5-1-1994

The design of a knowledge base for a cooperative teleassistance system

Avare Stewart
Clark Atlanta University

ABSTRACT

COMPUTER SCIENCE

STEWART, AVARE B.S., RENSSELAER POLYTECHNIC INSTITUTE, 1989
M.S., CLARK ATLANTA UNIVERSITY, 1994

THE DESIGN OF A KNOWLEDGE BASE FOR A COOPERATIVE TELEASSISTANCE SYSTEM

Advisor: Dr. Erika Rogers

Thesis dated May, 1994

In order for knowledge base systems to behave in an intelligent way, domain knowledge must be built into them. The Teleopertive Visual Interactive Assistant (teleVIA) is a knowledge based system, designed to facilitate cooperation between a human and a remote robot. However, teleVIA must have a repository of facts, or domain knowledge, that reflects up-to-date information about the status of the teleassistance system.

Two questions with regard to this domain knowledge must be addressed. In particular, what information does the knowledge base consist of and how is it organized? In light of these concerns, this work proposes a design for a knowledge base using frames that will support cooperation between a remote robot and a human.

CLARK ATLANTA UNIVERSITY

THE DESIGN OF A KNOWLEDGE BASE FOR A COOPERATIVE TELEASSISTANCE SYSTEM

A THESIS SUBMITTED TO
THE FACULTY OF CLARK ATLANTA UNIVERSITY
IN CANDIDACY FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY

AVARE STEWART

ATLANTA, GEORGIA

MAY 1994

R= Vi P= 120

ACKNOWLEDGEMENTS

Dr. Erika Rogers, one whom I was fortunate enough to have the opportunity to work with, thank you for your patience and guidance in assisting me in completing a Masters degree. I would like to thank my committee members, Dr. Skrikanth and Dr. Warsi, for their advice and input. I extend a special thank you to Dr. Warsi for mentoring and guiding me during my entire academic stay at Clark Atlanta University; and not letting me forget how to smile. I would like to thank Elizabeth Nitz and Dr. Robin Murphy at the Colorado School of Mines for prompt and thorough repsonses that allowed me to get off the ground during the early stages of this work. Khalil Khalif, to whom I owe a million thanks for all his indispensable assistance which allowed me make progress at times when I was completely stumped. I thank Leslie Smart for his boost of confidence. I thank all the administrative staff in the Computer Science Department, Dr. Bota and the staff in Sponsored Programs for everything that they have done to help me through the Masters program. I thank Dr. Yvonne Freeman for her recommendations which led me to Clark Atlanta University. I thank Jayson Taylor for supporting me through this endeavor.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

Purpose Statement

In order for knowledge based systems to behave in an intelligent way, domain knowledge must be built into them. The purpose of this work is to design a knowledge base that supports cooperation between a remote robot and a human. The focus of building the knowledge base is placed on identifying the domain specific facts which must be incorporated into the intelligent system and determining how to represent them.

Motivation

In light of ongoing space explorations, scientists have been seeking to successfully implement and improve current telesystems paradigms. In fact:

> the technical and popular press alike have long envisioned the development of an intelligent robotic capability ... Such a robot will have the cognitive, perceptual, and motor skills to work cooperatively with humans, sharing the responsibilities for both decision making and task performance [Schenker, 1991].

Space is only one example of an environment for which

humans have no natural adaptation. With such inabilities, it is important that tools be developed that allow scientists to understand and interact with such hostile environments as space.

Semi-autonomous robots are ideal to use in remote, unexplored, hazardous or hostile environments. However, semi-autonomous robots have limited capabilities in responding to an unexpected change in their environment. Since unexpected changes may render the robots inoperable, researchers are developing teleassisted systems to get past the robot's limitations. Coiffet and Gravez suggest that a teleassistant would allow on-line communication between a human and a machine. Such collaboration would allow a human to help a remote agent recover from an unanticipated problem [Coiffet and Gravez, 1991]. To this end, one teleassistance system is being jointly developed by Rogers, at Clark Atlanta University, and Murphy, at the Colorado School of Mines. Their teleassistant is comprised of two systems, teleSFX (Sensor Fusion Effects) and teleVIA (Visual Interaction Assistant). TeleSFX, as part of the remote system, supports the robot's perception and motor behavior using state based intelligent sensor fusion. TeleVIA, supports the local system, is a knowledge-based system that supports the exchange of information between human perception and problem solving within a blackboard-style architecture.

In this work, one or more mobile robots, with limited autonomous capabilities, operate in a distant site away from a human operator. The robot may be engaged in some task in a remote, hazardous, or previously unexplored environment. Robots gather data from their environment using different types of sensors. The data from these sensors are combined in a process called sensor fusion. If a failure occurs during sensor fusion, the recovery system of the robot attempts to autonomously classify the failure, by 1) generating a set of hypotheses, and 2) testing each hypothesis. Based on confirmations of the hypotheses, the recovery mechanism selects a plan that will allow the robot to continue its task. Since successful recovery requires extensive knowledge about the domain, the semi-autonomous robot may be unsuccessful in recovering from failures. When this happens, the remote signals for help from the human operator.

