRARY_CONSTANTS

feature

        active: INTEGER
                        -- Gets the current Dribble State
                ensure
                        dribble_failure: Result >= 0
                end

        dribbleoff
                        -- Commands the CLIPS Dribble state to off
```

```
        ensure
                dribble_failure: clips_dribbleoff.integer_result = 1
        end

dribbleon
        -> filename: STRING
        -> state: BOOLEAN
                -- Commands the CLIPS Dribble state to on
        require
                state_exists: state /= void
                file_exists: filename /= void
        ensure
                dribble_failure: clips_dribbleon.integer_result = 1
        end

getwatchitem: INTEGER
        -> item: STRING
                -- Gets the state of a watch item in the
                -- CLIPS system
        require
                item_exists: item /= void
        ensure
                watchitem_failure: Result /= - 1
        end

unwatch
        -> item: STRING
                -- Commands the CLIPS Watch off for the item
        require
                item_exists: item /= void
        ensure
                watch_failure: clips_unwatch.integer_result >= 0
        end

watch
        -> item: STRING
                -- Commands the CLIPS Watch on for the item
        require
                item_exists: item /= void
        ensure
                watch_failure: clips_watch.integer_result >= 0
        end

make
```

end -- ***class*** *CLIPS_DEBUG_FNS*

```
indexing
        description: "Provides the interface to the CLIPSDLLevironment functions."
class CLIPS_ENV_FNS

inherit

        BASE

        DESC

        DESC_GENERAL

        SHARED_LIBRARY_CONSTANTS

feature

        add_route
                        -- Adds a router to the system

        bload
                -> filename: STRING
                        -- Load the binary rule file
                require
                        exists: filename /= void
                ensure
                        failure_to_load: clips_bload.integer_result = 1
                end

        bsave
                -> filename: STRING
                        -- save the knowledge base to a binary file
                require
                        exists: filename /= void
                ensure
                        failure_to_save: clips_bsave.integer_result = 1
                end

        clear
                        -- Clears the CLIPS environment

        exit: INTEGER
                        -- Remove the CLIPS environment

        initialize
```

```
            -- Initializes the clips system

    load
        -> filename: STRING
            -- Load the rule file
        require
            exists: filename /= void
        ensure
            failure_to_load: clips_load.integer_result = 1
        end

    reset
            -- Resets the CLIPS system

    run
            -- Run a CLIPS session

    runs: INTEGER
        -> cycles: INTEGER
            -- Run the system
        require
            exists: cycles /= void
        ensure
            failure_to_run: Result /= - 1
        end

    save
        -> filename: STRING
            -- save the knowledge base to a file
        require
            exists: filename /= void
        ensure
            failure_to_save: clips_save.integer_result = 1
        end

    make

end -- class CLIPS_ENV_FNS
```

```
indexing
        description: "Provides the interface to the CLIPSDLLfact functions."
class CLIPS_FACT_FNS

inherit

        BASE

        DESC

        DESC_GENERAL

        SHARED_LIBRARY_CONSTANTS

feature

        assert
                -> fact_ptr: ANY
                        -- Assert a fact given by a fact pointer object
                require
                        exists: fact_ptr /= void
                end

        assertstring: ANY
                -> fact: STRING
                        -- Assert a fact given by a string
                require
                        exists: fact /= void
                end

        assignfactslotdefaults
                -> fact: ANY
                        -- Assigns defaults to a fact
                require
                        exists: fact /= void
                ensure
                        assignfsd_failed: clips_assignfsd.integer_result /= 0
                end

        createfact: ANY
                -> fact: ANY
                        -- Creates a fact pointer from a deftemplate
                require
                        exists: fact /= void
```

