THE ILTIS DIAGNOSTIC SYSTEM

by

Major K. Ritchie

Project and Report submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE
IN
SYSTEMS ENGINEERING

Dr J.M. de la Garza, Chair

Dr R.T. Sumichrast

Dean B.S. Blanchard

July 1996
Blacksburg, Virginia

Key words: Iltis, Diagnosis, Troubleshooting, Software

THE ILTIS DIAGNOSTIC SYSTEM

by

Major Karen Ritchie

Committee Chairman: Dr J.M. de la Garza

Civil Engineering

(ABSTRACT)

Troubleshooting information for the Iltis Canadian Military Truck is in short supply. The applicable Canadian Forces Technical Orders are limited to one copy for each maintenance workshop and the personnel reductions recently experienced by the Canadian Forces has resulted in a need for greater efficiency in the workplace.

The need for a troubleshooting aid for the vehicle was investigated. Since Maintenance Manuals in use are limited in number and time-consuming to update, a computer-based system was considered.

A diagnostic software tool was designed that mirrored the function of the existing repair manuals. A programme was developed that could be used as a model for future development of computer-based diagnostic tools within the Canadian Armed Forces.

A description of the needs analysis, the expert system shell used to develop the software, and the diagnostic tool are included.

Acknowledgements

I would like to acknowledge the members of my advisory committee for their support during this project. In particular, many thanks to Dr. J.M. de la Garza for his guidance and interest throughout. Also, Mr. Primo Alcantara who spent some of his valuable time teaching me the finer points of Kappa-PC.

Furthermore, my deepest appreciation to the Electrical and Mechanical Engineering Branch of the Canadian Armed Forces who sponsored my studies at Virginia Tech.

Finally, to my husband Don, without whom I would never have made it through this year. Thankyou for your support and patience.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1: Introduction

Background

The Iltis Canadian Light Utility Truck is a four-wheel drive cross-country vehicle of military design that has been in service with the Canadian Armed Forces (CF) since 1985. Produced by Bombardier Inc, the Iltis replaced an aging fleet of American made jeeps.

The CF Iltis is based upon a Volkswagen design, modified to suit the particular requirements of employment in Canada. It can climb gradients of almost 60 percent and can cross water obstacles 0.6 meters deep while carrying a 550 kilogram payload. The Iltis is used to transport personnel and light equipment for the army, communications, airborne operations and support tasks. It is adaptable for the specific roles of cable laying, ambulance and Militia reconnaissance training.

There are approximately 1,900 Iltis in use today. It is used throughout Canada and on peacekeeping missions worldwide.

Problem Statement

An extensive set of Canadian Forces Technical Orders (CFTO) has been issued by the CF to address all facets of the operation and maintenance of the Iltis. The maintenance manuals within this set of CFTOs are used by CF Vehicle Technicians on all Bases where

the Iltis is located. In most cases, only one set of maintenance manuals is available in each workshop.

Approximately eighty percent of maintenance organizations, whether they be Maintenance Platoons within combat arms units, Maintenance Companies of Service Battalions or Base Maintenance Workshops, are tasked to train young men and women as Vehicle Technicians. These trainees undergo a basic vehicle mechanic course at a central school and are then dispatched to a maintenance organization for one year to undergo On The Job Training (OJT). Once the OJT is complete, they then proceed to another mechanic course after which they are considered to be fully qualified Vehicle Technicians.

A typical repair situation will arise when the driver of a vehicle notices something wrong. The driver will report the malfunction to his maintenance coordinator who will arrange for the vehicle to be inspected and the technical fault isolated and repaired. In some cases the information provided by the driver may be quite detailed, however it is usually fairly general in nature. Due to the lack of specific information, a vehicle technician will start at the highest vehicle sub-system when conducting a fault diagnosis.

The technician will note the reported malfunction and either consult the troubleshooting guide in the CFTO to establish what possible faults could cause the malfunction or use his own experience to form a diagnosis. On average, a trainee mechanic will be able to form a diagnosis without aid 30% of the time.

If the trainee is not capable of forming a diagnosis unaided, he will take the list of possible causes from the CFTO and check the vehicle for each one. In some cases the

technician will be perfectly capable of verifying the existence of a fault with no further information. For those faults that are more difficult to find, he might consult his supervisor or the CFTO.

Once he has determined which of the faults exist, he will consult the troubleshooting guide again to find out what repairs have to be completed. Information regarding exactly how to effect the repair is found throughout a number of CFTOs depending upon the specific repair required. Repairs to vehicles by OJT personnel are normally undertaken under the supervision of a qualified technician.

With the end of the cold war and the demise of the Soviet Union, there has been increasing political pressure in Canada to reduce the size of the CF. The long standing commitment to the North Atlantic Treaty Organization (NATO) in Europe has been curtailed resulting in the closure of two major bases in Germany. Across Canada there has been a reorganization of the army which has seen the closure of at least seven bases and the Land Engineering Test Establishment. As a result of these closures, and an increasing reliance on the contracting of work to civilian organizations, the CF has been reduced in personnel strength by roughly one quarter.

Canada has been a member of every United Nations (UN) peacekeeping mission since the UN was founded in 1952. Although the cold war has reduced East-West tension worldwide, there is now a larger number of UN peacekeeping missions than ever before. Canada continues to provide one of its four Battle-Groups to the UN mission in the Former Republic of Yugoslavia, as well as providing smaller contingents to Haiti, the

Golan Heights and Egypt, and individual UN Military Observers in many other locations.

The immediate effect of the personnel strength reductions and increasing commitments abroad is that the remaining members of the CF in Canada are expected to maintain the same fleet of equipment with less money and less people. The large number of OJT personnel in some workshops places a great demand on maintenance manuals for all vehicles, particularly the troubleshooting guides. Due to the scarcity of manuals and the abundance of young trainees, a way of leveraging the troubleshooting information as well as increasing the efficiency of supervisory personnel would be very beneficial.

One possible area of exploitation is the use of computers. Leveraging of troubleshooting information for vehicles is a good way to alleviate the demand on a limited source of manuals and enable supervisors to spend their time more productively than answer the same basic questions many times over.

Another obvious solution might be to simply produce more manuals, however, in other areas such as Administration, the cost of production and frequent updating of manuals has been found to be much higher than that of procuring and maintaining a computerized database.

Aim

To develop a limited computer-based diagnostic programme for the Iltis.

Project Scope

A prototype was developed that concentrated only on the engine of the Iltis. It listed all possible malfunctions by engine sub-system, and all possible faults which could cause the malfunctions. The user was asked to indicate which faults were present, it was then his responsibility to determine how to find the faults and test for them. Once the user had indicated which faults were present, the prototype gave general advice on how to repair the malfunction.

The Iltis Diagnostic System (IDS) expanded the prototype in both depth and breadth of application. It includes three of the five major sub-systems of the Iltis that require maintenance. In addition, the user is now provided with detailed instructions regarding how to determine if a fault exists that could cause a malfunction (except in those areas where the fault is obvious). This includes which tests to perform and how to perform them.

A table of contents has also be added. This will enable technicians to search for a malfunction if they are not sure from which vehicle sub-system it originates. For example, a driver may report that the alternator is squealing. The technician may be unsure whether the alternator is considered as part of the electrical system of the Iltis or the engine. The table of contents lists all of the major malfunctions that may be reported and instructs the technician where in the programme to find them. This addition is an improvement to the original information source as well as the prototype since the manual does not include this feature either.

The programme would be produced using Kappa-PC and be available for use on IBM compatible computers. Kappa-PC is a knowledge-based expert system tool and, although IDS is not an expert system, is extremely well adapted for use in this project.

Limitations

IDS is a troubleshooting guide, and therefore is not designed to teach the user how to conduct specific repairs. However, in order to demonstrate the potential future development of this tool, the system is capable of instructing the user how to repair faults within the chassis and suspension sub-system.

Also, IDS cannot be thought of as a simulator programme that mechanics can use to gain qualification before actually working on an Iltis. As stated in the scope, this is purely a diagnostic tool to be used primarily by trainee mechanics.

Chapter 2: The Systems Engineering Process

Systems Engineering Defined

Systems engineering is defined as the application of scientific and engineering efforts to:

> Transform an operational need into a description of system performance parameters and a preferred system configuration through the use of an iterative process of functional analysis, synthesis, optimization, definition, design, test and evaluation;
>
> Incorporate related technical parameters and ensure compatibility of all physical, functional, and program interfaces in a manner that optimizes the total system definition and design; and
>
> Integrate performance, producibility, reliability, maintainability, supportability, and other specialties into the overall engineering effort [Blanchard, 1990].

In other words, it is an approach where operational needs are translated into specific design requirements through a top-down, iterative process that considers the

entire product life-cycle.

The systems engineering process commences with the definition of a need which arises from a deficiency within an organization. The definition of need for this project is discussed in Chapter One.

Conceptual Design

Once a need is identified, then the actual system operational requirements and maintenance concept must be defined. This is usually accomplished by studying what the new item will be expected to do, for how long, and under what type of environmental conditions.

System requirements for a software package usually focus on the required performance characteristics rather than environment or life-cycle. Software is usually update frequently to keep pace with hardware improvements, and is not affected by the physical environment it is used in, nor the length of duty cycle. Similarly, software maintenance is usually much more simplified than that of a physical system.

For this project, the operational requirements are that the software must be able to provide the user information that is 100% correct, and in a manner that complements the manual methods in use. The system will be in use across the CF, meaning that adequate copies and licenses must be available.

One level of maintenance is anticipated for this software since the only expected failures will be due to programming faults.

Feasibility Study

The production of software programmes by the CF has been notoriously poor in the past. There are few formally trained software engineers, and the inherent fluidity of the personnel management system means that the average time spent in one job is only four years. The Electrical and Mechanical Engineering (EME) Branch uses a time accounting programme that has been written by military personnel, but it is unwieldy and is frequently undergoing updates to deal with faults.

In the recent past, the CF has turned more and more to civilian industry to provide customized software packages under license to CF. Mainly in the administrative area, there are now programmes to help personnel complete annual evaluations of their subordinates and order supplies. Also, a growing number of Technical and Administrative Orders are being produced on CD-ROM, replacing the paper versions of these documents.

Because of this increasing proliferation of computer-based information, personal computers have become common place, even on the shop floor within maintenance organisations.

Fiscal restraint and re-engineering have also taken their toll on most Federal agencies. The personnel of the CF find themselves asked to do "more with less". Contracting services to civilian industry is becoming the fashion since it incurs significant cost savings in most cases.

There are three alternatives regarding the production of this software:

The first alternative is to produce the software in-house, and provide support to

amend it as necessary. The second alternative is to contract the production and support of the software to a civilian agency. The third is to have a civilian firm produce the software and sell the rights to CF for their future support.

Preliminary Design

The preliminary design phase includes functional analysis, allocation of requirements and a trade-off study.

Functional analysis involves identifying what functions the software must perform. Once the different functions have been established, the resources required to perform them, as well as any support equipment, can be identified.

There is a baseline requirement for support capability in every system design process, this project addresses the support of the Iltis vehicle system. Figures 1 and 2 depict the top level functional flow diagram for Iltis vehicle system. Once the need for diagnostic software has been established, then the software requirements must be considered (Figure 3). In this case, the diagnostic programme would be required to request fault data, process that data and then display results to the user (Figure 4). Therefore, there is a need for a user interface and a processing unit that can manipulate the knowledge from the troubleshooting guide and the data given by the user.

Maintenance requirements must also be considered (Figure 5). In this case one level of maintenance is envisioned.

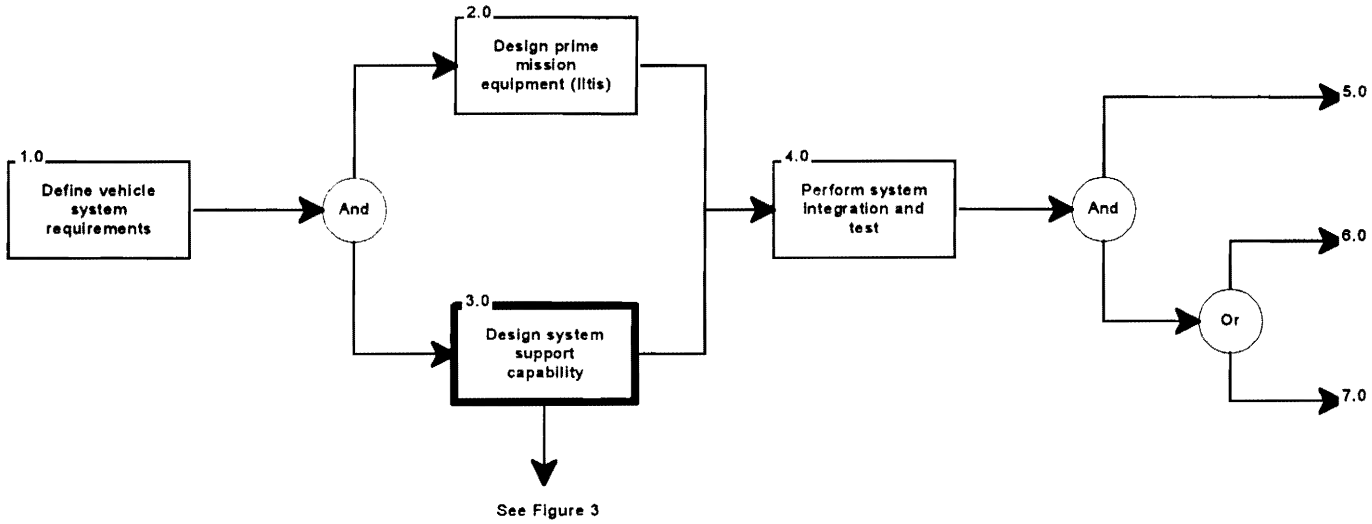

Figure 1. System level functional flow diagram [Blanchard, 1990]