At the local site, the human, who may have information the robot does not have, uses teleVIA to help the robot recover from the failure. Certain images received from the robot at the time of failure are presented to the operator via the blackboard. The operator attempts to interpret the contents of the images and draw conclusions about why failure has occurred. TeleVIA focuses the user's attention by presenting relevant information and visual enhancements at specific stages during the human's problem solving

process. Once the operator reaches a conclusion, a possible recovery strategy is suggested to the remote system.

Problem Statement

Besides the knowledge supplied by the robot and the human, teleVIA has its own domain knowledge that it uses in the cooperative problem solving process. This knowledge is represented by facts, specific to the robot and its environment, that are stored in teleVIA's knowledge base. Without domain knowledge, teleVIA would have no intelligent capabilities. Given the significance of domain knowledge, the following questions must be addressed:

1) What domain specific facts about the robot and its environment must be incorporated in teleVIA's knowledge base?

2) How can these facts be represented in teleVIA's knowledge base?

In light of these concerns, this work proposes a design for a knowledge base that will help a human assist a remote robot in recovering from a failure.

Approach

This research was approached by first doing background reading on the teleassistance system developed by Rogers and Murphy. Facts were collected based on the assumption that teleVIA would need to know the factors that determine the robot's motor behavior. Facts then thought to be important to the human were collected. Information gained from a graduate student working with Dr. Murphy at the Colorado School of Mines allowed: 1) unnecessary facts to be eliminated, 2) facts about the robot that were initially not considered, to be added, 3) details about the robot and the environment to be obtained, and finally, 4) concepts obtained from the data collected, to be identified.

Next, a model was designed that reflected the relationship among the concepts. To organize these concepts, possible representations were considered. The concepts were then mapped into the chosen representation. Finally, code was written to reflect the design. The C code was written using a Sun Sparc Station.

The remainder of this work is organized as follows: Chapter 2 provides background information about the cooperative teleassistance developed by Rogers and Murphy,

and addresses question number one of the problem statement. Chapter 3 discusses details about the knowledge base design, and addresses question number two of the problem statement. In Chapter 4, a description is given of the programs used to implement the knowledge base, sample output is provided, and an evaluation is made of the knowledge base. Finally, Chapter 5 provides a summary, comments on future work and conclusions.

CHAPTER 2

BACKGROUND

Overview

In this chapter, a brief overview of Rogers and Murphy's cooperative teleassistance system is given to lay the foundation for the concepts that are represented in teleVIA's knowledge base. An explanation is provided for each component of the teleassistant as it relates to the type of information that must be stored in the knowledge base.

Semi-autonomous robots are ideal in hazardous environments; however, semi-autonomous robots have limited capabilities in responding to unanticipated changes in their environment that may hinder their performance. The semi-autonomous system is limited because it is difficult for developers to predict all possible, unforeseen phenomena when building the robot's intelligence. Given this, researchers have considered teleassisted systems for circumventing the problem associated with semi-autonomy. Coiffet and Gravez refer to teleassistance as a particular configuration of computing resources that provides task-oriented assistance to a human operator that is interacting with some remote system [Coiffet and Gravez, 1991]. They

suggest that a machine configured as a teleassistant provides the operator with "punctual and powerful help." Considering these suggestions, a cooperative teleassistant is being developed by Rogers and Murphy [Rogers and Murphy, 1993]. Their teleassistant is comprised of two systems, teleSFX and teleVIA.

TeleSFX is a modification of Murphy's work on Sensor Fusion Effects - SFX [Murphy, 1992]. In teleSFX, the motor behavior of a robot is supported by Intelligent Sensor Fusion. Intelligent Sensor Fusion constitutes the robot's perception i.e., the robot's ability to "know" what is in its environment and "react" to this knowledge.

TeleVIA is an expansion of the work done by Rogers on the Visual Interactive Assistant (VIA) [Rogers, 1992]. TeleVIA is designed to "cooperatively assist human perception and problem solving in a diagnostic visual reasoning task" [Rogers and Murphy, 1993]. TeleVIA provides information to help the human focus on the relevant information, at the right time, during the human perceptual and problem solving process. Perception is the human's ability to see an image and make an internal representation of that image. Human problem solving refers to the human's ability to provide an explanation of the internal representation obtained from the perceptual process.

TeleSFX is used to support robotic perception and motor behavior, whereas, TeleVIA is used to enhance the human's

perception and problem solving abilities. By combining these two systems, the human's enhanced perceptual capabilities can augment the robot's autonomous perceptual system, thereby circumventing the problem associated with using semi-autonomous robots. A key aspect of their work includes features that facilitate cooperation between the remote robot and human.

Components of the Cooperative Teleassistant

In this domain, an autonomous, mobile robot operates in a remote environment. The environment may be a previously unexplored planet, or, the environment may contain chemical contaminants that are harmful to humans. However, since Rogers & Murphy's cooperative teleassistance is still under development, the robots are restricted to a laboratory. Figure 1 shows how the laboratory may be partitioned into several scenes; they are: drill press, vcr monitor, door, or the student desk scenes. Each scene contains at least one object that distinguishes it from the other scenes.