```
        ensure
                createfact_failed: Result /= void
        end

decrementfactcount
        -> fact: ANY
                -- Decrements the fact count for the given fact
        require
                exists: fact /= void
        end

factindex: INTEGER
        -> fact: ANY
                -- Decrements the fact count for the given fact
        require
                exists: fact /= void
        ensure
                factindex_failed: Result /= void
        end

facts
        -> output_device: STRING
        -> fact: ANY
        -> start: INTEGER
        -> stop: INTEGER
        -> max: INTEGER
        require
                output_device_exists: output_device /= void
                fact_exists: fact /= void
                start_exists: start /= void
                stop_exists: stop /= void
                max_exists: max /= void
        end

getfactduplication: INTEGER
                -- gets the fact duplication behavior flag
        ensure
                getfactdup_failed: Result = 0 or Result = 1
        end

getfactlistchanged: INTEGER
                -- gets the fact list changed flag
        ensure
                getfactlistchanged_failed: Result = 0 or Result = 1
        end
```

```
getfactppform: STRING
        -> fact: ANY
                -- gets the fact in pretty print form
                -- uses an ANY object as a parameter since
                -- Eiffel won't allow parameters of calls to
                -- be changed
        require
                exists: fact /= void
        ensure
                getppform_failed: Result /= void
        end

getnextfact: ANY
        -> fact: ANY
                -- gets a fact from the fact list

getnumberoffacts: INTEGER
                -- gets the number of facts in the fact list
        ensure
                getnumberoffacts_failed: Result >= 0
        end

incrementfactcount
        -> fact: ANY
                -- increases the fact count for this fact by 1
        require
                exists: fact /= void
        end

loadfacts
        -> filename: STRING
                -- Loads the fact file
        require
                exists: filename /= void
        ensure
                loadfacts_failed: clips_loadfacts.integer_result /= 0
        end

removeallfacts
                -- Remove all facts that are in the WM

retract: INTEGER
        -> fact: ANY
                -- Retract a fact
```

```
                require
                        exists: fact /= void
                ensure
                        retract_failed: Result /= void
                end

        savefacts
                -> filename: STRING
                -> scope: INTEGER
                        -- Save the facts
                require
                        exists: filename /= void
                        scope_exists: scope /= void
                ensure
                        loadfacts_failed: clips_savefacts.integer_result /= 0
                end

        setfactduplication
                -> state: INTEGER
                        -- sets the fact duplication behavior flag
                require
                        exists: state /= void
                        valid: state = 0 or state = 1
                ensure
                        setfactdup_failed: clips_setfactduplication.integer_result /= state
and (clips_setfactduplication.integer_result = 0 or clips_setfactduplication.integer_result
= 1)
                end

        setfactlistchanged
                -> state: INTEGER
                        -- sets the fact list changed flag
                require
                        exists: state /= void
                        valid: state = 0 or state = 1
                end

        make

end -- class CLIPS_FACT_FNS
```

```
indexing
        description: "Provides the interface to the CLIPSDLLdeftemplate functions."
class CLIPS_INSTANCE_FNS

inherit

        BASE

        DESC

        DESC_GENERAL

        SHARED_LIBRARY_CONSTANTS

feature

        createrawinstance: ANY
                -> instance: ANY
                -> name: STRING
                        -- Creates an empty instance of a class
                require
                        exists: instance /= void
                        name_exists: name /= void
                ensure
                        createrawintance_failed: Result /= void
                end

        deleteinstance: INTEGER
                -> instance: ANY
                        -- Deletes an instance
                require
                        exists: instance /= void
                ensure
                        delete_failed: clips_deleteinstance.integer_result >= 0
                end

        findinstance: ANY
                -> module: ANY
                -> name: STRING
                        -- Find an instance in a class
                require
                        exists: module /= void
                        name_exists: name /= void
                end
```