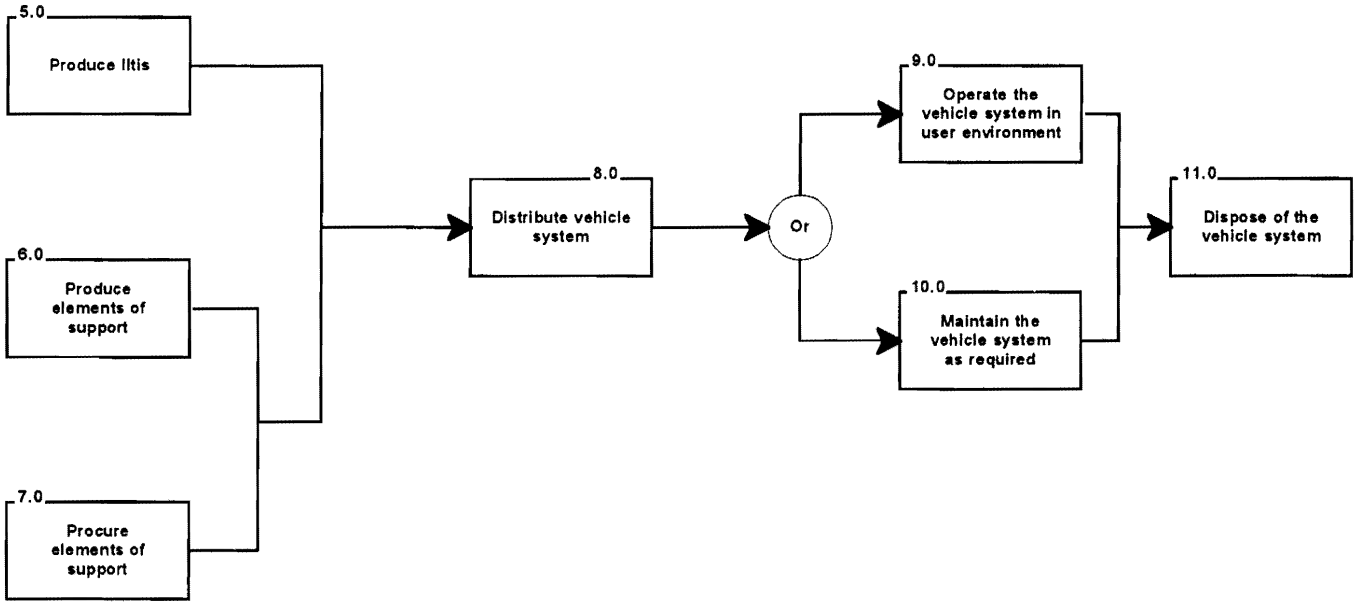

Figure 2. System level functional flow diagram (cont) [Blanchard, 1990]

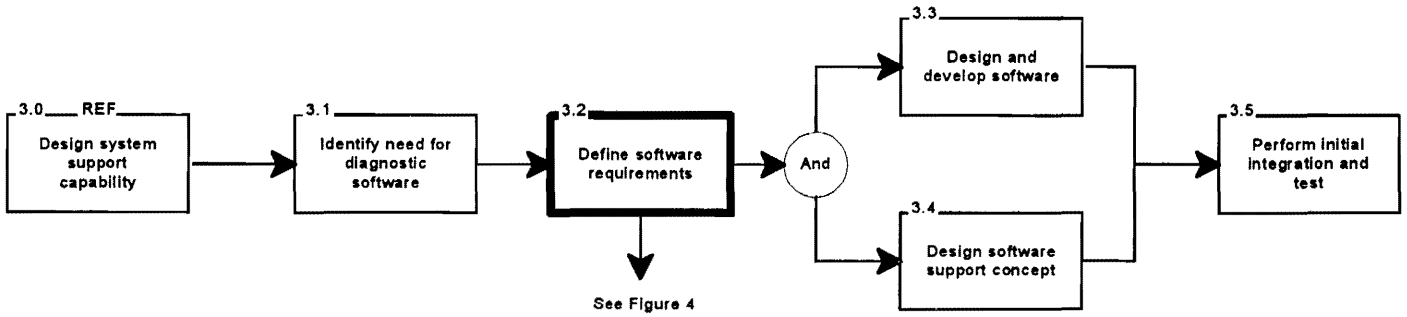

Figure 3. First level functional flow diagram

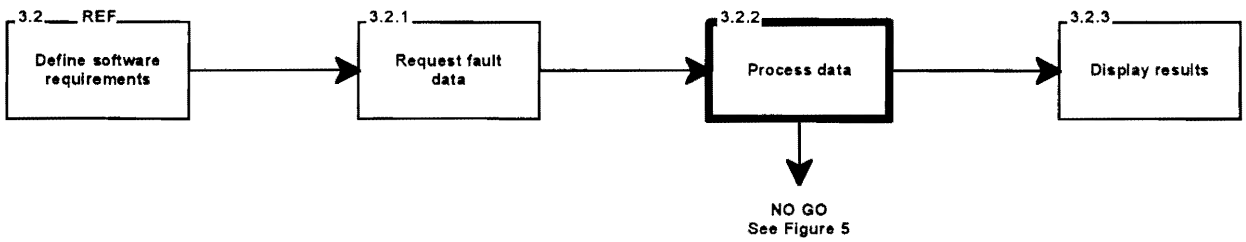

Figure 4. Second level functional flow diagram

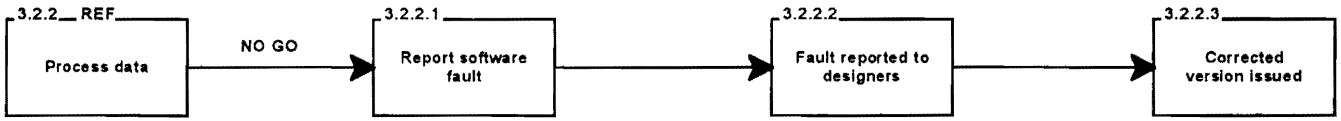

Figure 5. Maintenance flow diagram

Comparison of the three feasible alternatives previously identified is the core of the trade-off study. A simplified cost breakdown structure of the project is shown in Figure 6. The project total cost is sub-divided into four categories: Research & Development, Investment, Operation & Maintenance, and Retirement & Disposal. Each sub-category is further divided into a number of costs which have been described and weighted for each alternative in Tables 1 to 3. Certain areas are considered to be comparable for all three options and are not discussed. These areas include the cost of production of the software and distribution and are identified in Figure 6. The weighting used is scored out of ten points, where 1.0 is excellent and 10.0 is very poor. A rating of 3.0 or lower corresponds to the fact that the task is already being accomplished by the organization and little to no change in operating procedure would be required. Ratings from 3.1 to 6.0 denote that the organization may not have complete control of the task or not be fully versed in its accomplishment. Ratings above 6.0 represent serious difficulties in carrying out the task. A "military factor" has been considered where warranted to account for the fact that service personnel are required to undergo training and testing in soldiercraft and are usually less current than their civilian counterparts regarding software development advances.

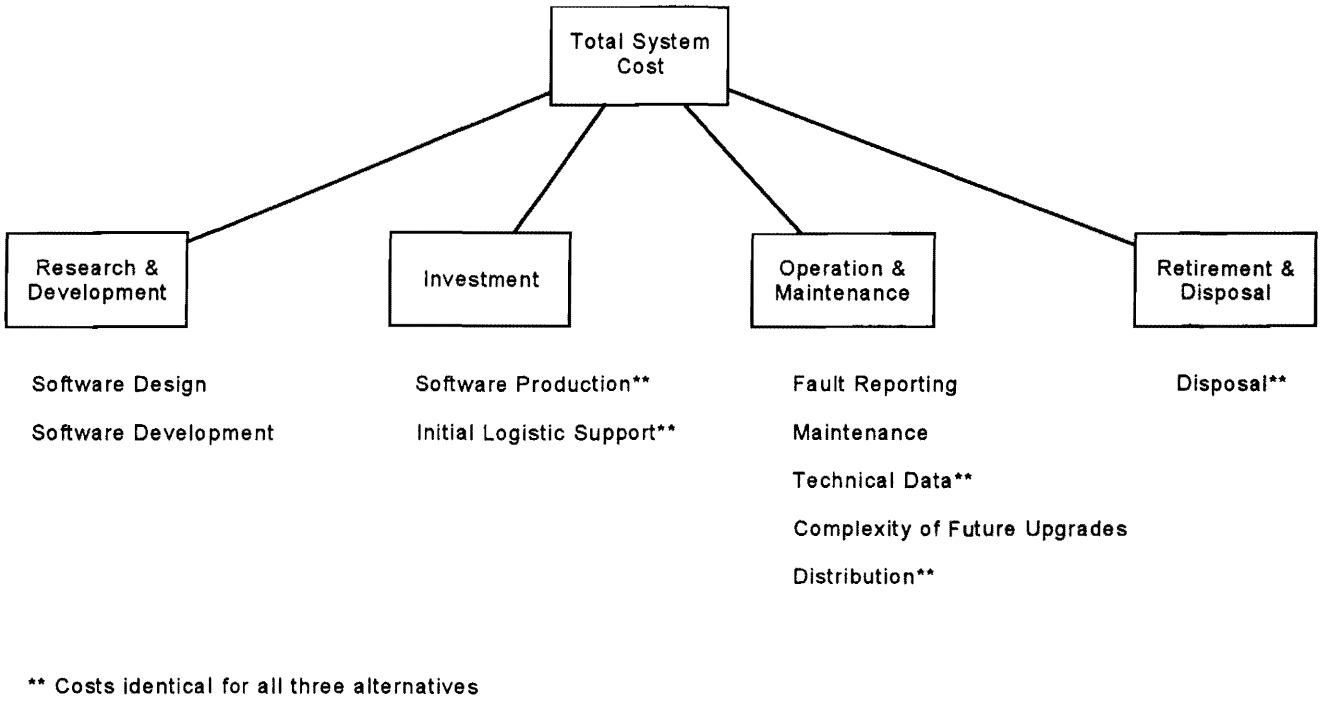

Figure 6. Cost breakdown structure

Base costs have been estimated from personal experience and as such are speculative in nature. These figures were arrived at by gauging how many person-years of work would be involved and at what rate of pay, and adding additional costs such as infrastructure. Costs were calculated in Canadian Dollars. For example, Software Development Base Costs in Table 1 were arrived at by the following method:

- approximately 24 months to complete development
- require one programmer
- annual salary of $35,000/person
- 10% overhead costs
- Base Cost = 2 years x 1 programmer x $35,000 = $70,000

  plus 10% ($7,000) = $77,000

The Weighted Cost is calculated by multiplying the Weighting by the Base Cost. The total rated cost of each alternative is highly sensitive to changes in Weighting or Base Cost for any cost factor.

A summary of the three alternative cost calculations may be found in Table 4.

Table 1. Alternative one cost calculations

Cost/Benefit	Weighting	Base Cost	Rated Cost
Software Design All design activities would be performed by military software engineers with input from applicable users. Direct control by the military would be possible since the design personnel would be in the direct chain of command. Design parameters could be easily changed, resulting in perhaps a better design but also a lengthier time requirement.	3.0	$20,000	$60,000
Software Development Performed solely by military software engineers. Based upon past experience, this will be a lengthy process when compared with similar projects performed by civilian industry. Code efficiency could be lower due to the fact that military personnel rotate through jobs at regular intervals which causes disruption in some areas.	5.0	$77,000	$385,000
Maintenance The likelihood of faulty code is fairly high. Delays caused by the wait for updated versions of the software may result in a loss of confidence in the program.	7.0	$30,000	$210,000
Fault Reporting A fault reporting network is already in existence within the CF. The establishment of at least one person to man the office of primary responsibility for the first year after fielding must be considered here.	2.0	$40,000	$80,000
Complexity of Future Upgrades There is a high potential that those personnel who developed the original software will not be available to perform future updates. With no standardized coding procedures in the CF, it may be difficult to upgrade the software in an efficient manner.	7.0	$77,000	$539,000
Total rated cost ($CND)			$1,274,000

Table 2. Alternative two cost calculations

Cost/Benefit	Weighting	Base Cost	Rated Cost
Software Design All design activities would be performed by a civilian firm with input from the military. A significant amount of time and effort will have to be expended in this area to ensure that the user's (military's) needs are being met. Due to contractual constraints, the CF will not have the flexibility to change requirements throughout the design phase, but this will probably result in a better engineered product.	2.0	$60,000	$120,000
Software Development Performed by a civilian firm using experienced software engineers and perhaps proprietary software. Code should be well organized and streamlined. Financial cost would be significantly higher than alternative one	2.0	$60,000	$120,000
Maintenance Assuming that the company is experienced in software development, the likelihood of program faults is considered to be lower than that of alternative one.	3.0	$30,000	$90,000
Fault Reporting A method of reporting program faults would have to be devised. Perhaps a 1-800 phone number that would link directly to the software development firm.	4.0	$50,000	$200,000
Complexity of Future Upgrades More easily done by a software development firm due to a more stable personnel environment and standardized coding practices. However, more financially burdensome due to the additional demands on the contract.	2.0	$60,000	$120,000
Total rated cost ($CND)			$650,000

Table 3. Alternative three cost calculations

Cost/Benefit	Weighting	Base Cost	Rated Cost
Software Design Performed by a civilian firm using experienced software engineers and perhaps proprietary software. Code should be well organized and streamlined. Financial cost would be significantly higher than alternative one	2.0	$40,000	$80,000
Software Development Performed by a civilian firm using experienced software engineers and perhaps proprietary software. Code should be well organized and streamlined. Financial cost would be significantly higher than alternative one	2.0	$60,000	$120,000
Maintenance Assuming that the company is experienced in software development, the likelihood of program faults is considered to be lower than that of alternative one. However, the assumption of responsibility for maintenance of the program by the CF may prove difficult since the agency attempting to maintain the program did not develop it. Also, the question of proprietary right would have to addressed. Either the CF would buy the rights to the package or pay the originating firm for the right to maintain it.	7.0	$40,000	$280,000
Fault Reporting A fault reporting network is already in existence within the CF. The establishment of at least one person to man the office of primary responsibility for the first year after fielding must be considered here.	2.0	$40,000	$80,000
Complexity of Future Upgrades Future upgrades by military software engineers to a program written by another organization may be difficult.	7.0	$60,000	$420,000
Total rated cost ($CND)			$1,160,000

Table 4. Summary of total rated costs

Alternative	Description	Total Rated Cost ($CND)	Ranking
A	Produce software in-house and provide support to amend it as necessary.	1,274,000	LOW
B	Contract production and support of the software to a civilian agency.	650,000	HIGH
C	Civilian firm produces the software and sells the rights to CF for future support.	1,160,000	MEDIUM

As this project was undertaken to fulfil the academic requirements of a degree programme, alternative one will be adopted as the best course of action. Discussion of future activity concerning the software, and recommended courses of action may be found in Chapter 6.