There can be one or more robots in the lab, each engaged in different tasks. Two robots are used in developing the teleassistant. The first robot is George, a Denning DRV mobile robot at Georgia Institute of Technology. The second is Clementine, a Denning MRV mobile robot, at the Colorado School of Mines.

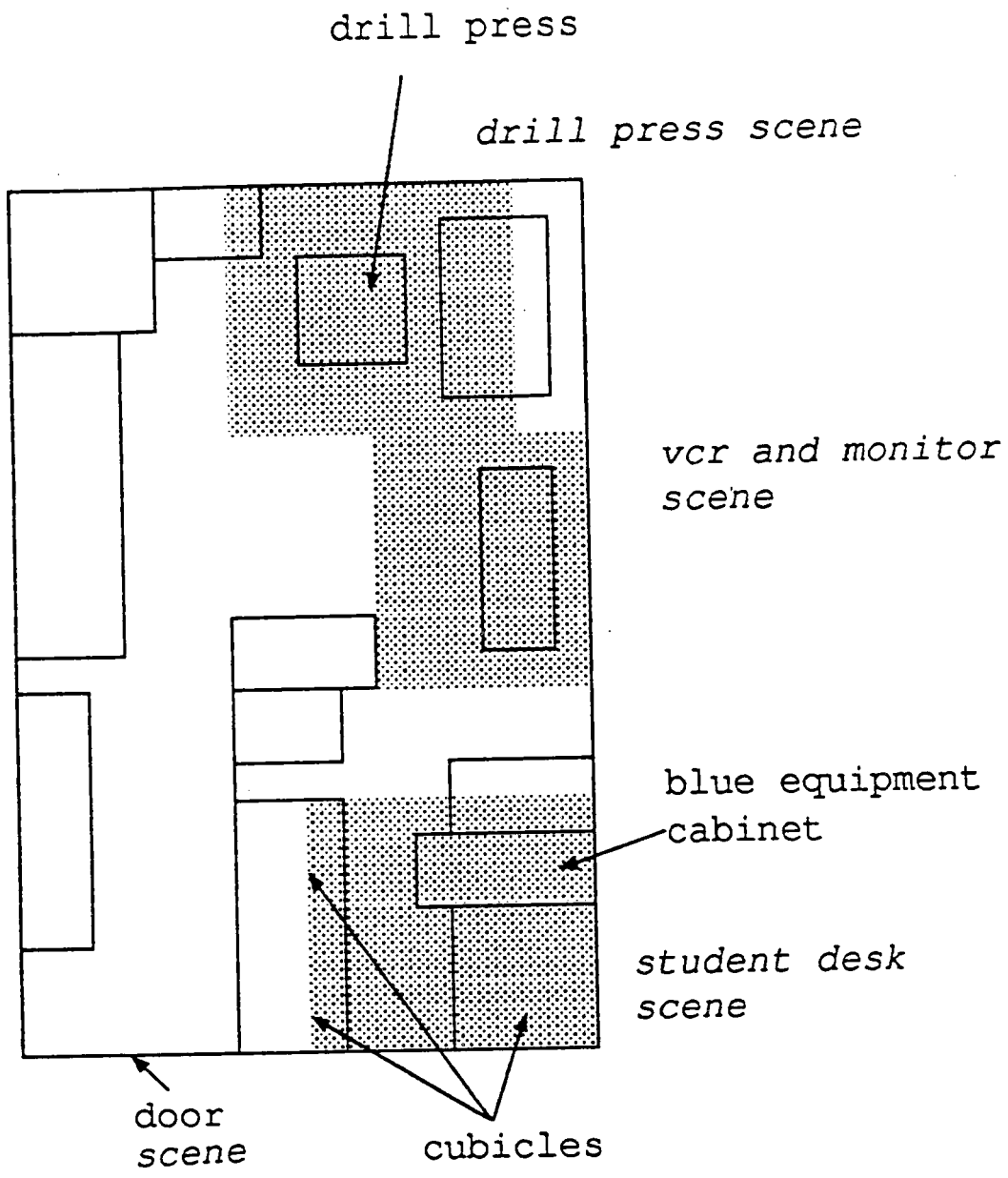

Figure 1: The Laboratory Environment

Although George is no longer being used, it has been included to emphasize the fact that this teleassistance system may support more than one robot.

The robot gathers data from its surroundings using different types of sensors. Collectively these sensors are called a sensor suite. A suite may consist of the following sensors: a Sony Hi8 video camcorder, a Pulnix black and white camera, an Inframetrics infra-red camera and a Polaroid 24 ring Ultrasonics transducer. Each robot makes observations, via its sensors, for different scenes.

The components of the cooperative teleassistant (written in bold text) are described below. Figure 2 shows the components of the cooperative teleassistant and how they interface with each other.

Autonomous Execution (defined as the robot's ability to autonomously gather and combine observations from multiple sensors [Murphy, 1992]), begins with a sensing plan. Part of this sensing plan identifies the suite of sensors (**sensor1**, **sensor2**,...) and particular aspects of the current scene that will be observed (i.e., color or heat).