```
getglobalnumberofinstances: INTEGER
        -- gets the total number of instances in all modules
    ensure
        getinstanceclass_failed: Result /= void and Result >= 0
    end

getinstanceclass: ANY
    -> instance: ANY
        -- gets the class reference for this instance
    require
        exists: instance /= void
    ensure
        getinstanceclass_failed: Result /= void
    end

getinstancename: STRING
    -> instance: ANY
        -- gets the class name for this instance
    require
        exists: instance /= void
    ensure
        getinstancename_failed: Result /= void
    end

getinstanceppform: STRING
    -> instance: ANY
        -- gets the instance in pretty print form
        -- uses an ANY object as a parameter since
        -- Eiffel won't allow parameters of calls to
        -- be changed
    require
        exists: instance /= void
    ensure
        getppform_failed: Result /= void
    end

getnextinstance: ANY
    -> instance: ANY
        -- gets a instance from the instance list
    require
        exists: instance /= void
    end
```

```
getnextinstanceinclass: ANY
        -> instance: ANY
        -> cname: ANY
                -- gets a instance from the class
        require
                exists: instance /= void
                class_exists: cname /= void
        end

loadinstances
        -> filename: STRING
                -- Loads the instance file
        require
                exists: filename /= void
        ensure
                load_failed: clips_loadinstances.integer_result /= 0
        end

makeinstance: ANY
        -> command: STRING
                -- Make an instance using a command string
        require
                exists: command /= void
        ensure
                make_failed: Result /= void
        end

saveinstances
        -> filename: STRING
        -> scope: INTEGER
                -- Save the instances
        require
                exists: filename /= void
                scope_exists: scope /= void
        ensure
                save_failed: clips_saveinstances.integer_result /= 0
        end

validinstanceaddress: INTEGER
        -> instance: ANY
                -- Determines if the instance is still vaild
        require
                exists: instance /= void
        ensure
                validity_failed: Result /= void
```

```
        end

    make

end -- class CLIPS_INSTANCE_FNS

indexing
    description: "Provides the interface to the CLIPSDLLstrategy functions."
class CLIPS_STRATEGY_FNS

inherit

    BASE

    DESC

    DESC_GENERAL

    SHARED_LIBRARY_CONSTANTS

feature

    getstrategy: INTEGER
            -- return the current reasoning strategy
        ensure
            exists: Result /= void
        end

    memoryrequests: INTEGER
            -- gets the number of memory requests
        ensure
            exists: Result /= void
        end

    memoryused: INTEGER
            -- gets the amount of memory used
        ensure
            exists: Result /= void
        end

    setstrategy: INTEGER
        -> strategy: INTEGER
            -- sets the current reasoning strategy
        require
            exists: strategy /= void
```

```
			ensure
					exists: Result /= void
			end

		make

end -- class CLIPS_STRATEGY_FNS

indexing
		description: "Provides the interface to the CLIPSDLL"
class CLIPS_WRAPPER

inherit

		BASE

		DESC

		DESC_GENERAL

		SHARED_LIBRARY_CONSTANTS

feature

		make
					-- create the wrapper objects

end -- class CLIPS_WRAPPER
```

```
indexing
    description: "The component specific class."
class  COMPONENT

feature

    get_maximum: REAL
            -- get the maximum concentration
        ensure
            Result = maximum
        end

    get_measured: BOOLEAN
            -- get the measured state
        ensure
            Result = measured
        end

    get_minimum: REAL
            -- get the minimum concentration
        ensure
            Result = minimum
        end

    get_name: STRING
            -- get the name for the stream
        ensure
            Result = name
        end

    get_normal: REAL
            -- get the normal concentration
        ensure
            Result = normal
        end

    get_units: STRING
            -- get the units for the stream
        ensure
            Result = units
        end

    set_maximum
        -> t: REAL
```

```
                -- Set the maximum concentration
        require
                exists: t /= void
        ensure
                configured: maximum = t
        end

set_measured
        -> state: BOOLEAN
                -- set the measured state
        require
                exists: state /= void
        ensure
                configured: measured = state
        end

set_minimum
        -> t: REAL
                -- Set the minimum concentration
        require
                exists: t /= void
        ensure
                configured: minimum = t
        end