Detail Design and Development

For a description of the software package designed as part of this project please see Chapter 4.

Chapter 3: Knowledge-Based Expert Systems

Introduction

Artificial Intelligence (AI) is defined as being a collection of computer supported techniques emulating some of the natural capabilities of human beings, primarily problem solving, understanding natural language, vision and robotics, expert systems and knowledge acquisition and knowledge representation methodologies [Savory, 1988].

In the simplest of terms it can be described as a way to make computers think and react like human beings.

The area of AI that is of main concern in this report is that of Expert Systems (ES). These are computer-based programmes which take the place of a human expert. The expert system uses the human expert's knowledge and solves problems that normally only he could, using the same inference procedures. A key characteristic of an ES is that it not only draws conclusions from given data, but can explain to the user what chain of reasoning that it followed to arrive at the answer. In this way the ES is more than just a programme that can compute mathematical equations given appropriate input data (like a calculator). It is capable of explaining its answer and asking for additional information if required.

Because of its reliance on the factual and heuristic knowledge of an expert, expert systems are also known as Knowledge-Based Expert Systems (KBES).

The obvious advantage of an expert system is that the limited knowledge of one expert (or a group of experts) could be put into a format that many users could access at any time. In this way the expert's knowledge and experience is leveraged and the expert himself is able to devote his time to more complex problems or advancement of his field. For this project the primary source of knowledge was CFTO C-30-108-000/MP-001, "First, Second and Third Line Maintenance Instructions".

Elements of a KBES

During the development stage of a KBES, an expert source is required. In most cases this source is a human expert, although documentation, such as books on a specific subject may also be consulted. The developer must be able to extract not only knowledge from the source, but also the way in which this knowledge is used to solve problems (the reasoning).

Once all of the knowledge has been collected, it must be organized into a database of knowledge. Called the knowledge base, this element combines both the facts and heuristic (rule of thumb, or experience-based) information that is required to solve problems in the field of expertise under study.

The inference engine is the control module of the KBES, it processes the information found in the knowledge base and provides solutions based on the information input by a user. One important feature of a KBES is that, for similar domains, the same inference engine can be used to process information from different knowledge bases.

There are two important types of inference engine. Forward chaining engines are data driven, that is they take the data provided by the user and derive as many facts as possible from the knowledge base. Backward chaining engines are goal driven, they are given a goal to work toward and use the knowledge base to this end.

The knowledge base and inference engine would be totally useless without some sort of interface between computer and user. The interface allows the user to consult with the KBES, input data when requested, and receive solutions to the problems posed. It is important to design the user interface so that the user sees little or no difference between consulting a human or computer expert.

Rule-Based Reasoning

Rule based reasoning occurs within a KBES when the knowledge base is in the following form:

IF <condition> THEN <conclusion>

For example, the rule "IF Baffles are Loose THEN Replace the muffler" indicates that if the baffles in a muffler are loose then the muffler should be replaced.

Further refinement of this basic rule is accomplished using the key words ELSE, NOT, AND, and OR. For example, rules can be expanded in the following manner:

IF Diode Test 1 works

THEN Goto Diode Test 2

ELSE Stop Testing and Replace Diode

IF NOT Battery Master Switch is "ON"

THEN Turn on Battery Master Switch

IF Headlights are Dim AND Start Button is Depressed

THEN The Battery is Run Down

IF Headlights are Out OR Headlights are Dim

THEN The Battery Connections are Loose

Forward Chaining And Rule-Based Reasoning

The inference engine interprets rules and applies them to the situation at hand. In a forward chaining scenario, the inference engine will examine all of the rules in the knowledge base and select those which have a condition that is satisfied by the information provided by the user. These rules are termed the conflict set.

Once the conflict set has been determined, the first rule is applied, or fired. The new data resulting from this rule is added to that already provided by the user and any new rules that now have a condition that is met are added to the conflict set. The inference

engine continues in this manner until either it can go no further due to lack of information or it reaches a conclusion. A graphical representation of forward chaining may be found in Figure 7.

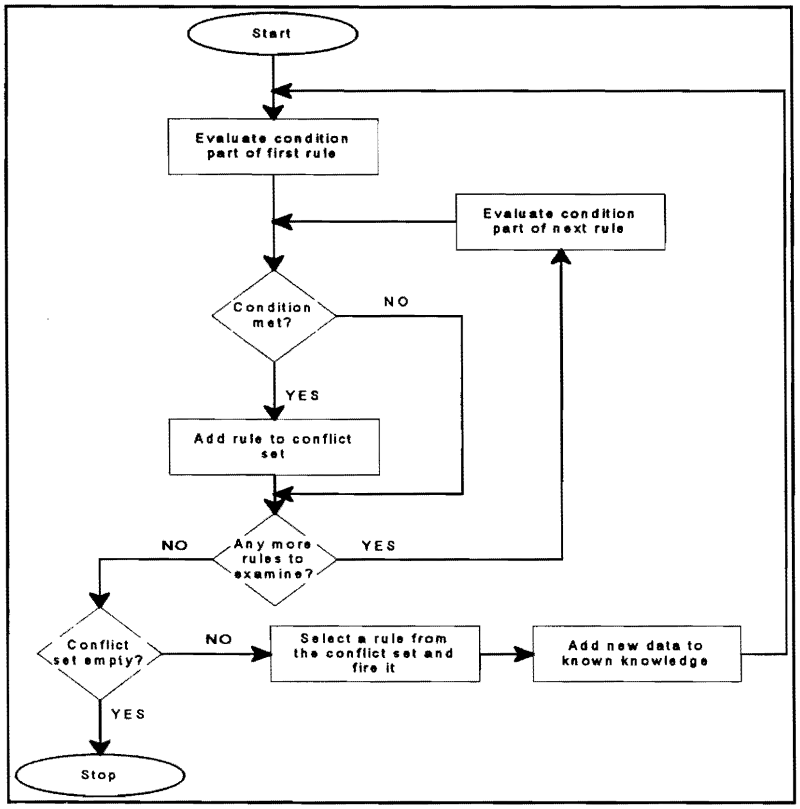

Figure 7. Forward chaining [Hopgood, 1993]

Kappa-PC

Kappa-PC is a commercially produced KBES shell. It contains an inference engine and the tools to build a user interface, the only portion to be added is a knowledge base.

Kappa-PC requires at least a 386 PC compatible computer with a minimum of 8

MB RAM. Depending upon the size of the KBES to be built, Kappa-PC becomes too slow and unwieldy unless used on a 486 with at least 16 MB RAM.

There are three types of forward chaining available in Kappa-PC. Depth First, Best First or Breadth First. All of these techniques follow the general procedures shown in Figure 7, however, the order of rules fired differs among them.

In a breadth-first scenario, rules are added to bottom of the conflict set as they are evaluated and become relevant. The rules are fired beginning with the upper-most rule in the conflict set. In this way, all of the rules that were considered relevant due to the initial data provided by the user are fired first, followed by the rules that became relevant after all of the first rules were fired etc.

Depth-first forward chaining differs in that the rules are added to the top of the conflict set as they become relevant. The rules are still fired in order from top to bottom, but the manner in which newly relevant rules are added to the conflict set means that all of the rules connected to one of the originally supplied conditions tend to be fired before rules containing another original conditions.

Best-first makes use of prioritization of rules. Rules are added to the top of the conflict set in the order of a given importance. As new rules are added to the conflict set, they are slotted into the list also according to their importance.

Experimentation is often the only way to determine which method of forward chaining is most appropriate to the knowledge base at hand.

Chapter 4: The Iltis Diagnostic System

Introduction

The Iltis Diagnostic System (IDS) is a computer based diagnostic software package that uses the Kappa-PC expert system tool as a basis. It is capable of running on IBM PC compatible desktop and laptop computers (486 and higher). IDS requires at least 8 MB of RAM, although 16 MB is highly recommended. A mouse and VGA monitor are also required. Because IDS uses Kappa-PC as a base, it must be run under Windows 3.1 or higher.

IDS uses Kappa-PC to create and store the knowledge base and to conduct forward-chaining operations. The user interface makes use of Kappa-PC images, text files created using WordPerfect and bit-maps. A listing of the various IDS files may be found in Appendix A.

IDS is accessed by loading and running Kappa-PC and opening the file iltis.kal. All other files are entered via the iltis.kal file.

Background

In fulfilment of the academic requirements for CE5064 Knowledge-Based Expert Systems, a prototype Iltis Diagnostic System (pIDS) was designed by Stacey Giberson and Karen Ritchie.

pIDS was designed to be a diagnostic tool for use by trainee mechanics undergoing OJT. It assumed a basic level of knowledge by the user and did not intend to go beyond the diagnostic function by providing repair instructions.

The pIDS basically took diagnostic data from a Canadian Forces Technical Manual and produced it in an expert system format using Kappa-PC. The user was asked to identify a malfunction with the vehicle. pIDS then listed all possible causes of that malfunction and how to correct them.

pIDS considered a very narrow domain. Diagnostic information was available only for the mechanical parts of the engine. All other sub-systems of the Iltis were left undeveloped. In addition, the inference engine of the pIDS was extremely shallow with none of the rules inter-relating.

pIDS assumed that the user was capable of checking for the causes of a malfunction with no instruction regarding exactly how to perform the check. For example, a cause of the vehicle not starting could be low ambient temperature. Any trainee mechanic would be capable of determining that information with no additional instruction. However, another cause could be that the starter commutator is worn. To verify if this condition is present, a user must first remove the starter from the vehicle and disassemble it.

Following on from pIDS, the Iltis Diagnostic System has been designed to provide a diagnostic and limited repair capability to any level of mechanic. IDS continues to assume a basic level of user knowledge and still lists all of the possible malfunctions within

a specified vehicle sub-system. Where IDS expands on the prototype, however, is in the breadth and depth of information provided.

Three of the five major Iltis sub-systems have been fully developed for diagnostic purposes (Engine, Power Train and Body & Chassis Suspension). In addition, repair instructions have been added for one sub-system (Body and Chassis Suspension). It should be noted that this repair capability goes beyond the classic definition of fault diagnosis and has been added purely for the purposes of demonstrating the future capability of a system such as IDS.

The IDS Knowledge Base

As stated previously, the major source of knowledge for IDS was a CFTO detailing the maintenance procedures for all levels of maintenance of the Iltis. A secondary source of knowledge was a trained CF vehicle mechanic, Warrant Officer D.R. Peddle. Mr Peddle provided clarification regarding the capabilities of a trainee mechanic, as well as initial testing and evaluation of the programme.

The CFTO lists all troubleshooting data in table format. Malfunctions are listed under the sub-system in which they would occur. Probable causes of a malfunction as well as the corrective action required to repair it is given. Detailed instructions regarding how to test for probable causes, remove and dismantle components, and repair items are found throughout the book. A sample of the CFTO troubleshooting guide may be found in Figure 8.

The IDS knowledge base was developed directly from the troubleshooting guides of the CFTO. Each probable cause and the related corrective action are written as a Kappa-PC rule. Rules are grouped according to the vehicle sub-system to which they relate. The rule numbering system follows the troubleshooting guide index to ensure quick reference between the two. An example of some Kappa-PC rules with an equivalent translation into plain text is depicted in Figure 9.

Rules generally interact by enabling other rules to fire and being enabled by preceding rules. An example of this can be seen by studying the rules in Figure 9. If the value bright_and_solenoid_clicks is entered for the value of Light:State2 and it is also known that the Starter Runs, then the following rule will be selected by Kappa-PC for addition to the conflict set.

```
MakeRule( R2951c4, [],
        Light:State2 #= bright_and_solenoid_clicks,
        {
        SetValue( Solenoid, Clicks, yes );
        Gotos2951c4( );
        } );
SetRulePriority(R2951c4,1);
```

Once rule R2951c4 has fired, the resulting new knowledge (that the Solenoid Clicks) will

enable Kappa-PC to add another rule to the conflict set.

```
MakeRule( R2951c5, [],
        Starter:Runs #= yes And Solenoid:Clicks #= yes,
        {
        PostMessage( "Replace solenoid." );
        SetValue( Variable, varlight, TRUE );
        } );
SetRulePriority(R2951c5,1);
```

This process will continue until there are no more rules whose conditions have been met. A manner of visually depicting the rule relationships is through the use of a rule dependence network. Figure 10 shows such a network for the rules contained in Figure 9.

The sub-systems and components of the Iltis are broken down into a network of objects, instances and slots. The objects and instances represent sub-systems or components of the Iltis while the slots represent the various states of repair of the components. A partial object hierarchy is shown in Figure 11.

ITEM NO	MALFUNCTION	PROBABLE CAUSE/TEST	CORRECTIVE ACTION
1.	**ENGINE WON'T RUN OR RUNS INTERMITTENTLY**	a. Primary ignition fault.	a. Test primary system as described in Section 8.
		b. Secondary ignition fault.	b. Inspect secondary system for carbon tracking, cracks, grounding, continuity and insulation quality.
		c. Fuel system fault.	c. See fuel system troubleshooting.
2.	**ENGINE RUNS BUT MISSES**	a. Defective spark plug.	a. Check plugs, clean and gap or replace.
		b. Defective distributor cap, high tension cables or rotor.	b. Inspect cap for carbon tracking and cracks. Check continuity and insulation of cables. Inspect the rotor. Replace defective components.
		c. Stuck valve, broken valve spring, burnt valve, flat spot on cam lobe or blown head gasket.	c. Remove valve cover or head if necessary. Repair or replace faulty component.
		d. Timing belt has jumped.	d. Adjust engine timing.
		e. High voltage leak across coil.	e. Replace the coil.
		f. Ballast resistors.	f. Replace if misfire is unexplained. Check ignition system and repair or replace faulty components.
3.	**ENGINE LACKS POWER (HOT OR COLD)**	a. Ignition system problems.	a. Check ignition system and repair or replace faulty components.