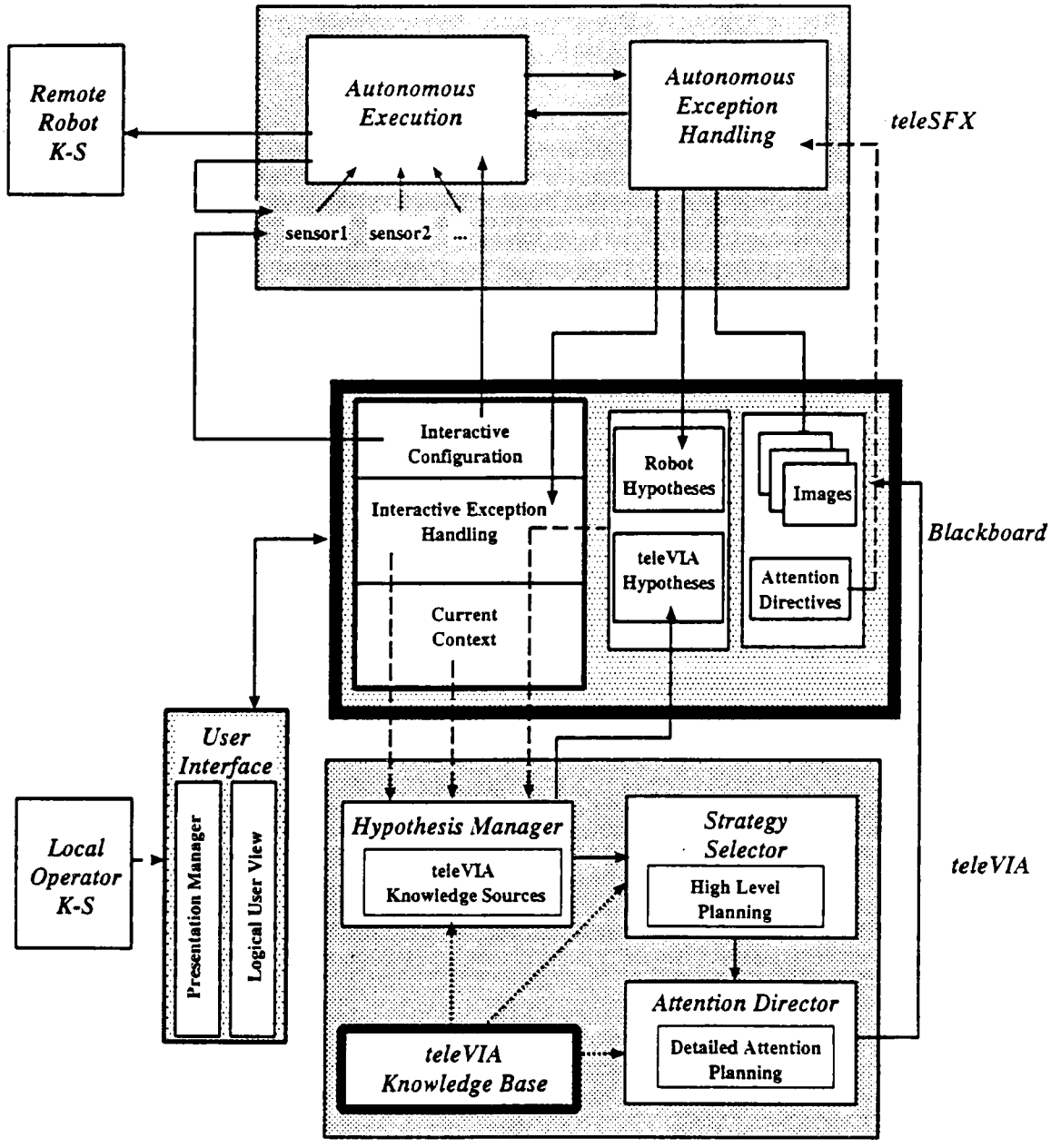

Figure 2: Cooperative Teleassistance System

The information gathered from each sensor is combined by the Autonomous Execution mechanism in a process called sensor fusion. As mentioned previously, fusion constitutes the robot's perception of what is in its environment. As part of the fusion process, evidence is generated and used to support the motor behavior of the **Remote Robot** in a given task.

If a failure occurs at a particular state of the fusion process, control passes from the Autonomous Execution module to **Autonomous Exception Handling**. Using a two-part strategy, Autonomous Exception Handling attempts to make a quick recovery so the robot may continue its task.

In part one of the recovery process, Autonomous Exception Handling identifies the type of failure. Robot hypotheses are generated and tested to determine: 1) if a sensor has malfunctioned, or 2) if a change in the environment has taken place. Based on confirmation of the hypothesis, the Exception Handling module will first try repairing the current sensing plan. A plan may be repaired by either removing a sensor from the suite or adding one to it. If the sensing plan cannot be successfully repaired then an attempt is made to replace the current sensing plan. The replacement is based on challenges to environmental preconditions that specify the operating conditions that must be met in order for the sensors to operate.

Autonomous Exception Handling may not always get the

robot going again. Two examples illustrate this point. First, the hypothesis generated by the Autonomous Exception Handling may be confirmed, but actual recovery might necessitate human intervention. For example, if the lights go out, the exception handler may successfully identify that an environmental change has taken place; however, human intervention would be required for the lights to be turned back on.