set_name
        -> t: STRING
                -- set the name for the stream
        require
                exists: t /= void
        ensure
                configured: name = t
        end

set_normal
        -> t: REAL
                -- Set the normal concentration
        require
                exists: t /= void
        ensure
                configured: normal = t
        end

set_units
        -> t: STRING
```

```
            -- set the units for the stream
        require
            exists: t /= void
        ensure
            configured: units = t
        end

    make
            -- initialize the object

end -- class COMPONENT

indexing
    description: " Allows the user to select components from a list box."
class COMPONENT_SELECT_DIALOG

inherit

    WEL

    WEL_WINDOWS

    WEL_CONSTANTS

    APP_IDS

    WEL_BN_CONSTANTS

    WEL_LBN_CONSTANTS

    WEL_MODAL_DIALOG

end -- class COMPONENT_SELECT_DIALOG

indexing
    description: " Allows the user to select component specific properties."
class COMPONENT_SPECIFIC_DIALOG

inherit

    WEL

    WEL_WINDOWS

    APP_IDS
```

```
        WEL_MODAL_DIALOG

end -- class COMPONENT_SPECIFIC_DIALOG

indexing
        description: " The root of a candidate solution. "
class DB_ANALYZER

feature

        add_component
                -> comp: DB_COMPONENT
                        -- add a component to the list

        analyzerserialnumber: STRING

        applnumb: STRING

        carriera: STRING

        carrierb: STRING

        comment: STRING

        components: LINKED_LIST [G] [DB_COMPONENT]

        cycletime: STRING

        detct1: STRING

        finaltemp: STRING

        initaltemp: STRING

        isotherzo: STRING

        methanator: BOOLEAN

        effective out: STRING
                        -- Display contents

        ovconfig: STRING

        primary: CHARACTER
```

```
process: STRING

progtemp: BOOLEAN

projectnumber: STRING

set_analyzerserialnumber
        -> t: STRING
                -- Set `analyzerserialnumber' with `t'
        require
                argument_exists: not (t = void)
        ensure
                analyzerserialnumber = t
        end

set_applnumb
        -> t: STRING
                -- Set `applnumb' with `t'
        require
                argument_exists: not (t = void)
        ensure
                applnumb = t
        end

set_carriera
        -> t: STRING
                -- Set `carriera' with `t'
        require
                argument_exists: not (t = void)
        ensure
                carriera = t
        end

set_carrierb
        -> t: STRING
                -- Set `carrierb' with `t'
        require
                argument_exists: not (t = void)
        ensure
                carrierb = t
        end

set_comment
        -> t: STRING
```

```
                        -- Set `comment' with `t'
            require
                        argument_exists: not (t = void)
            ensure
                        comment = t
            end

set_cycletime
            -> t: STRING
                        -- Set `cycletime' with `t'
            require
                        argument_exists: not (t = void)
            ensure
                        cycletime = t
            end

set_detct1
            -> t: STRING
                        -- Set `detct1' with `t'
            require
                        argument_exists: not (t = void)
            ensure
                        detct1 = t
            end

set_finaltemp
            -> t: STRING
                        -- Set `finaltemp' with `t'
            require
                        argument_exists: not (t = void)
            ensure
                        finaltemp = t
            end

set_initaltemp
            -> t: STRING
                        -- Set `initialtemp' with `t'
            require
                        argument_exists: not (t = void)
            ensure
                        initaltemp = t
            end

set_isotherzo
            -> t: STRING
```

```
			-- Set `isotherzo' with `t'
		require
			argument_exists: not (t = void)
		ensure
			isotherzo = t
		end

	set_methanator
		-> t: BOOLEAN
			-- set `methanator' with `t'
		require
			argument_exists: not (t = void)
		ensure
			methanator = t
		end