Figure 8. Sample CFTO troubleshooting guide [C-30-108-000/MP-001]

Kappa-PC Rule	Translation
MakeRule(R2951, [], Variable:varlight #= FALSE, PostMessage("You have not entered enough data. Please try again."));	Rule R2951 IF variable called "varlight" is FALSE THEN display message in parentheses. Rule R2951 has a priority of 0.
MakeRule(R2951c4, [], Light:State2 #= bright_and_solenoid_clicks, {SetValue(Solenoid, Clicks, yes); Gotos2951c4( );}); SetRulePriority(R2951c4,1);	Rule R2951c4 IF State2 = bright and solenoid clicks THEN Solenoid Clicks AND Goto screen s2951c4. Rule R2951c4 has a priority of 1.
MakeRule(R2951c5, [], Starter:Runs #= yes And Solenoid:Clicks #= yes, {PostMessage("Replace solenoid."); SetValue(Variable, varlight, TRUE); }); SetRulePriority(R2951c5,1);	Rule R2951c5 IF Starter Runs AND Solenoid Clicks THEN display message in parentheses AND variable called "varlight" is given the value TRUE. Rule R2951c5 has a priority of 1.
MakeRule(R2951c6, [], Starter:Runs #= yes And Solenoid:Clicks #= no, { Gotos2951c5( ); SetValue(Variable, varlight, TRUE); }); SetRulePriority(R2951c6,1);	Rule R2951c6 IF Starter Runs AND Solenoid does not Click THEN Goto user screen s2951c5 AND variable called "varlight" is given the value TRUE Rule R2951c6 has a priority of 1.
MakeRule(RSol1, [], AccesSwitch:Faulty #= TRUE, {PostMessage("Repair or replace component."); SetValue(Variable, varlight, TRUE); }); SetRulePriority(RSol1,1);	Rule RSol1 IF AccessorySwitch is Faulty THEN display message in parentheses AND variable called "varlight" is given the value TRUE. Rule RSol1 has a priority of 1.

Figure 9. Sample rules

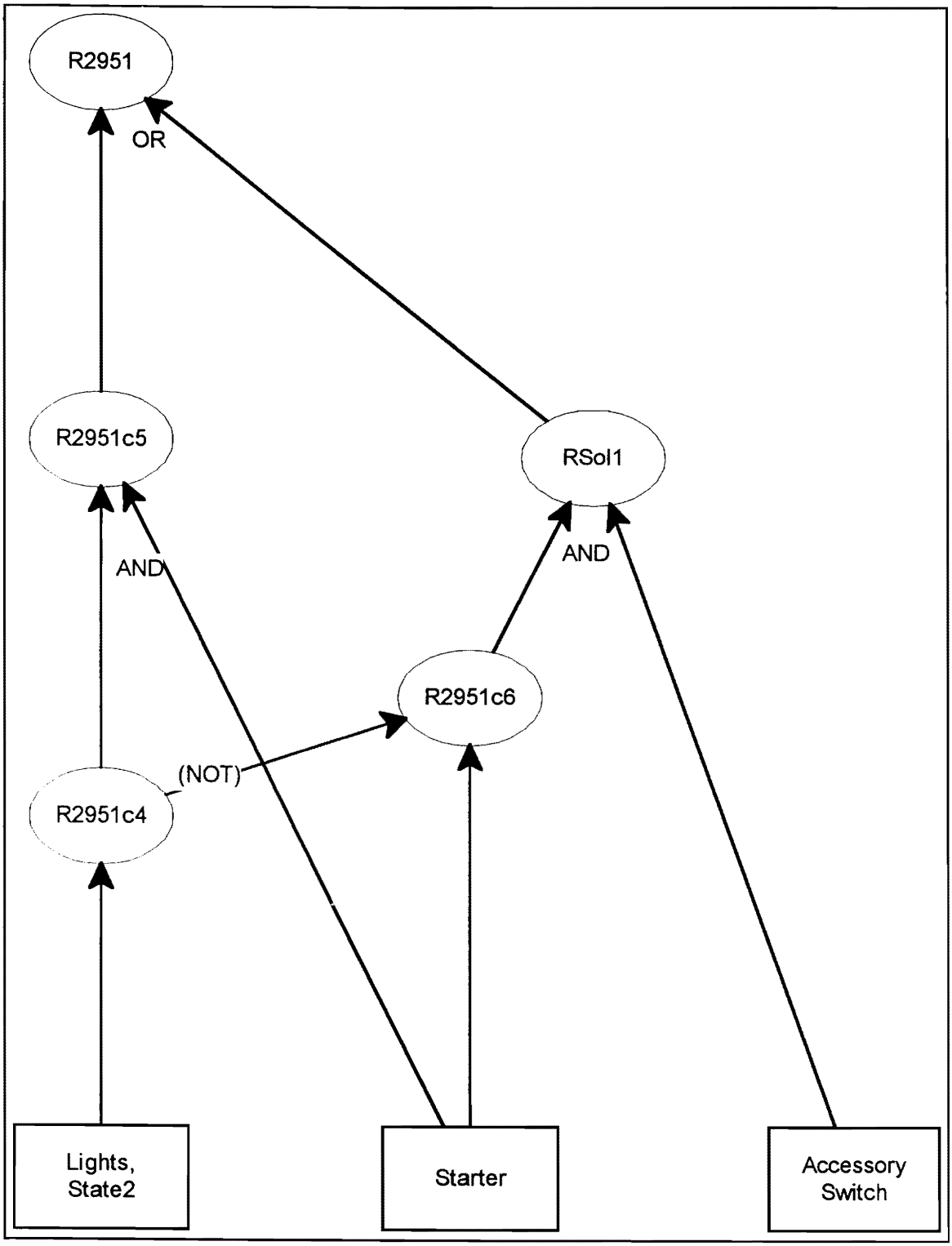

Figure 10. Partial rule dependence network

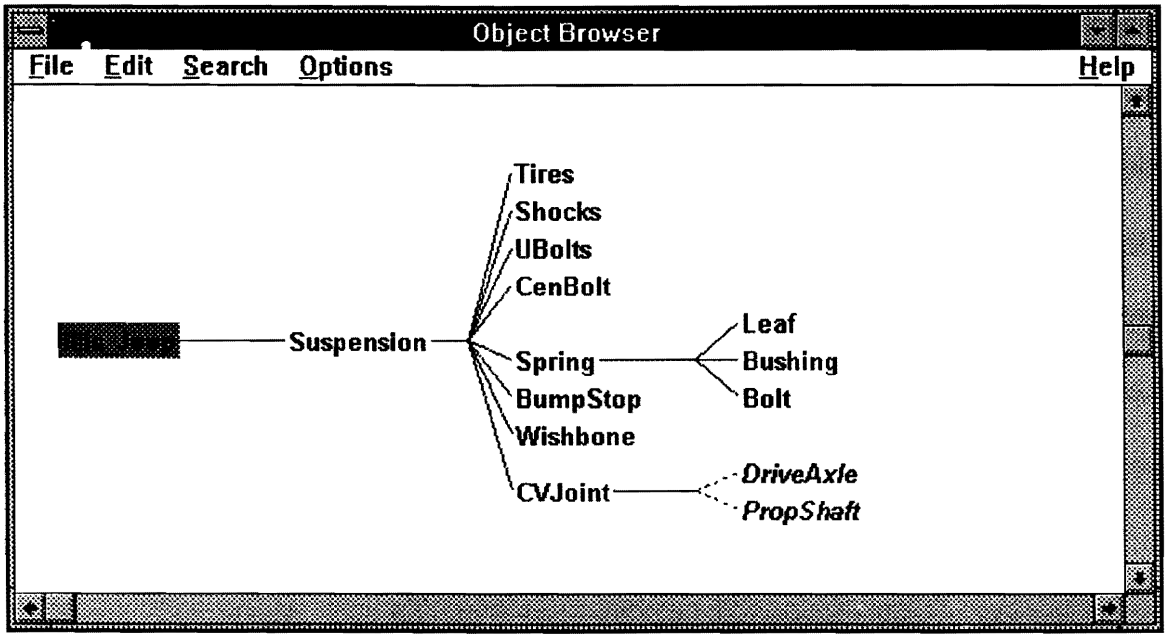

Figure 11. Partial object hierarchy

IDS Forward Chaining

A typical consultation with IDS will follow this sequence of events:

a. User descends through various levels of screens to find the malfunction that is present in the vehicle;

b. Faults that could cause the specified malfunction are identified;

c. IDS forward chains through the rules contained in the knowledge base;

d. The solution is displayed.

In Chapter 3, forward chaining in rule-based systems was discussed. IDS uses the forward chaining mechanism to arrive at a solution to the vehicle diagnosis, specifically, the BEST FIRST method is employed.

All of the IDS rules have been assigned a priority relative to each other from zero

to one. By default, a rule priority is set as zero.

Using the sample rules shown in Figure 9, one can see that rule R2951c5 has a priority of one while rule R2951 has no assigned priority meaning that the priority is set as the default (zero). In this case the Variable varlight has a default value of FALSE which is preset by IDS upon commencement of any consultation.

If the information that the Starter Runs and the Solenoid Clicks is entered into IDS, then the two rules below will be added to the conflict set for forward chaining.

```
MakeRule( R2951, [],
        Variable:varlight #= FALSE,
        PostMessage( "You have not entered enough data. Please try again." ));
MakeRule( R2951c5, [],
        Starter:Runs #= yes And Solenoid:Clicks #= yes,
        {
        PostMessage( "Replace solenoid." );
        SetValue( Variable, varlight, TRUE );
        } );
SetRulePriority(R2951c5,1);
```

If regular forward chaining is invoked, then the rules will fire in order of appearance and the following will occur:

a. "You have not entered enough data. Please try again." will be displayed;

b. " Replace solenoid" will be displayed; and

c. varlight will be set to the value TRUE.

This will obviously cause confusion to the user since he is being told that he has not entered enough data and then told to replace the solenoid.

BEST FIRST forward chaining selects the rule with the higher priority to be fired first. In this situation, the second rule will fire first, with the following result:

a. varlight will be set to the value TRUE; and

b. "Replace solenoid." will be displayed.

Since the variable varlight is now equal to the value TRUE, rule R2951 will not fire.

User Interface

The user interface is a critical element of any computer programme, it is important to try and design an interface that is familiar to the user in order to decrease training time and overcome any potential resistance to the programme.

Kappa-PC provides a myriad of visual images to communicate with the user. IDS uses a number of these images to enable the user to move to different levels of diagnosis, input information, view instructions for component repair or replacement, and exit the programme.

IDS opens with an introduction screen which gives the user the choice of selecting a malfunction from a table of contents or going directly to the vehicle sub-system in which the malfunction is occurring. This flexibility enables the more knowledgeable user

to go directly to the problem and the less experienced trainee to consult a table of contents.

Once the malfunction has been found. IDS will present a listing of all of the possible causes of this malfunction on the screen. In some cases the fault will be listed and the user will have the opportunity to identify to the programme whether that fault is present or not. Faults that are more complex, or require detailed testing or disassembly of components to isolate are also listed but the user than has the opportunity to investigate these further and learn how to isolate them.

Either way, when the fault is identified by the user as being present, a forward chaining process commences which will identify the repair action necessary to correct the fault. If an inadequate amount of information has been entered, the forward chaining process will result in a request for more data.

Due to the size of the programme, the user will actually be transferred between different files during his consultation. This is an invisible process but important to the speed of the consultation. Also, the user has the ability to step backwards through the screens if he wishes, or exit the programme completely at anytime.

Programme Validation

A validation test was performed by Mr Peddle. The validation goals were:

a. to confirm that IDS met the aim established in Chapter one of this report;

b. to comment on the level to which this programme will integrate with existing

systems; and

c. to suggest changes to the user interface.

Mr Peddle approached the validation by assuming that IDS would be available for use by mechanics in the vicinity of their repair bays. He used a number of mock diagnostic scenarios to explore the different areas of IDS and form his conclusions.

Overall, Mr Peddle's comments were positive, the majority of his concerns focussed around the user interface rather than integration or confirmation of the project aim.

Specifically, IDS is judged to accomplish the stated aim of providing a diagnostic tool for use by OJT personnel as well as trained mechanics. The programme closely follows the same process used by vehicle technicians at this time but gives an added feature of quick and easy access to schematics and an easier search system than is possible with the CFTOs. Mr Peddle was enthusiastic about the usefulness of this programme for supervisors as well as trainees, especially the fact that all possible causes of malfunctions are provided in list format and that diagrams are provided.

The user interface deficiencies identified during the validation have been addressed.

Areas For Future Improvement

At the commencement of this project, Kappa-PC was judged to be an adequate tool for the task of building a diagnostic system. Unfortunately there are limitations to the

abilities of the programme and the hardware that was available to me during production of IDS. These limitations have resulted in some slowing of the programme to lags of up to 3 seconds due to memory restrictions and sheer size of the data involved.

In order to be acceptable to the user, IDS must be at least as quick as any commercial software package. There are two options available to increase the programme speed - better software or better hardware.

To be a viable alternative for use within a workshop scenario, additional hardware is not an option. Most shops are equipped with 486 or Pentium computers, a programme that requires a large server is not realistic. I still feel that an expert system tool is an excellent vehicle for this type of project, however a more powerful version of Kappa-PC or similar system is required.

Chapter 5: Sample Consultations

This chapter contains two sample consultations from IDS. The screens are printed here in black and white, they are in colour on the computer screen and the resolution is much better.

Starter Is Not Operating When The Start Button Is Depressed

In this scenario, the technician is told that the starter is not working when the start button is depressed. He opens IDS and sees the initial screen (Fig 12).

The technician is unsure whether the starter is considered part of the engine or electrical system so he decides to consult the Table of Contents (Fig 13). Although I am unable to show this, if the technician inputs the index number 2951 for starter problems he would be given the message to access the Engine sub-system.

Returning to the initial screen, the technician presses Engine and is moved to the next screen (Fig 14) which lists the sub-systems within the engine. In this case he selects Starter and is presented with the major malfunctions that could occur with the starter (Fig 15).

Selecting, Starter Does Not Operate When Start Button Depressed, the technician now sees the exact faults that could cause the malfunction (Fig 16). Assuming that he knows that the accessory and battery master switch are ON, he decides to investigate any

electrical problems that could exist. The starter electrical test screen (Fig 17) instructs the technician. He finds that the lights are bright but the solenoid clicks when he presses the start button. Once he enters this information, IDS automatically moves to the next screen (Fig 18), a solenoid test. These instructions include a schematic diagram (Fig 19) which is accessed by pressing the related button.

After having completed the test, the technician notes that the starter does run. He inputs this data and is told to replace the solenoid (Fig 20). This would conclude the diagnosis.

Excessive Noise From The Suspension When Driving

In this case, the driver has complained that the vehicle is very noisy (clanking) when driving. The technician knows that this is probably a suspension problem so he accesses that sub-system directly from the initial screen (Fig 21) and moves to Excessive Noise When Driving in the next screen (Fig 22).