In the second example, the generated hypothesis may not be confirmed, and again, Interactive Exception Handling is necessary. Here, the exception handler would conclude that a sensor was responsible for the failure when in actuality the robot may be attempting to make observations of the wrong scene. Here, the sensors are operating properly but a failure has occurred in the fusion process and the robot is not able to continue its task.

If remote recovery is not successful, then part two of the recovery strategy signals the local operator for **Interactive Exception Handling**. At this time, pertinent information related to the perceptual status of the robot at the time of failure gets posted to the Blackboard.

The **teleVIA Blackboard** is a key component of the teleassistant because it facilitates communication between the robot and the human. The Blackboard is used to hold specific, task-related information obtained from the robot that teleVIA uses to help the human assist the robot. The

blackboard reflects the contents of the knowledge base before and after a robot experiences a failure.

Interactive Exception Handling receives the signal from the remote for the operator to assist in the recovery process. This panel will also contain information about the type of failure, the sensors presumed to be causing the failure, and evidence for each sensor's observations at the time of failure. If the remote tried to recover a second time, the operator would also have evidence for the second attempt. The contents of this panel reflect the information contained in the knowledge base after a failure has taken place.

The **Current Context** panel contains up-to-date information about the robot's task, the environmental conditions such as light and temperature, and the sensors that are active. This reflects information stored in the knowledge base before a failure has taken place.

Robot Hypotheses, which were generated when the remote was attempting to recover autonomously, get posted to the blackboard when assistance from the local operator is requested. The **TeleVIA Hypothesis** panel contains hypotheses generated by teleVIA during Interactive Exception Handling. A selected TeleVIA hypothesis may result in the local operator selecting sensors or an alternate recovery plan. Once an alternate is selected, the robot may be able to continue its task. Thus, **Interactive Configuration** has

taken place.

Images, obtained from the robot, may be displayed and enhanced. TeleVIA guides the user in a direction that may lead to a recovery by affording visual enhancements of displayed images.

TeleVIA attempts to draw upon the experience of the human to compensate for that which the robot may lack in recovering from an unexpected change. However, teleVIA's ability to do so is based upon information that must be stored in the knowledge base before and after a failure.

In this chapter, facts about cooperative teleassistance designed by Rogers and Murphy have been presented. The next chapter gives details about the design of the knowledge base. The design is determined by the facts specific to the domain presented in this chapter.

CHAPTER 3

DESIGN OF THE TELEVIA KNOWLEDGE BASE

Overview

This chapter explains the details of the teleVIA knowledge base design. The choice of frames as a representation schema is explained. Next, the concepts represented in this schema and their relationships are defined, followed by an explanation for each concept's attributes. Finally, examples are given for navigating through the knowledge base.

Representation Schema

Representation schema refers to the type of structure that will be used to store knowledge about the concepts. In the teleVIA knowledge base, frames are used as a representation schema for several reasons. First, the choice of frames is related to the type of problem solving encountered in exception handling. Waterman suggests that frame structures are good to use if the "content" of the data is the basis for problem solving [Waterman, 1986]. Much of the information stored in teleVIA's knowledge base represents facts about the robot and its surroundings. When

a failure occurs, the data obtained from the robot at the time of failure can be compared with a priori knowledge stored in the knowledge base. Differences between the two data sets may provide insight into the reasons why failure occurred. If the knowledge base contained information about particular objects in a scene, a local operator may know if the robot was attempting to make observations of the wrong scene. By comparing the object list for the correct scene with the objects in the image obtained from the robot, the local operator may conclude that the scenes are different if all the objects are different.

Second, frames allow storage of both static and dynamic information. Dynamic information is important because some data can only be obtained from the remote during Interactive Exception handling. As mentioned previously, when the local system is invoked, pertinent information related to the perceptual status of the robot at the time of failure is passed to the local system. This information is dynamic because it is obtained upon failure.

Static data is obtained by the teleassistant before run-time. Static data is important for successful exception handling because it represents facts that are independent of those that may be obtained because of Interactive Exception Handling.

Third, frames allow the concepts to be represented as a collection of attributes. Some attributes link concepts

together. Thus, representing concepts as a collection of attributes easily and naturally expresses the relationships that exist among the concepts in the robot's surroundings. Additionally, frames allow the knowledge base to be easily expanded as more information is obtained that will enhance cooperative teleassistance. Finally, the use of frames is consistent with the representation schema used in both teleSFX and teleVIA.

Concepts and Their Relationships

This section describes the concepts and the relationships among the concepts that are represented in the knowledge base. The facts specific to the discussion in Chapter 2 are used to determine the concepts, their roles and attributes.