	set_ovconfig
		-> t: STRING
			-- Set `ovconfig' with `t'
		require
			argument_exists: not (t = void)
		ensure
			ovconfig = t
		end

	set_primary
		-> t: CHARACTER
			-- Set `primary' with `t'
		require
			argument_exists: not (t = void)
		ensure
			primary = t
		end

	set_process
		-> t: STRING
			-- Set `process' with `t'
		require
			argument_exists: not (t = void)
		ensure
			process = t
		end

	set_progtemp
		-> t: BOOLEAN
```

```
            -- set `progtemp' with `t'
        require
            argument_exists: not (t = void)
        ensure
            progtemp = t
        end

    set_projectnumber
        -> t: STRING
            -- Set `projectnumber' with `t'
        require
            argument_exists: not (t = void)
        ensure
            projectnumber = t
        end

    set_splitter
        -> t: BOOLEAN
            -- Set `splitter' with `t'
        require
            argument_exists: not (t = void)
        ensure
            splitter = t
        end

    set_sv1model
        -> t: STRING
            -- Set `sv1model' with `t'
        require
            argument_exists: not (t = void)
        ensure
            sv1model = t
        end

    set_sv1size
        -> t: STRING
            -- Set `sv1size' with `t'
        require
            argument_exists: not (t = void)
        ensure
            sv1size = t
        end

    set_sv1type
        -> t: STRING
```

```
                -- Set `sv1type' with `t'
        require
                argument_exists: not (t = void)
        ensure
                sv1type = t
        end

set_temprate
        -> t: STRING
                -- Set `temprate' with `t'
        require
                argument_exists: not (t = void)
        ensure
                temprate = t
        end

set_typecolumn
        -> t: STRING
                -- Set `typecolumn' with `t'
        require
                argument_exists: not (t = void)
        ensure
                typecolumn = t
        end

set_vortex
        -> t: BOOLEAN
                -- set `vortex' with `t'
        require
                argument_exists: not (t = void)
        ensure
                vortex = t
        end

splitter: BOOLEAN

sv1model: STRING

sv1size: STRING

sv1type: STRING

temprate: STRING

typecolumn: STRING
```

```
        vortex: BOOLEAN

        make
                        -- initialize the db_object

end -- class DB_ANALYZER

indexing
        description: " The components of a candidate solution. "
class DB_COMPONENT

feature

        analyzerserialnumber: STRING

        componentname: STRING

        concentration: DOUBLE

        init_strings

        measurement: DOUBLE

        set_analyzerserialnumber
                -> t: STRING
                        -- Set `analyzerserialnumber' with `t'
                require
                        argument_exists: not (t = void)
                ensure
                        analyzerserialnumber = t
                end

        set_componentname
                -> t: STRING
                        -- Set `componentname' with `t'
                require
                        argument_exists: not (t = void)
                ensure
                        componentname = t
                end

        set_concentration
                -> t: DOUBLE
                        -- Set `concentration' with `t'
```

```
            require
                    argument_exists: not (t = void)
            ensure
                    concentration = t
            end

    set_measurement
            -> t: DOUBLE
                    -- Set `measurement' with `t'
            require
                    argument_exists: not (t = void)
            ensure
                    measurement = t
            end

    set_streamnumber
            -> t: INTEGER
                    -- Set `streamnumber' with `t'
            require
                    argument_exists: not (t = void)
            ensure
                    streamnumber = t
            end

    streamnumber: INTEGER

    make
                    -- initialize the db_object

end -- class DB_COMPONENT
```

```
indexing
        description: "The main interface to the Eiffel ODBC classes."
class DB_WRAPPER

inherit

        STORE

        UTILITIES

        RDBMS_HANDLE

        DBMS

        ESTORE_SUPPORT

        ACTION

        RDB_HANDLE

end -- class DB_WRAPPER

indexing
        description: "Provides the interface to the CLIPSDLLprint router."
class DLL_INTERFACE