All of the probable causes of excessive noise are listed in the next session screen (Fig 23). The technician quickly checks for all of the faults and notices that there is a worn bushing on the front spring, which he indicates, and is instructed to remove the entire spring assembly and replace the bushing (Fig 24). Once he acknowledges the instruction, IDS adds two access buttons which enable the technician to learn how to remove the spring assembly (Fig 25).

Accessing the front spring assembly removal screen, the technician is given

detailed instructions including schematic diagrams (Fig 26). He then can access a screen that tells him how to replace the bushings (Fig 27).

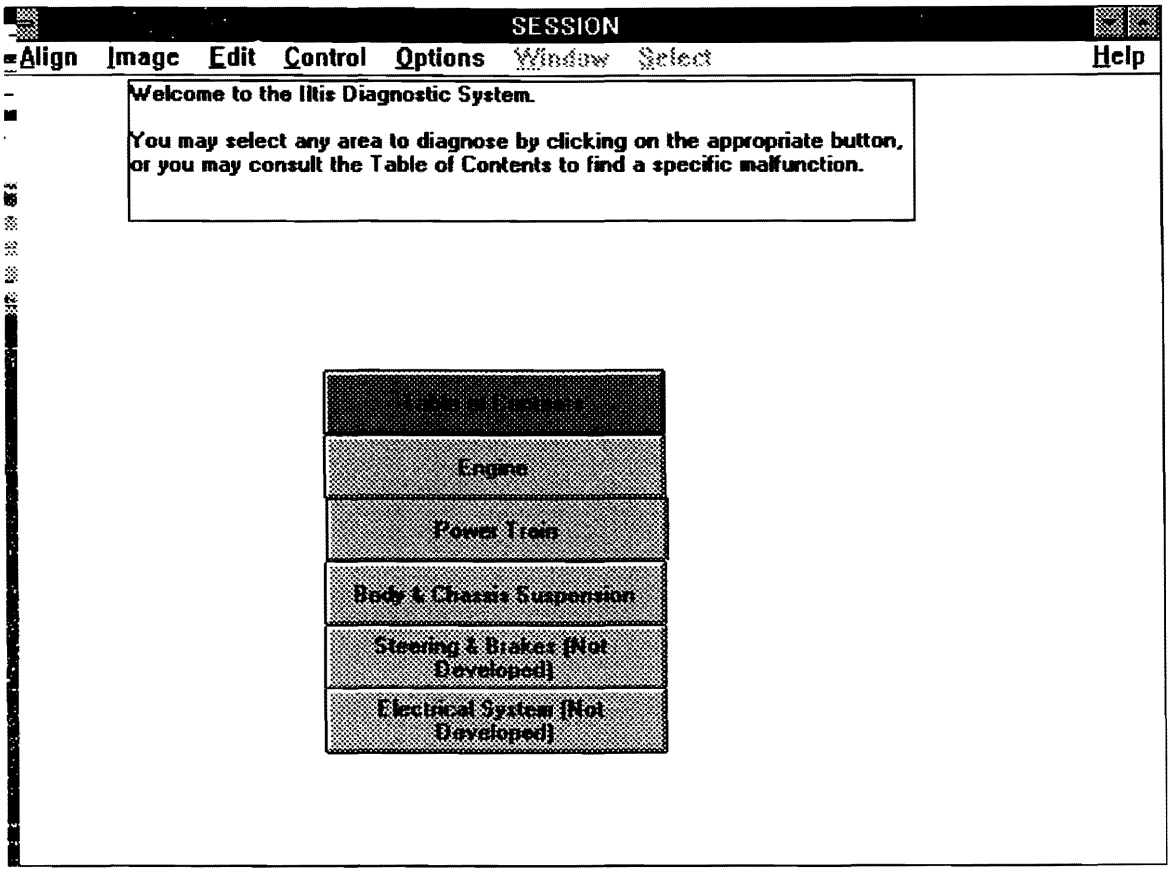

Figure 12. Sample consultation 1 - screen one

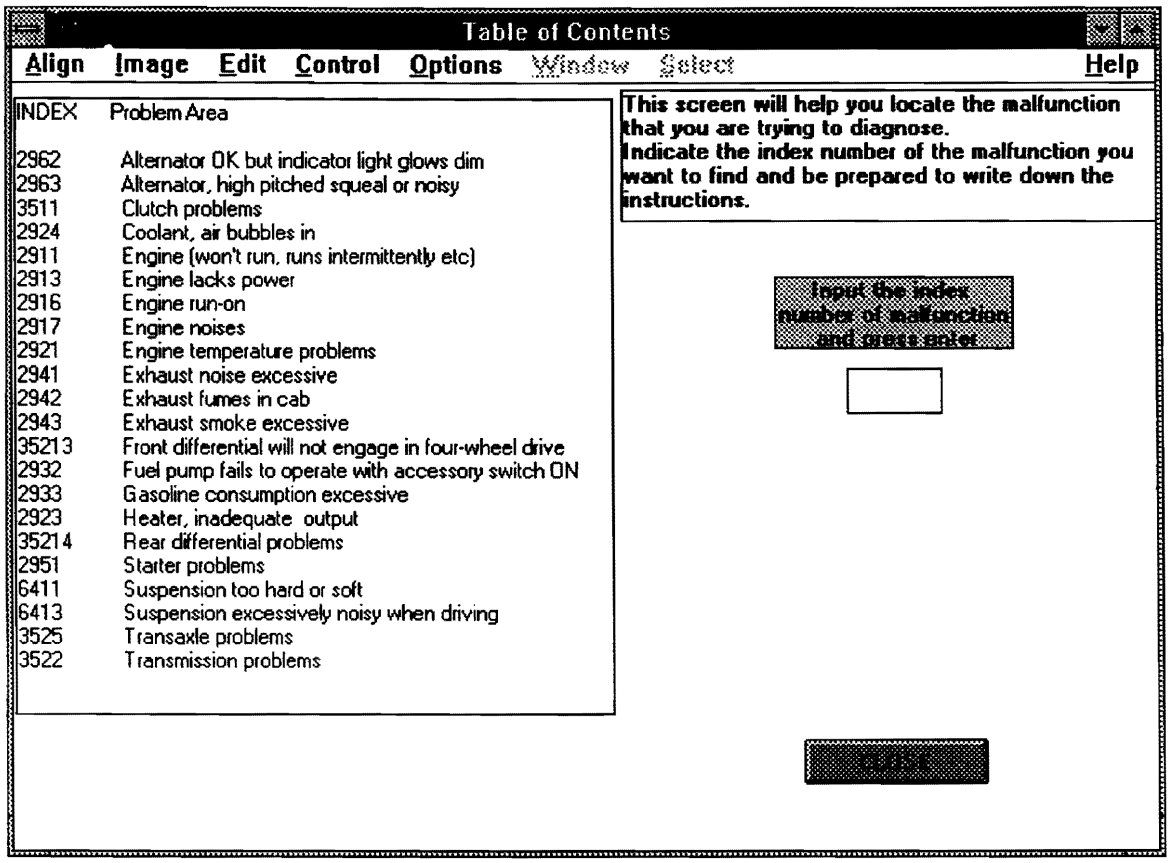

Figure 13. Sample consultation 1 - screen two

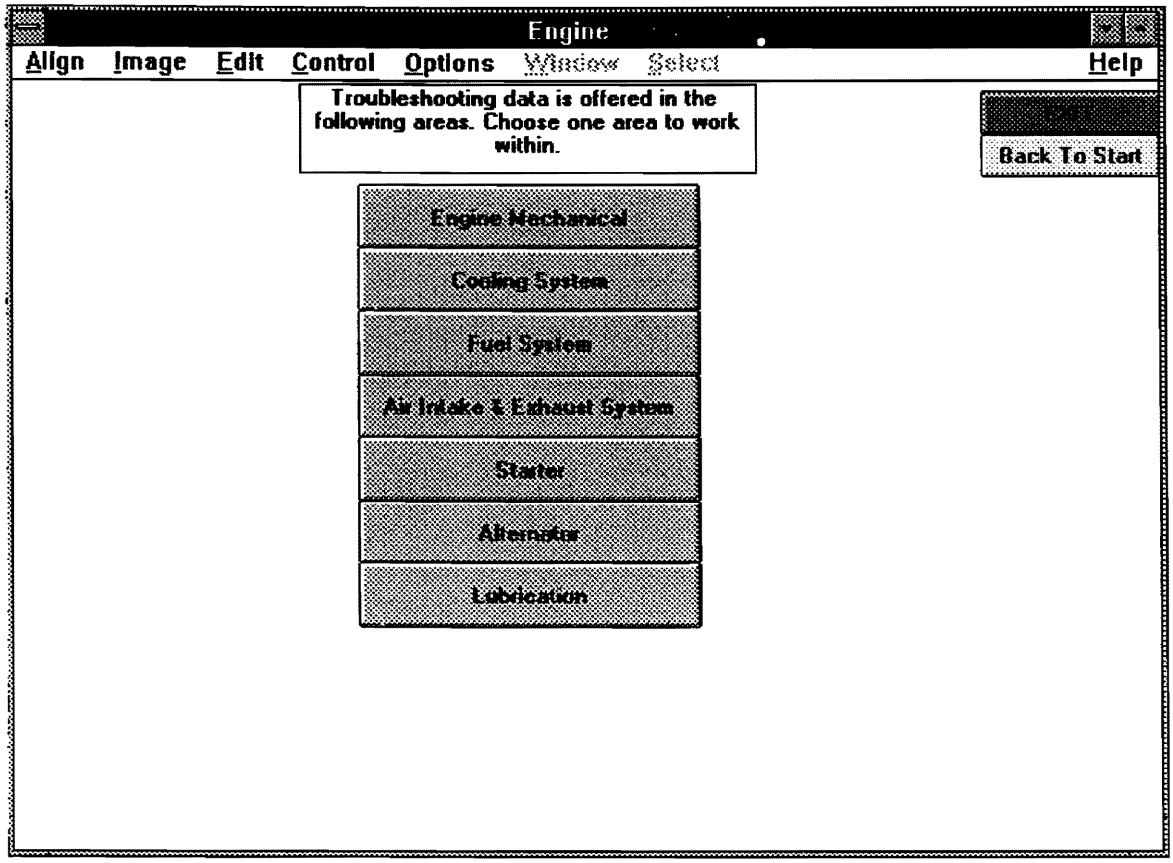

Figure 14. Sample consultation 1 - screen three

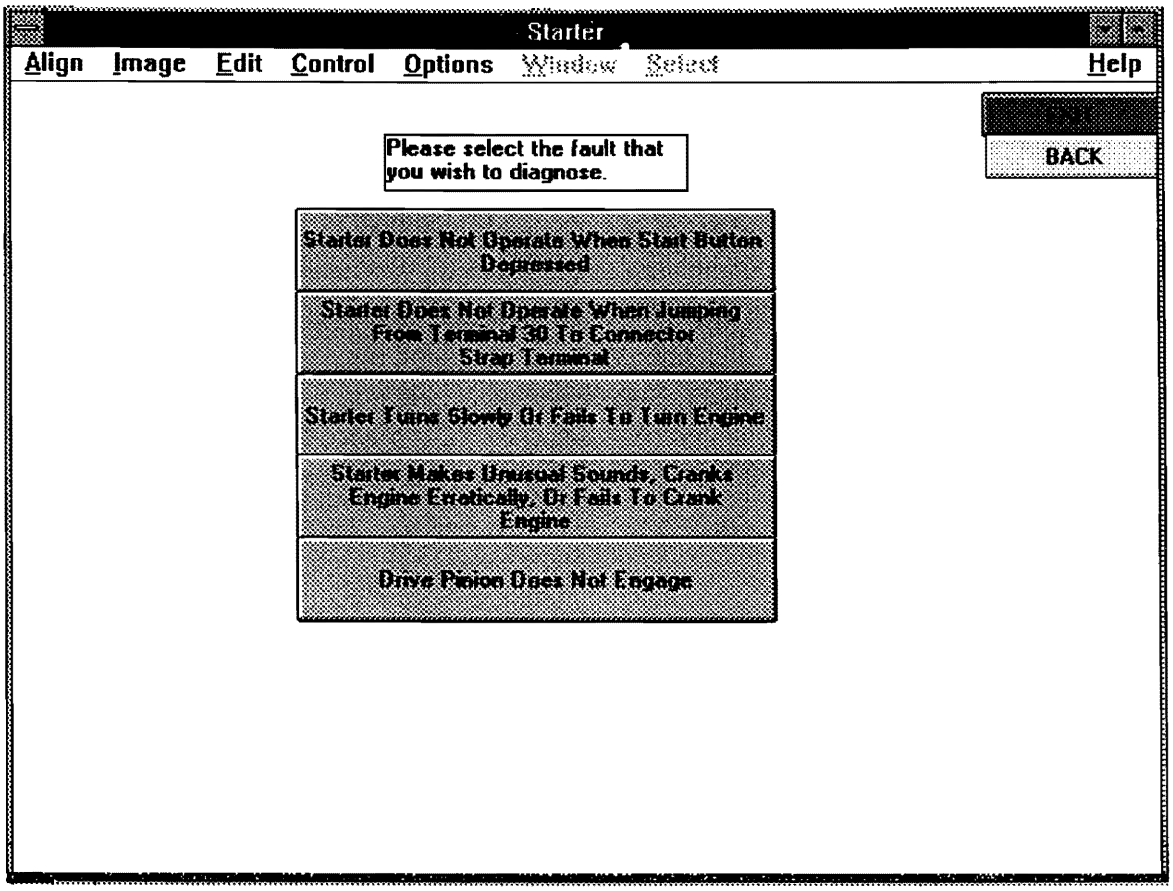

Figure 15. Sample consultation 1 - screen four

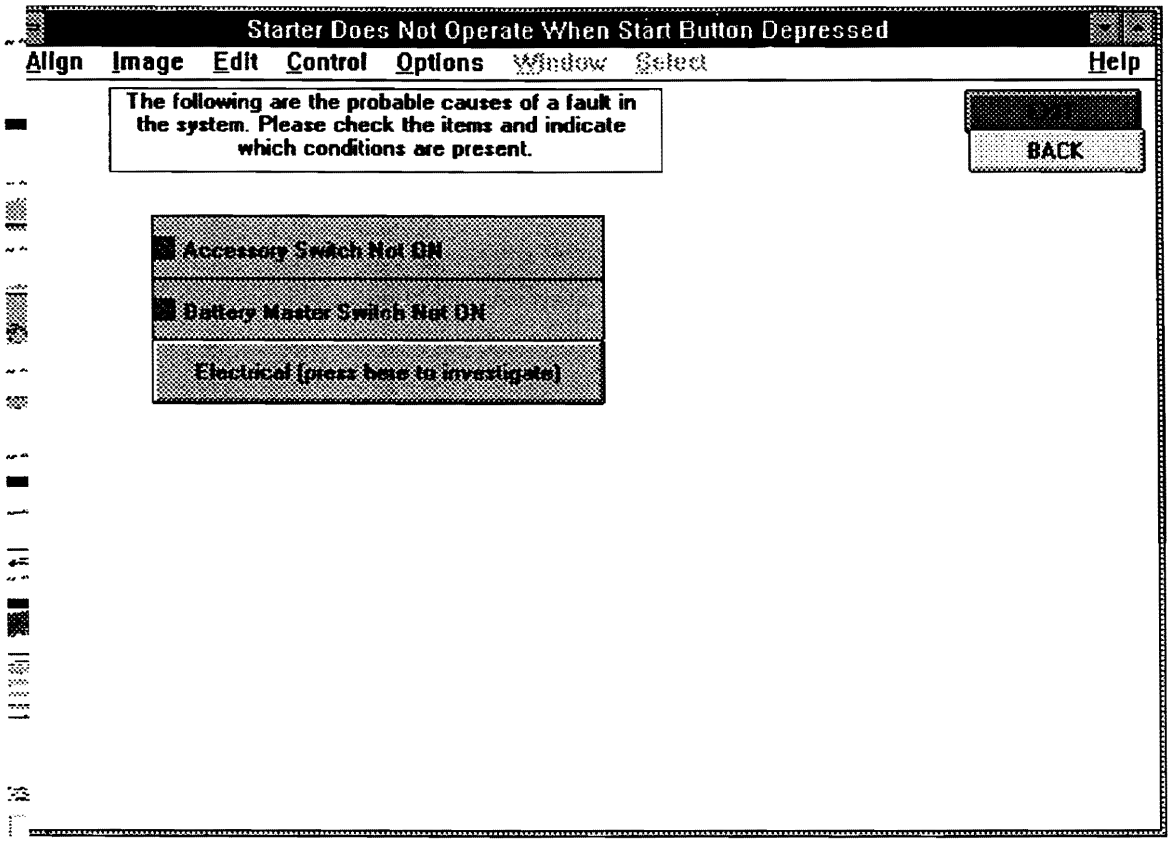

Figure 16. Sample consultation 1 - screen five

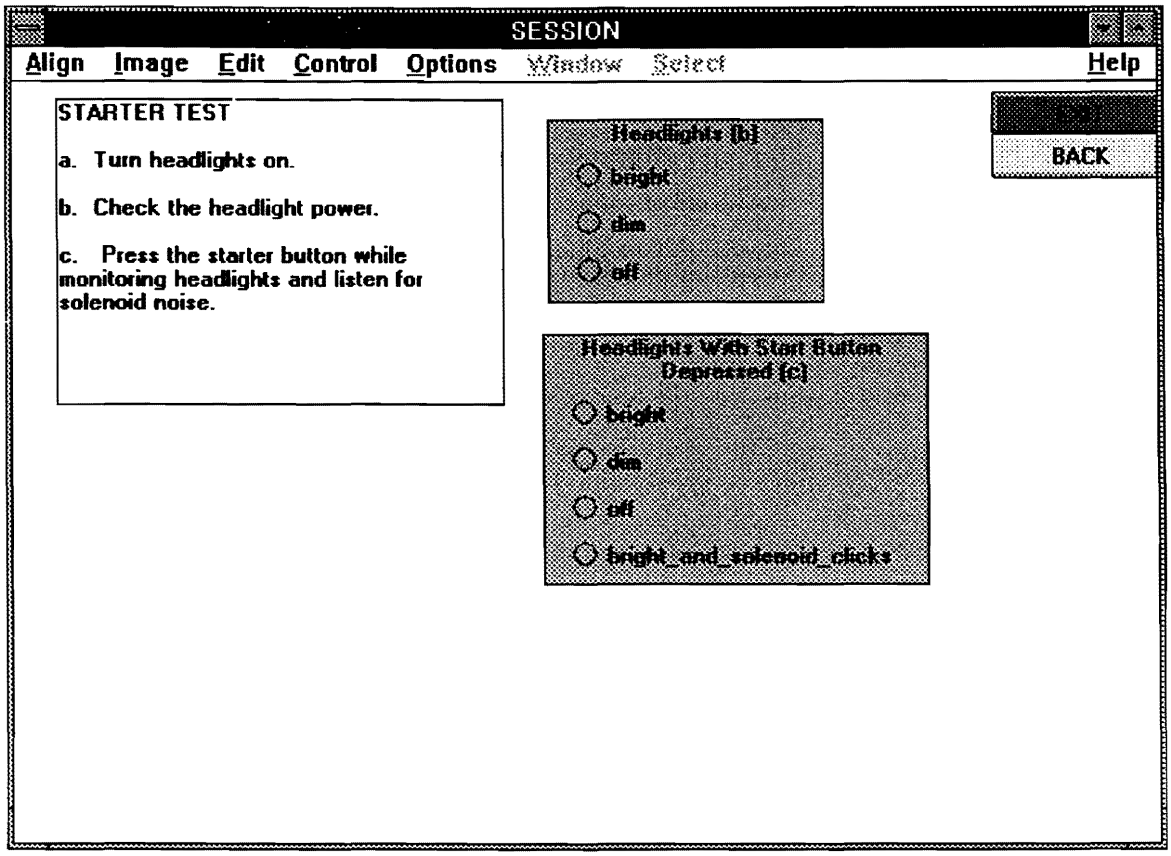

Figure 17. Sample consultation 1 - screen six

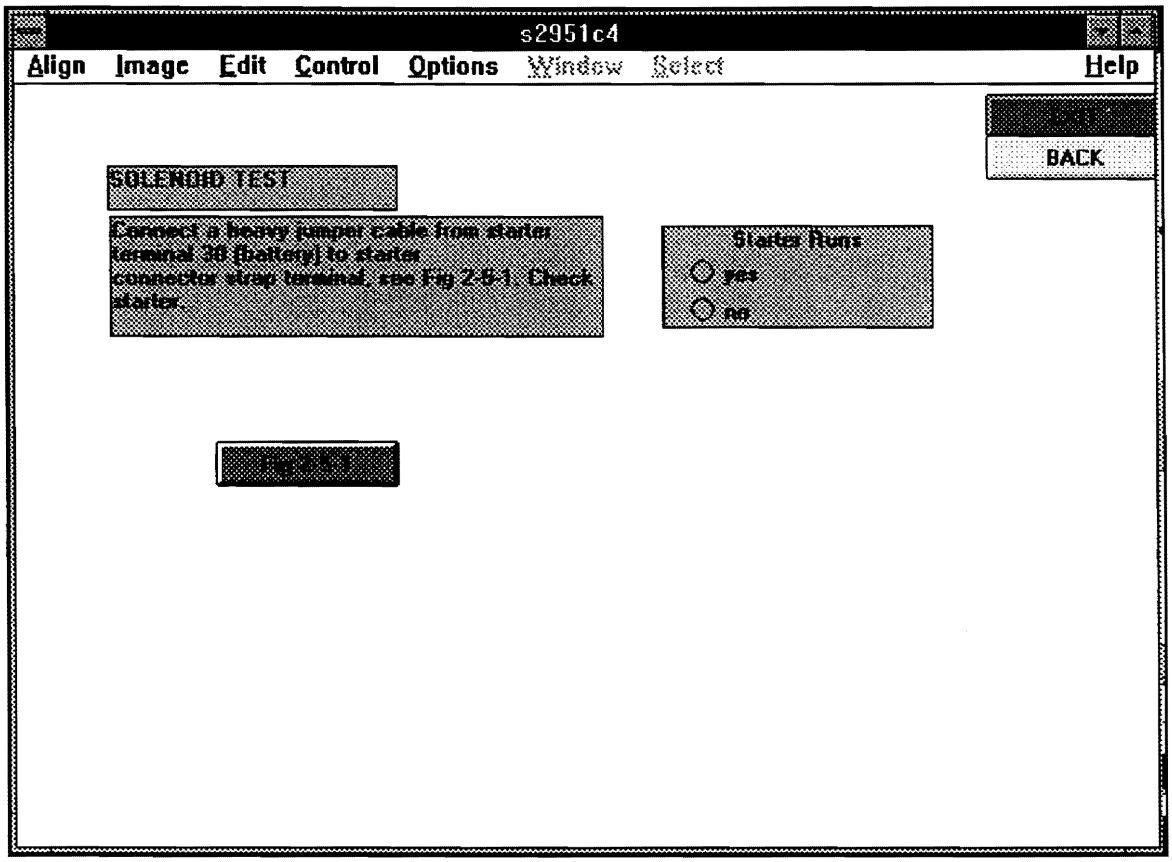

Figure 18. Sample consultation 1 - screen seven

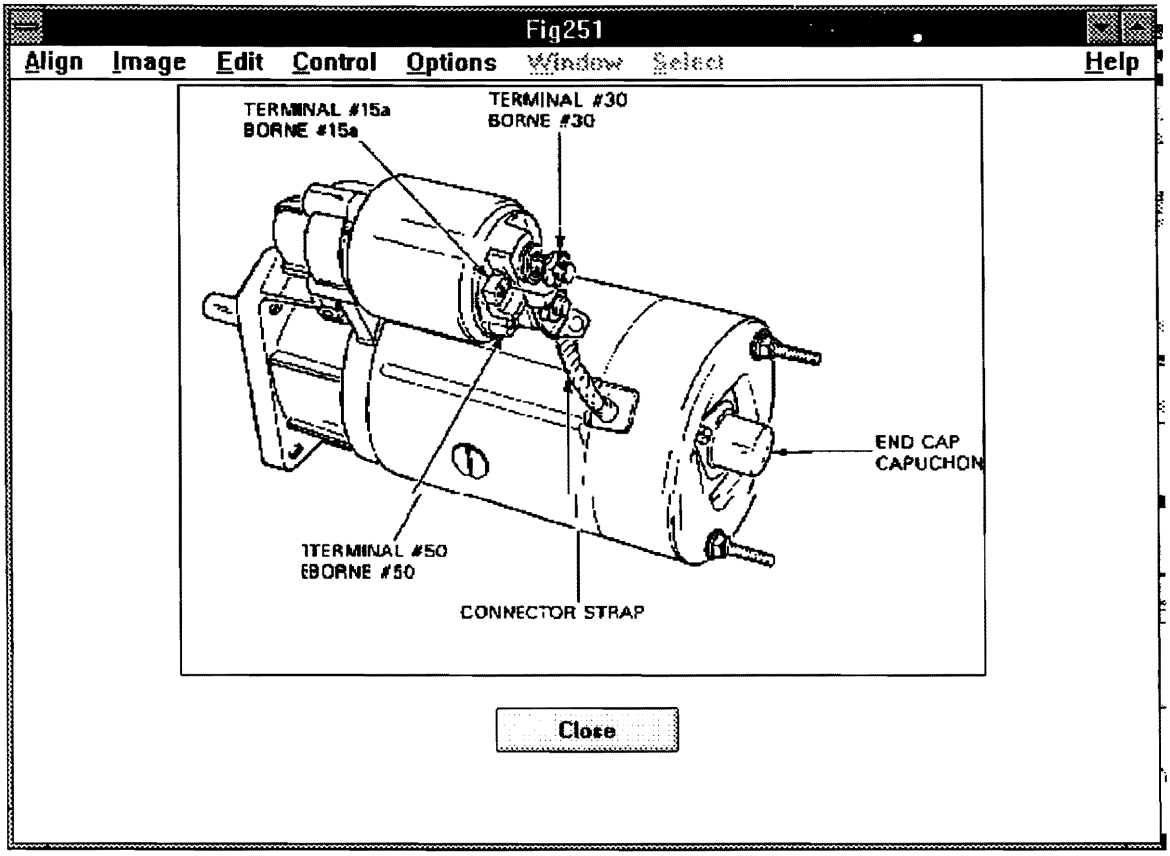

Figure 19. Sample consultation 1 - screen eight

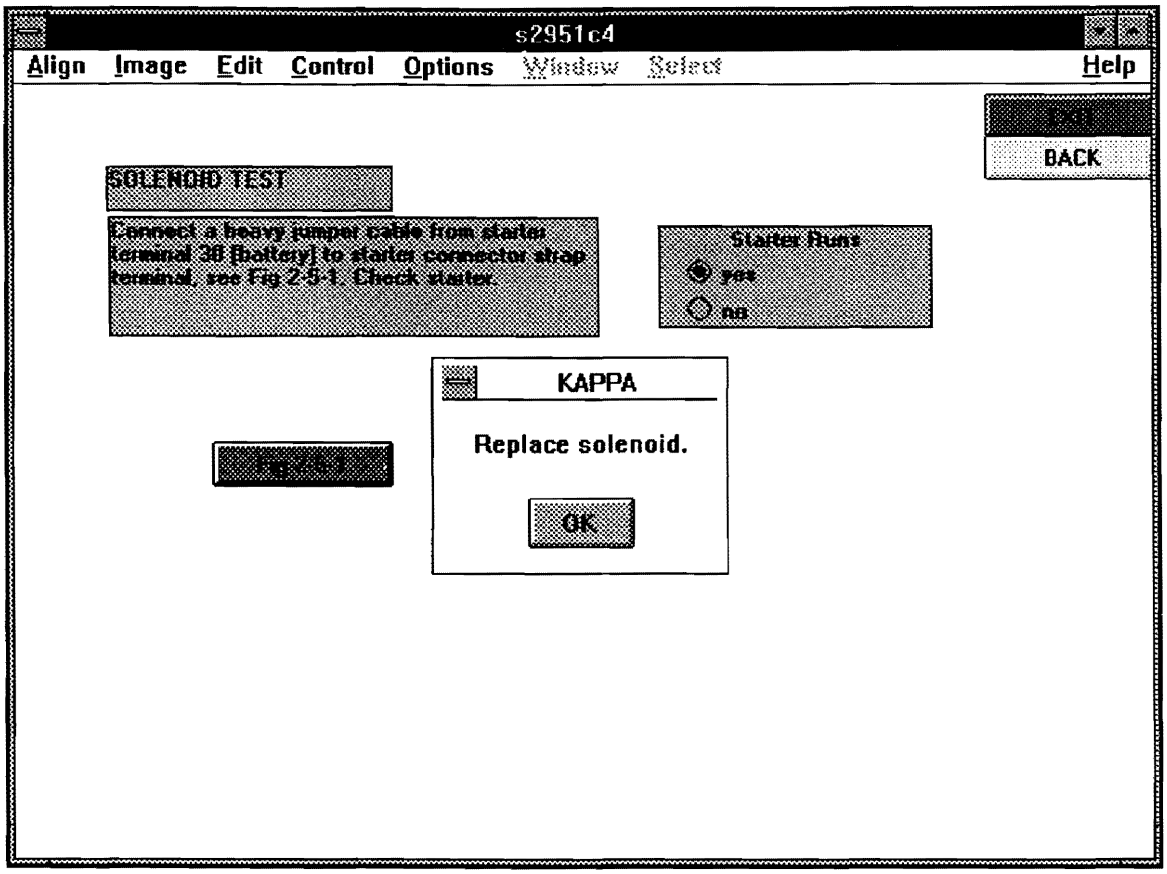

Figure 20. Sample consultation 1 - screen nine

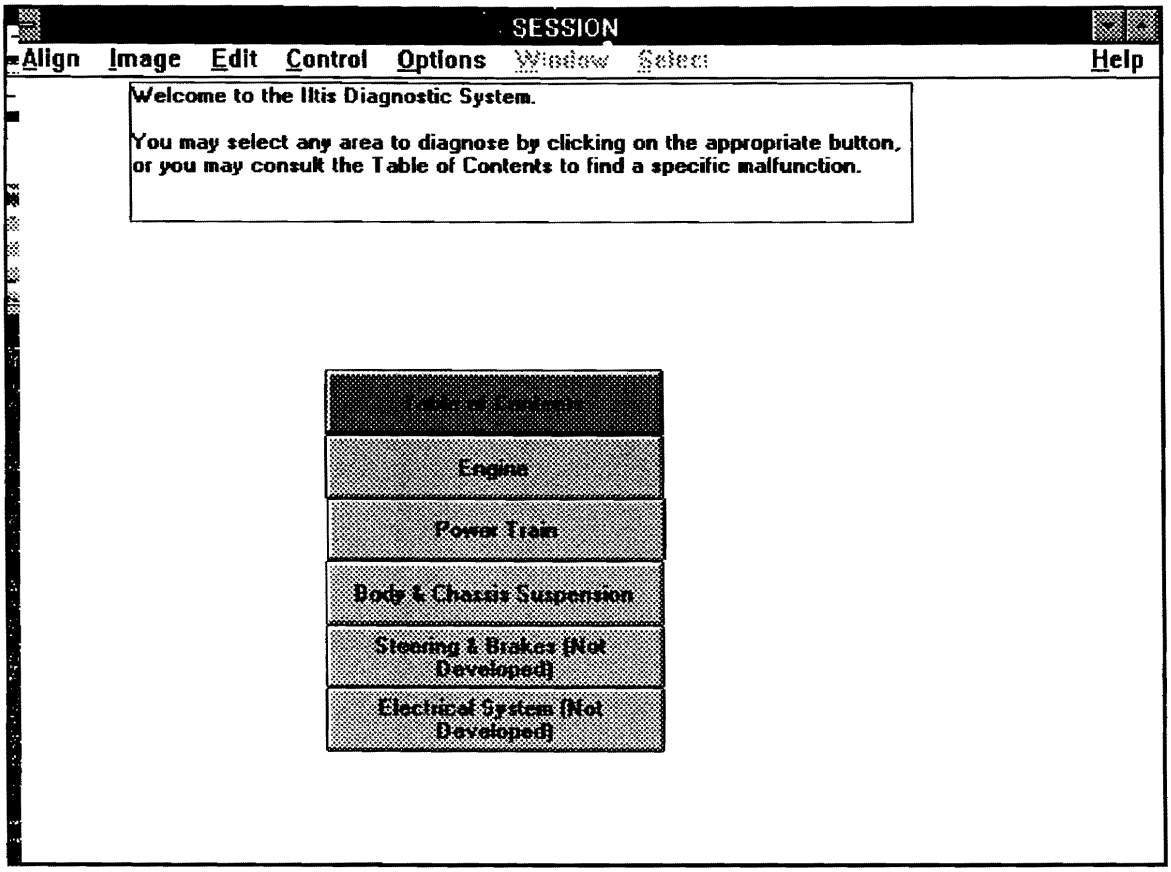

Figure 21. Sample consultation 2 - screen one

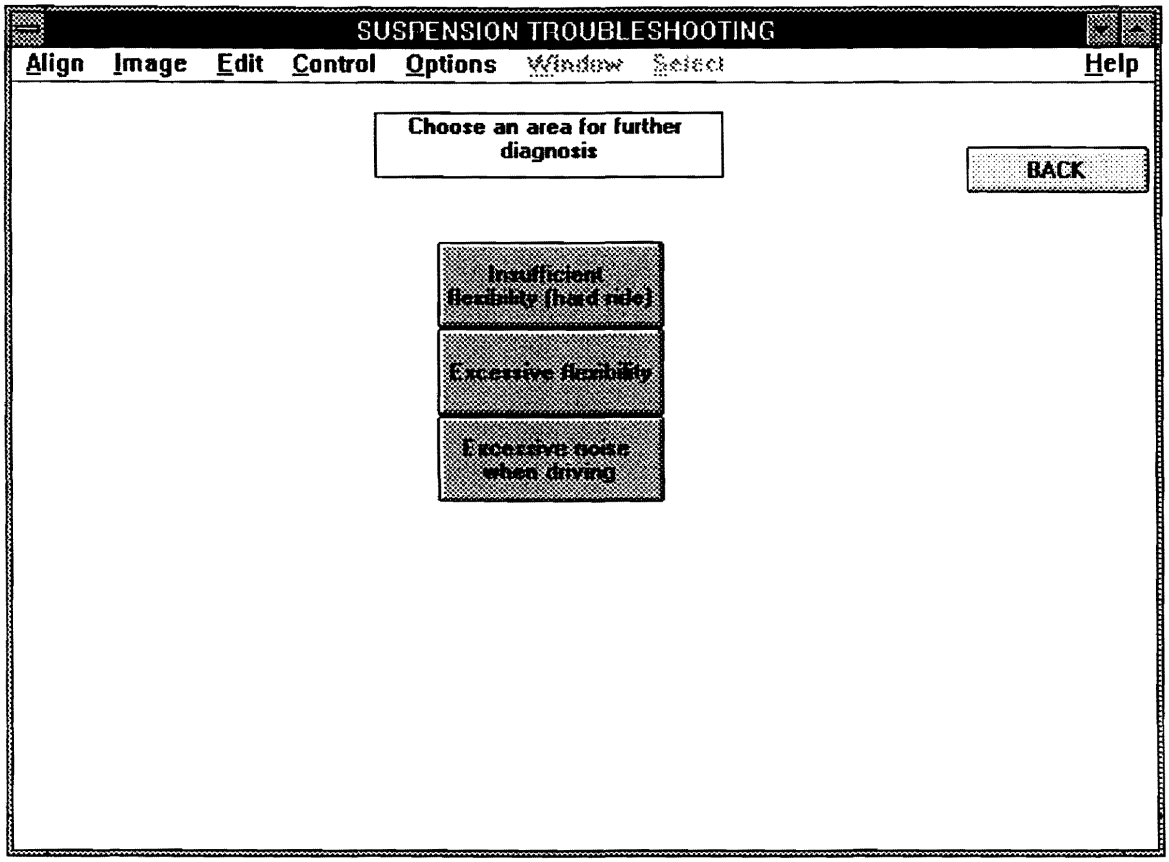