Figure 3 shows the network representation of the concepts in the knowledge base. The arrows represent the relationship between the concepts. In particular, the direction of the arrows indicates *how* the concepts are related to each other and the labeled arcs show the attributes that relate them. Note that some attributes are used in pairs to describe a particular relationship between

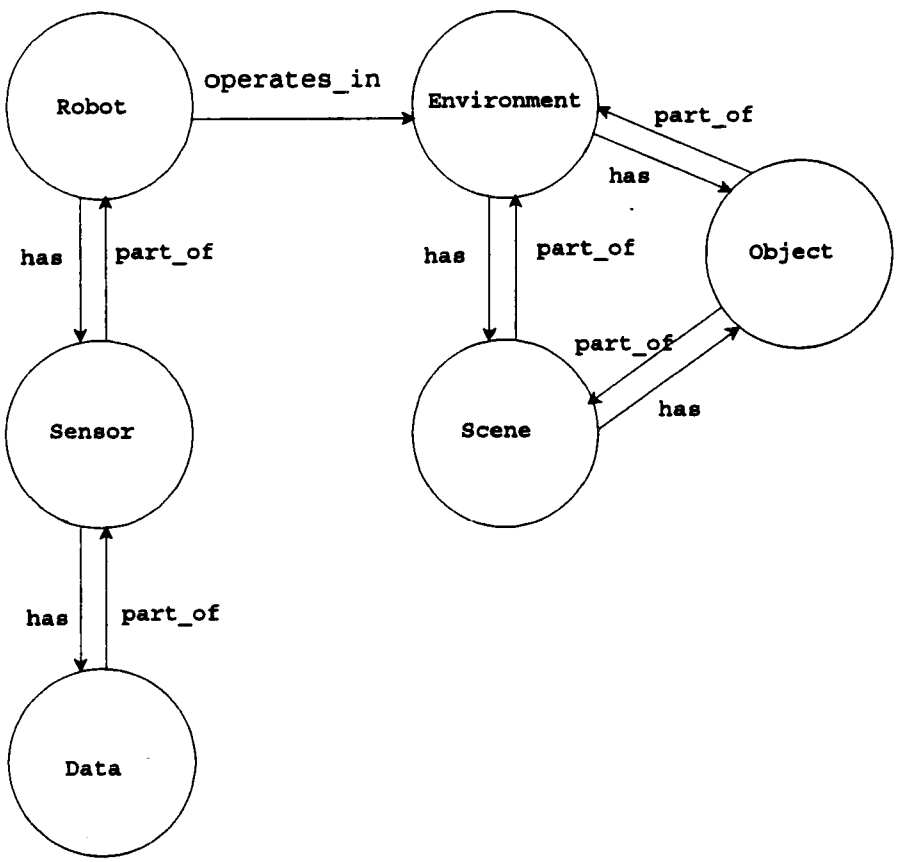

Figure 3: Network Representation of the Knowledge Base

two different concepts. The attribute pairs are represented by bi-directional links in the network. These attributes are called inverse attributes [Rich & Knight, 1991]. Relationships described by these attributes is one of inheritance. Note that the hierarchy among the concepts results from inheritance.

The concepts (written in bold text below) correspond to the nodes in Figure 3. Recall that the autonomous, mobile **robot** (or robots) operates in a remote, hazardous **environment**. For development purposes, however, the robots operate in the lab. The lab may contain several **scenes**, such as a student desk scene or a drill press scene (see Figure 1). The only means by which a robot can gather information about its environment is through its **sensors**. Each robot has multiple sensor types (or a suite of sensors). Data from these sensors are combined, in a process called sensor fusion, to produce a quantified value that supports the motor behavior of the robot in a given task. Each sensor provides **data** for its surroundings. This data is specific to the type of sensor used to obtain it.

Attributes

Each concept in Figure 3 has several attributes. In the discussion that follows, an explanation is provided for each attribute and why it was chosen. Symbols are used to show a specific role an attribute may have. The symbols

signify the following:

Rectangle: (□)	signifies that the attribute is a frame or set of frames.
Oval: (○)	signifies that attributes are grouped together because they are often used together.
Plus: (+)	signifies that the attribute relates the concept to another concept through membership.
Asterisk: (*)	signifies a dynamic attribute.

Attributes that have a plus sign next to them represent the arcs in Figure 3. Attributes that have an asterisk next to them indicate that their value is subject to change when the robot is operating autonomously or when a failure occurs.

Environment Frame

The Environment Frame attributes are shown in Figure 4. The attribute **NAME** identifies the environment the robot is in. The environment may have a number of scenes; thus, **HAS_SCENE** records all the scenes for the environment. In order for the robot to accomplish its