feature

        clips_printer
                -> char_pointer: POINTER
                        -- this feature is called indirectly by the clips dll

        get_obj
                -> obj: ANY

        get_proc
                -> function: POINTER

        make

        message: STRING

end -- class DLL_INTERFACE
```

```
indexing
        description: "Manages the CLIPS deffacts table."
class  FACTFILE

inherit

        BASE

        KERNEL

        PLAIN_TEXT_FILE

feature

        to_fact_file
                -> fact_string: STRING
                -> state: BOOLEAN
                        -- operate on the fact file
                require
                        exists: fact_string /= void
                end

end  -- class  FACTFILE

indexing
        description: "The main application class."
class  GC_APPLICATION

inherit

        BASE

        KERNEL

        STORABLE

feature

        carrier_gas: STRING
                        -- The carrier gas for the application

        customer_location: STRING
                        -- The customer location

        customer_name: STRING
```

```
                -- The customer name

cycle_time: INTEGER
                -- The application cycle time

fill_streams

get_stream: STREAM
        -> index: INTEGER
                -- get the indexed stream object
        require
                valid_index: index >= 0
                exists: index /= void
        ensure
                stream_obtained: Result = streams.item (index)
        end

number_streams: INTEGER
                -- The number of streams in this application

set_carrier_gas
        -> gas: STRING
                -- set the carrier gas for the application
        require
                exists: gas /= void
        ensure
                configured: carrier_gas = gas
        end

set_customer_location
        -> location: STRING
                -- set the customer location
        require
                exists: location /= void
        ensure
                configured: customer_location = location
        end

set_customer_name
        -> name: STRING
                -- Set the customer name
        require
                exists: name /= void
        ensure
                configured: customer_name = name
```

```
        end

    set_cycle_time
        -> time: INTEGER
                -- set the cycle time in seconds
        require
                exists: time /= void
        ensure
                configured: cycle_time = time
        end

    set_stream
        -> s: STREAM
        -> index: INTEGER
                -- put the stream into the array
        require
                stream_exists: s /= void
                index_exists: index /= void
                index_valid: index >= 0
        ensure
                stream_set: streams.item (index) = s
        end

    set_streams
        -> number: INTEGER
                -- set the number of streams
        require
                exists: number /= void
        ensure
                configured: number_streams = number
        end

    streams: ARRAY [G] [STREAM]
                -- The stream aggregate attribute

    make
                -- initialize the object

end -- class GC_APPLICATION
```

```
indexing
        description: "The main application window."
class MAIN_WINDOW

inherit

        WEL

        WEL_WINDOWS

        WEL_CONSTANTS

        APP_IDS

        WEL_OFN_CONSTANTS

        WEL_FRAME_WINDOW

end -- class MAIN_WINDOW

indexing
        description: "Routes messages between ODBC and CLIPS."
class MESSAGE_ROUTER

end -- class MESSAGE_ROUTER
```

```
indexing
        description: " The main application menu. "
class MY_MENU

inherit

        WEL

        WEL_SUPPORT

        WEL_MENU

feature

        disable_item_by_position
                -> position: INTEGER
                        -- disable 'position

        enable_item_by_position
                -> position: INTEGER
                        -- Enable 'position

end -- class MY_MENU

indexing
        description: " The application stream. "
class STREAM

feature

        add_component
                -> name: STRING
                        -- add a component to the
                        -- list of components
                require
                        exists: name /= void
                ensure
                        one_more_comp: components.count = 1 + old components.count
                end

        get_component_by_index: COMPONENT
                -> i: INTEGER
                        -- get a component from the list by its index
                require
```