Figure 22. Sample consultation 2 - screen two

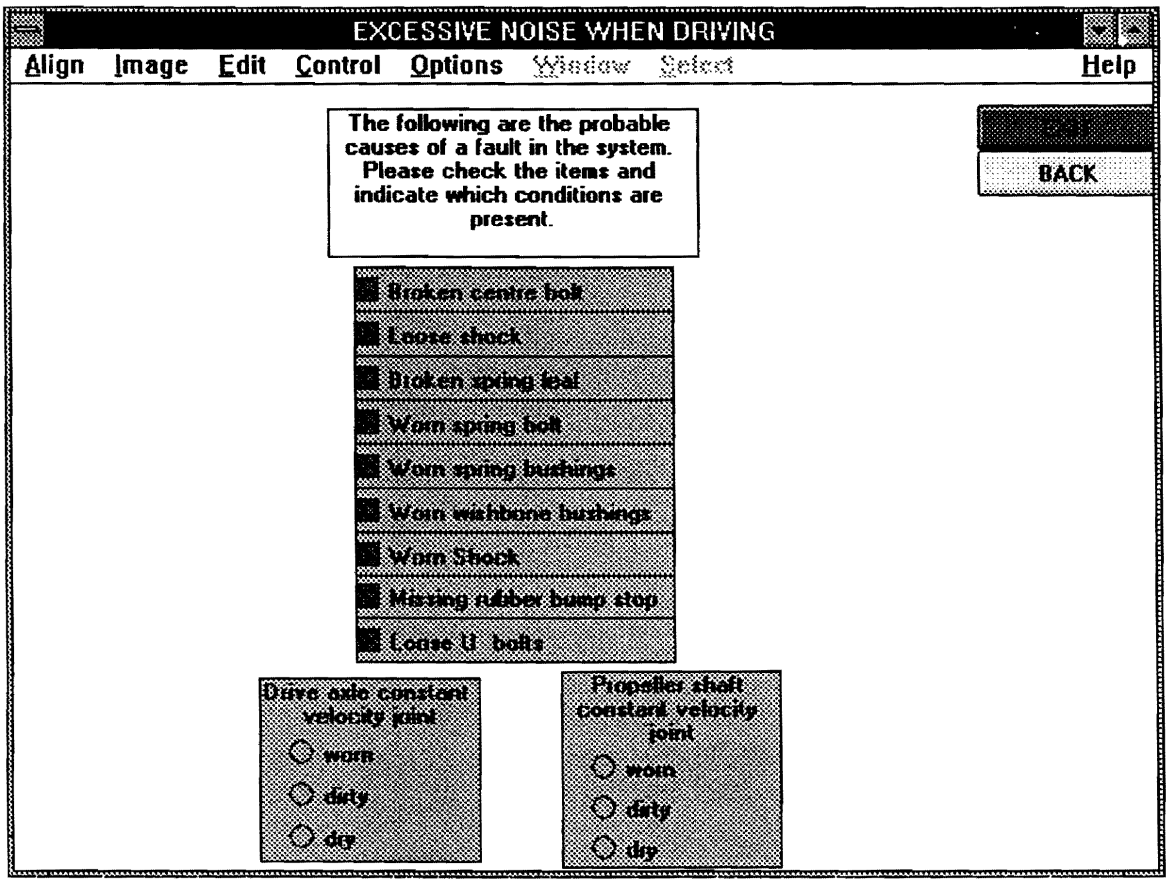

Figure 23. Sample consultation 2 - screen three

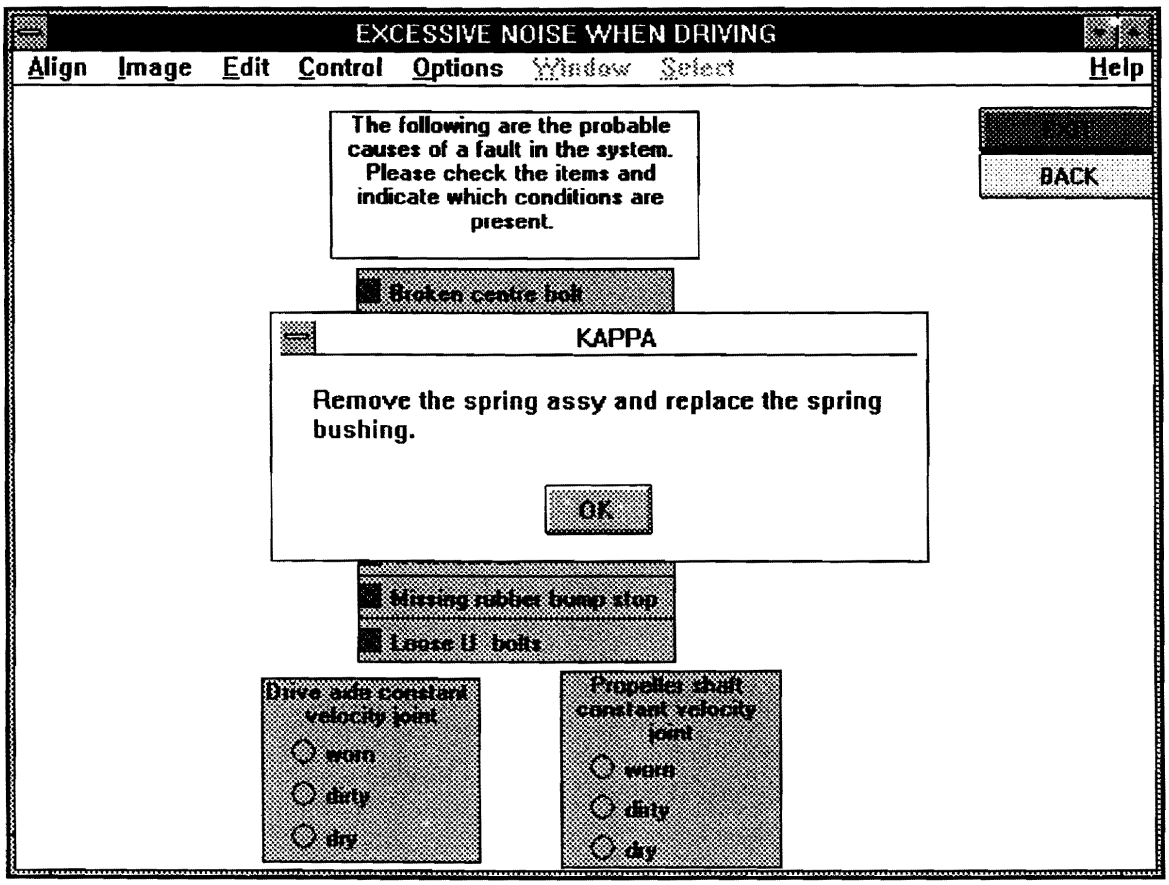

Figure 24. Sample consultation 2 - screen four

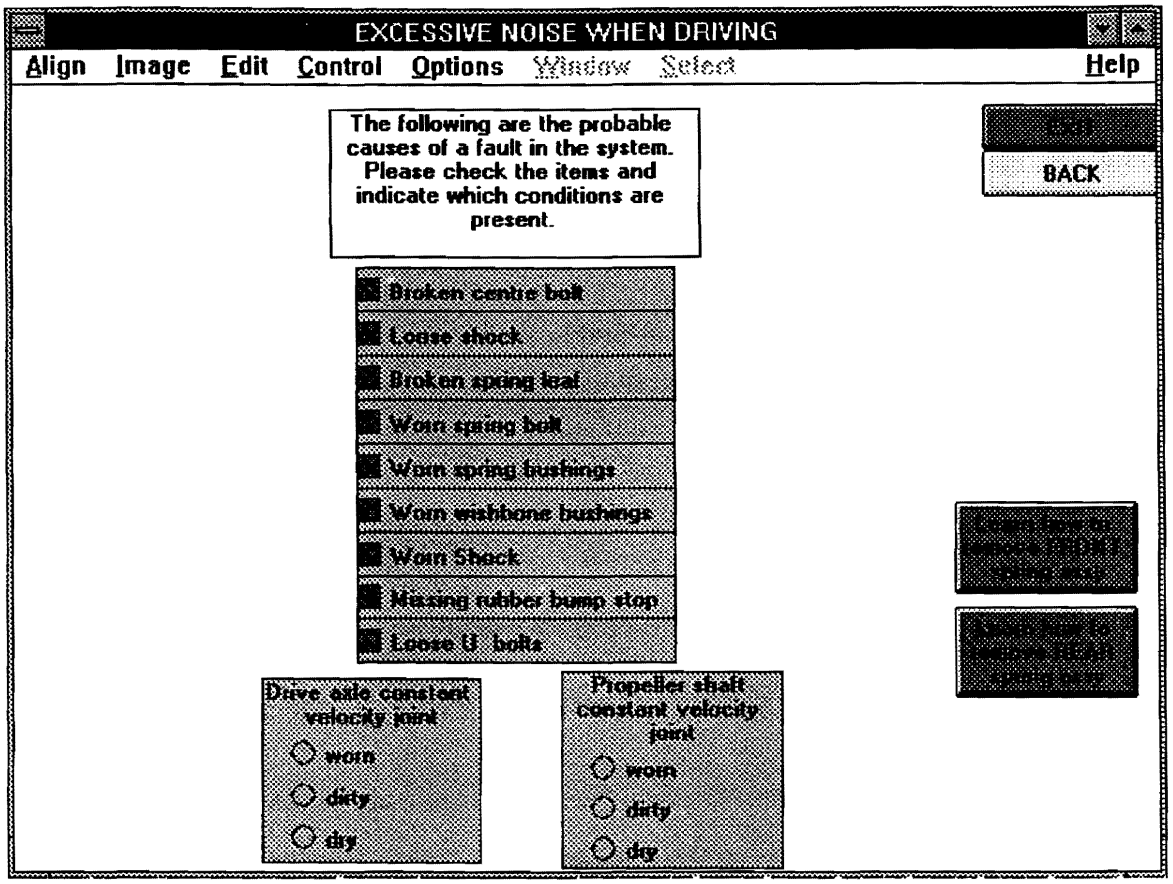

Figure 25. Sample consultation 2 - screen five

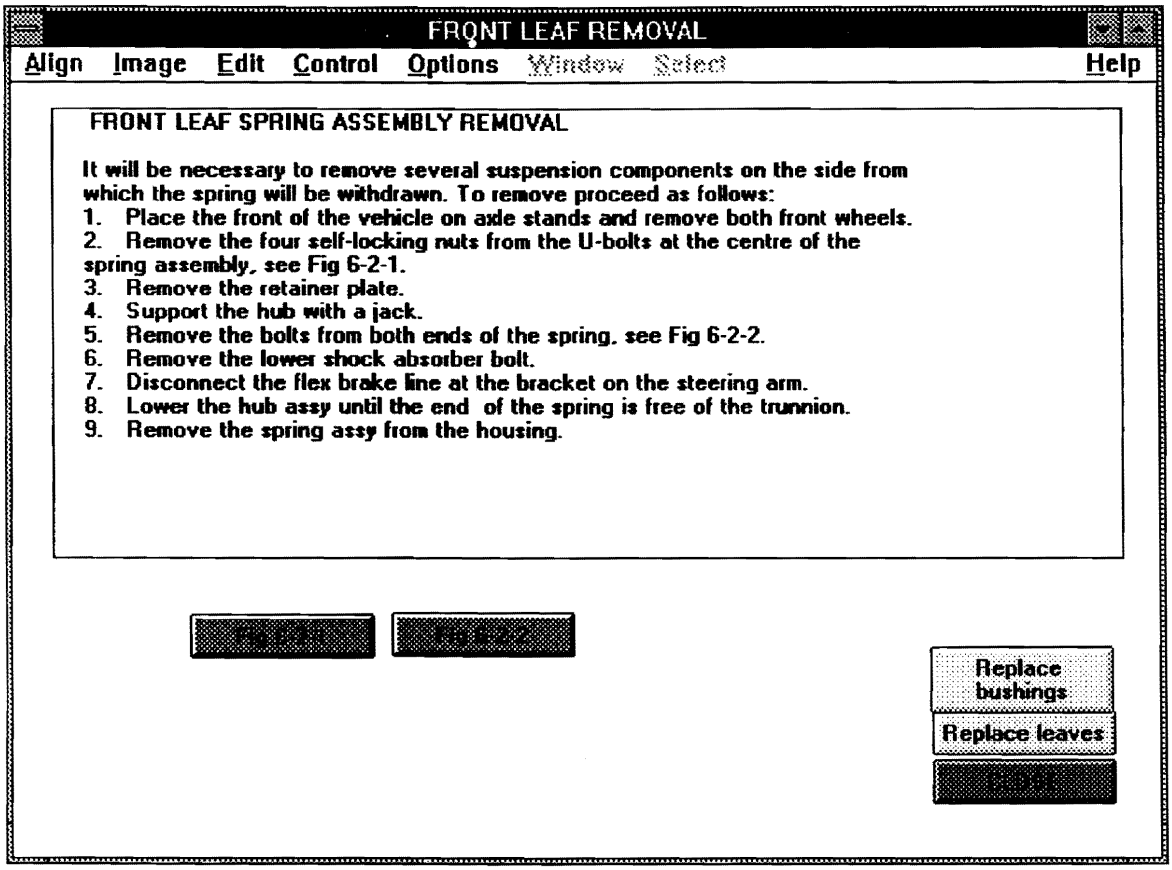

Figure 26. Sample consultation 2 - screen six

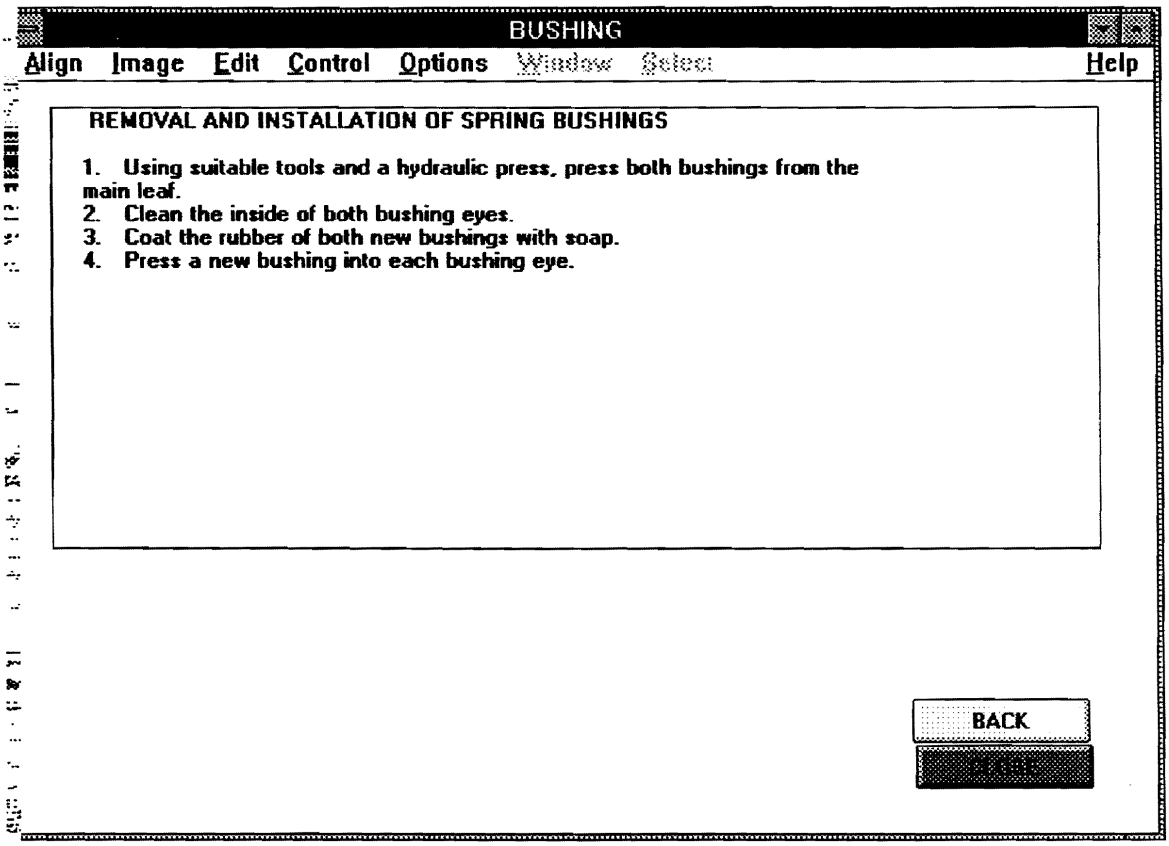

Figure 27. Sample consultation 2 - screen seven

Chapter 6: Conclusion

This report summarized the development of the Iltis Diagnostic System, a software tool to be used for the diagnosis of maintenance faults. A systems engineering approach to the design was adopted and a Knowledge-Based Expert System Tool was adapted for use in the development phase.

Originally the approach to this project was to design an expert system that could leverage information deemed to be of an expert nature and in short supply. Further study of the domain has shown that the source used to collect the knowledge did not impart any heuristic or inferential knowledge but only factual data. As a result the rules were written in a way that there is little room for an expert system to function among them.

IDS does, however, leverage the information available in the CFTO and absolve the supervisor of some of his repetitive duties with regards to OJT personnel. Kappa-PC has still proved to be a good tool to use for the development of IDS despite the fact that the programme itself is not an expert system.