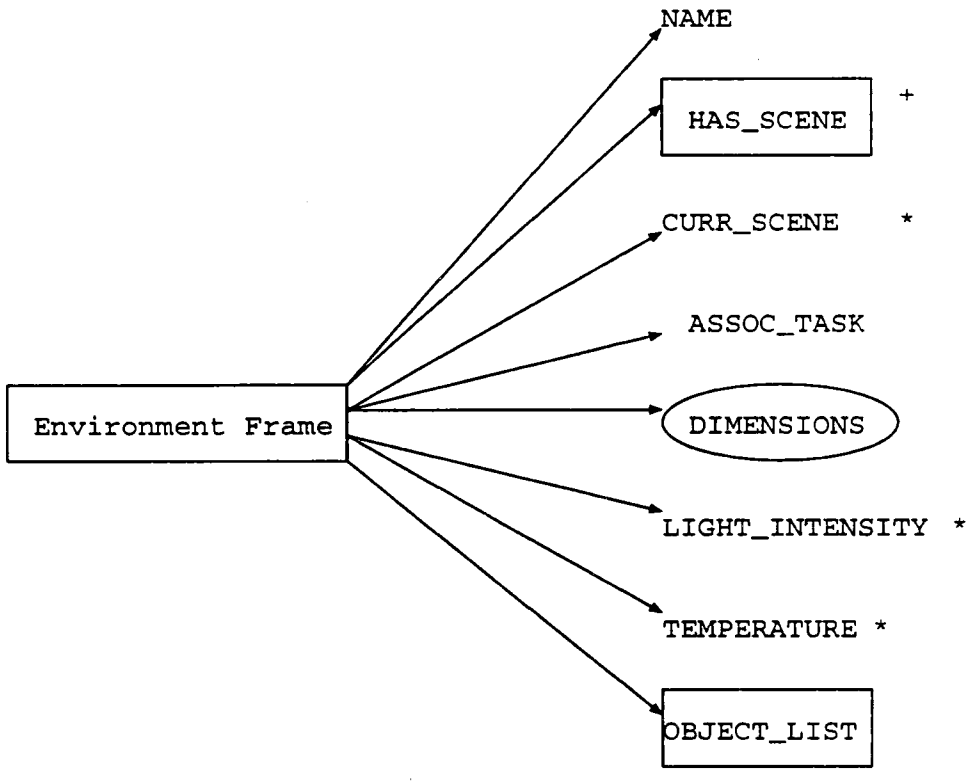

Figure 4: Environment Frame Attributes

task, it may have to gather data for a given scene; therefore, **CURR_SCENE** identifies the scene currently being observed. There may be several robots, each engaged in a different task, so the attribute **ASSOC_TASK**, lists all tasks associated with the environment.

Given that the operator will not share the same environment with the robot, it is essential to know as much as possible about the environment's physical properties. **DIMENSIONS**, measured in feet, consists of width, height and length, and are used to provide the operator with information about the extent of the robot's surroundings. **LIGHT_INTENSITY** captures information about the brightness of the light in an environment. The lights may either be 'on', 'off' or 'dim'. **TEMPERATURE** in an environment is measured in degrees fahrenheit. Recall that the recovery system's selection of a replacement plan is based on challenges to environmental preconditions. Preconditions specify the operating conditions that must be met in order for the sensors to operate. Insufficient lighting or extreme temperatures in the lab may prevent a sensor from working.

A number of objects may be in an environment; part of a robot's task may require it to manipulate or navigate around an object. Therefore, **OBJECT_LIST** keeps a list of all the objects known to be present in the environment. The Environment Frame is linked to the Scene Frame through the HAS_SCENE attribute.

Scene Frame

The attributes for the Scene Frame are shown in Figure 5. More than one scene may be part of an environment; therefore, **NAME** and **PART_OF** are used to hold the name of the scene and the environment, respectively. **PART_OF** links the scene frame with the object frame. A scene is comprised of a subset of all the objects from the environment frame's **OBJECT_LIST**. The attribute **HAS_OBJECT** lists all objects in the current scene. Each scene contains at least one object that distinguishes it from other scenes. Images that are requested from the remote are images of scenes in the environment. The HAS_OBJECT attribute may help the operator verify scene information by comparing the objects in the HAS_OBJECT list with the objects present in the image received from the robot.

Object Frame

There may be special features about an object that may be important to teleVIA only or, to both teleVIA and the robot. For example, the robot does not need to know the **NAME** of an object, see Figure 6. On the other hand,

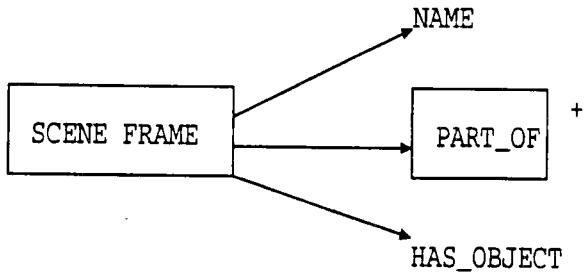

Figure 5: Scene Frame Attributes

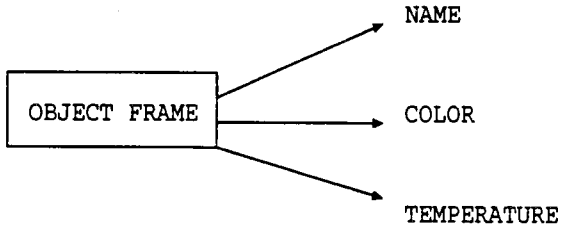

Figure 6: Object Frame Attributes

temperature and color may be important to both the robot and the operator. **TEMPERATURE** values can be 'above ambient', 'ambient', or 'below ambient'. Unlike the temperature slot in the Environment Frame, the value for this slot do not indicate an actual numeric temperature but a temperature relative to the environment. This is done because most of the objects in an environment will have a temperature close to the ambient temperature; therefore, the default value for this slot is 'ambient'. As an exception, some objects may have temperatures above or below the ambient temperature. Then, the robot would need to know that the object could have an exceptional thermal characteristic so that an infrared sensor could be used in the sensor suite. If teleVIA knew that an object had a temperature greater than the ambient temperature, it could use this information, (along with fact that an infra-red sensor was part of the sensor suite) to employ appropriate enhancements for images that contained this object.