```
            valid_index: i >= 0
            exists: i /= void
        end

    get_component_by_name: COMPONENT
        -> name: STRING
            -- get a component from the
            -- list having the name attribute
            -- given in name
        require
            exists: name /= void
        end

    get_configured: BOOLEAN
            -- get the configured state
        ensure
            Result = configured
        end

    get_corrosive: BOOLEAN
            -- get the corrosive state
        ensure
            Result = corrosive
        end

    get_disolids: BOOLEAN
            -- get the solids state
        ensure
            Result = dis_solids
        end

    get_number_of_components: INTEGER
            -- get the number of components in the stream
        ensure
            valid_count: Result >= 0
        end

    get_percent_stream: REAL

    get_ph: INTEGER
            -- get the pH
        ensure
            Result = ph
        end
```

```
get_phase: STRING
                -- get the phase for the stream
        ensure
                Result = phase
        end

get_polimer: BOOLEAN
                -- get the polimer state
        ensure
                Result = polimer
        end

get_rpress: REAL
                -- get the return pressure
        ensure
                Result = r_pressure
        end

get_spress: REAL
                -- get the stream pressure
        ensure
                Result = s_pressure
        end

get_tag: STRING
                -- get the tag for the stream
        ensure
                Result = tag
        end

get_temperature: REAL
                -- get the temperature
        ensure
                Result = temperature
        end

remove_component
        -> comp: COMPONENT
                -- remove a component from the list
        require
                exists: comp /= void
        ensure
                one_less_comp: components.count = old components.count - 1
        end
```

```
replace_component
        -> comp: COMPONENT
                -- replace the previous comp with this name
                -- with the new comp
        require
                exists: comp /= void
        ensure
                item_changed: components.item = comp
        end

reset_configured
                -- reset the configured state
        ensure
                configured: configured = false
        end

set_configured
                -- set the configured state
        ensure
                configured: configured = true
        end

set_corrosive
        -> state: BOOLEAN
                -- set the corrosive state
        require
                exists: state /= void
        ensure
                configured: corrosive = state
        end

set_disolids
        -> state: BOOLEAN
                -- set the solids state
        require
                exists: state /= void
        ensure
                configured: dis_solids = state
        end

set_ph
        -> p: INTEGER
                -- Set the pH
        require
                exists: p /= void
```

```
        ensure
                configured: ph = p
        end

set_phase
        -> p: STRING
                -- set the phase for the stream
        require
                exists: p /= void
        ensure
                configured: phase = p
        end

set_polimer
        -> state: BOOLEAN
                -- set the polimer state
        require
                exists: state /= void
        ensure
                configured: polimer = state
        end

set_rpress
        -> number: REAL
                -- set the return pressure
        require
                exists: number /= void
        ensure
                configured: r_pressure = number
        end

set_spress
        -> number: REAL
                -- set the stream pressure
        require
                exists: number /= void
        ensure
                configured: s_pressure = number
        end

set_tag
        -> t: STRING
                -- set the tag for the stream
        require
                exists: t /= void
```

```
                ensure
                        configured: tag = t
                end

        set_temperature
                -> t: REAL
                        -- Set the temperature
                require
                        exists: t /= void
                ensure
                        configured: temperature = t
                end

        make
                        -- initialize the object

end -- class STREAM

indexing
        description: " Allows the user to specify stream properties. "
class STREAM_SPEC_DIALOG

inherit

        WEL

        WEL_WINDOWS

        APP_IDS

        WEL_MODAL_DIALOG

end -- class STREAM_SPEC_DIALOG
```

VITA

George Eric Wolke

Candidate for the Degree of

Master of Science

Thesis: A RULE BASED EXPERT SYSTEM WHICH CONFIGURES GAS CHROMATOGRAPHS

Major Field: Computer Science

Biographical:

Education: Graduated from W.T. Clarke High School, Westbury, New York in June 1980; received Bachelor of Science degree in Electrical Engineering from Syracuse University, Syracuse, New York in May 1984. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in May 1997.

Experience: Over ten years experience as a professional Software Engineer. Currently employed as a Staff Software Engineer with Lockheed-Martin Astronautics, Denver, Colorado.

Professional Memberships: IEEE Computer Society, Association for Computing Machinery.