Although alternative one was considered the most feasible due to the nature of this project. I consider IDS to be a prototype that demonstrates the capabilities that exist in this domain and am of the opinion that this type of system could be adopted by the Canadian Forces for future development and employment.

Keeping this in mind, a review of the alternatives discussed in Chapter 2 suggests

that alternative two be adopted, that is the CF contract the development of diagnostic software to a civilian firm who would also be responsible for repair of the programme.

Future Considerations

Future considerations for the development of this software should include:

- further refinement of the cost data in Tables 1 to 3 to ensure that an accurate assessment of the best possible alternative has been made
- primary consideration be given to production for new vehicle acquisition
- production of the software and any supporting material in both official languages (English and French)
- a format that continues to closely match that of the CFTOs
- the acquisition of a CF-wide license for the software rather than having to but individual copies
- inclusion in the contract of a software support line (1-800 number) and the facility to report faults in the programme
- close liaison between the software developers and the civilian contractor responsible for the production of maintenance manuals, perhaps including the provision of this software in the vehicle acquisition contract
- maintain the option to upgrade the programme to keep pace with increasing hardware and software technology at an additional cost

List of Works Consulted

Blanchard, Benjamin S, and Wolter J. Fabrycky, Systems Engineering And Analysis. Englewood Cliffs, NJ: Prentice Hall, 1990.

Canadian Forces Technical Order C-30-108-000/MP-001 First, Second and Third Line Maintenance Instructions Military Design Iltis Canadian Series. Ottawa, Canada: Department of National Defence, 1991.

Giberson, Stacey and Karen Ritchie, Prototype Iltis Diagnostic System. Blacksburg, Va, 1996

Hopgood, Adrian A, Knowledge-Based Systems for Engineers and Scientists. Boca Raton, Fl: CRC Press, 1993.

Hunt, V. Daniel, Artificial Intelligence & Expert Systems Sourcebook. New York, NY: Chapman and Hall, 1986.

Savory, Stuart, ed. Artificial Intelligence And Expert Systems. Chichester, UK: Ellis Horwood Limited, 1988.

Volkswagen GTI, Golf, Jetta Service Manual. Cambridge, Ma: Robert Bentley, 1992.

Appendix A: List of IDS Files

Kappa-PC (.kal)	WordPerfect (.wpd)				Bitmap (.bmp)	
clutch	3511ac	drivocvd	ppi1	start1	251	344
diode	3511bde	enginei1	ppi2	start3	2511	345
iltis	3512cd	enginei2	ppi3	test292	252	346
light	3512cdef	enginei3	ppi4	test293	262	621
primary	3514bcde	enginei4	ppi5	test295	263	622
starter	3521b	enginei5	ppo1	testpri	264	623
suspension	3521c	enginei6	ppo2	testsol	265	624
transaxle	bump	enginei7	ppo3	test1	271	625
	bush	enginei8	ppo4	test2	272	
	clutch	enginei9	ppo5	test3	273	
	continui	engini10	propcva	test4	274	
	daxlei	engineo1	propcvd	test5	2810	
	daxleo	engineo2	propcvi	test6	2811	
	diode1	engineo3	propcvo	test7	285	
	diode2	engineo4	rleafi	wishi	3115	
	drivicva	fault	rleafo	wisho	312	
	drivicvo	fleafi	safety		321	
	drivicvd	fleafo	shocki		322	
	drivicvi	gearshif	shocko		341	
	drivocva	intro	springi		342	
	drivocvo	powerpac	springo		343	