The **COLOR** of an object may be important in the robot's perceptual process because the sensing plan may require a sensor to observe the object's color. On the other hand, the color may be important to the operator because the operator may easily identify an object in an image by its color.

Robot Frame

The Robot Frame attributes are shown in Figure 7. Every robot, uniquely identified by **NAME, OPERATES-IN** an environment. OPERATES_IN identifies the environment and establishes the link between the Robot Frame and the Environment Frame. Each robot is given a task. The task the robot is engaged in may be related to the failure; therefore, the **GIVEN_TASK** attribute is provided.

HAS_SENSOR consists of a list of sensor frames describing the types of sensors mounted on the robot. The types of sensors for each robot are known before run-time however, at any given time, different sensors may be active. **CURR_SENSORS_STATUS** keeps track of the current status of each sensor. The status of a sensor may be one of the following: 'not available', 'active', 'inactive' or 'suspect'. If a sensor is not available then it can never be part of a sensing plan because either the robot does not have that sensor mounted, or the sensor has malfunctioned and can no longer be used. Moreover, if a sensor is not available for the robot that experiences a failure, then this sensor cannot be selected in the recovery plan during Interactive Configuration. An active status means that the sensor is part of the current sensing plan. Inactive means that the robot has that particular sensor but it is not part

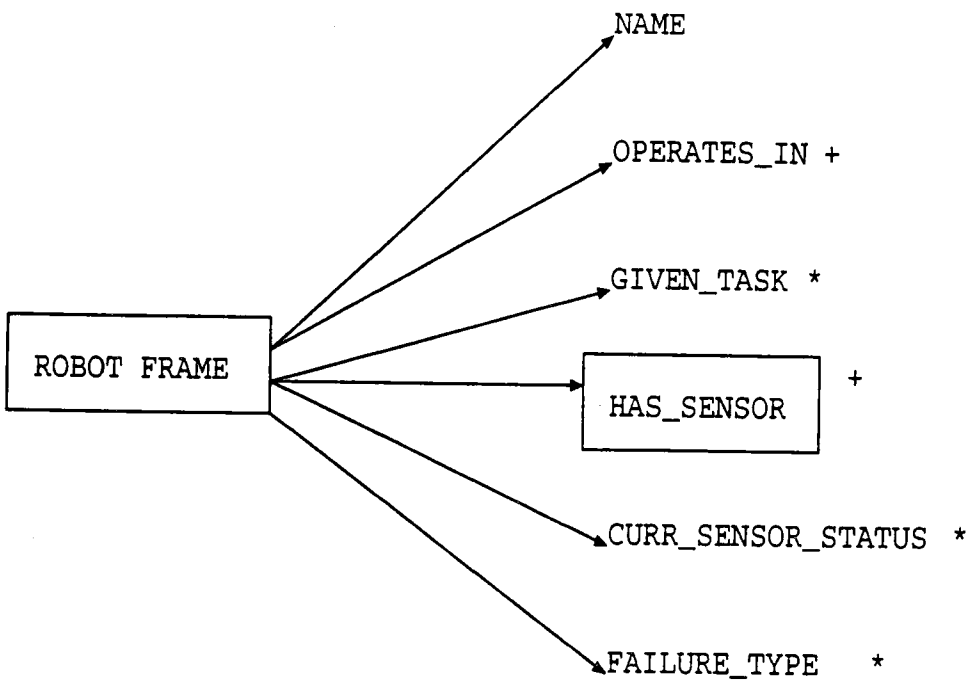

Figure 7: Robot Frame Attributes

of the current sensing plan. Suspect means that a failure has taken place and the sensor is considered as possibly causing the failure. TeleVIA needs to know which sensors are presumed to be causing the failure. Because it may attempt to provide information to the operator based on the status of a sensor.

A robot may experience four types of failures: 'high conflict', 'missing evidence', 'below minimum' and 'highly uncertain'. This information is stored in the **FAILURE_TYPE** attribute, and gets posted to the Interactive Exception Handling panel of the Blackboard when the local system is signalled.

Sensor Frame

Since each robot has different types of sensors, **NAME** and **USAGE** (in Figure 8) identify each sensor and why it is used, respectively. The usage for each sensor is as follows: black and white sensor may be used to detect visible light, the color sensor may be used to detect color in visible light, the infra-red sensor may be used to detect heat, the ultrasonics sensor may be used to detect range, and the ultraviolet sensor may be used to measure environmental changes.

DATA_TYPE is used to specify the type of data provided by a sensor because data is specific to a sensor. For this work, data can be either image or numeric. The black and white, color, infra-red, and ultraviolet sensors provide image data, whereas the ultrasonics sensor provides numeric data. Both the USAGE and DATA_TYPE attributes may help teleVIA select the appropriate enhancements. **PART_OF** identifies which robot the sensor belongs to. **FOV** stores information about the maximum horizontal and vertical field of view obtainable from the sensor.

EVIDENCE consists of PREV_EVIDENCE, CURR_EVIDENCE, and SECOND_EVIDENCE. PREV_EVIDENCE contains observation

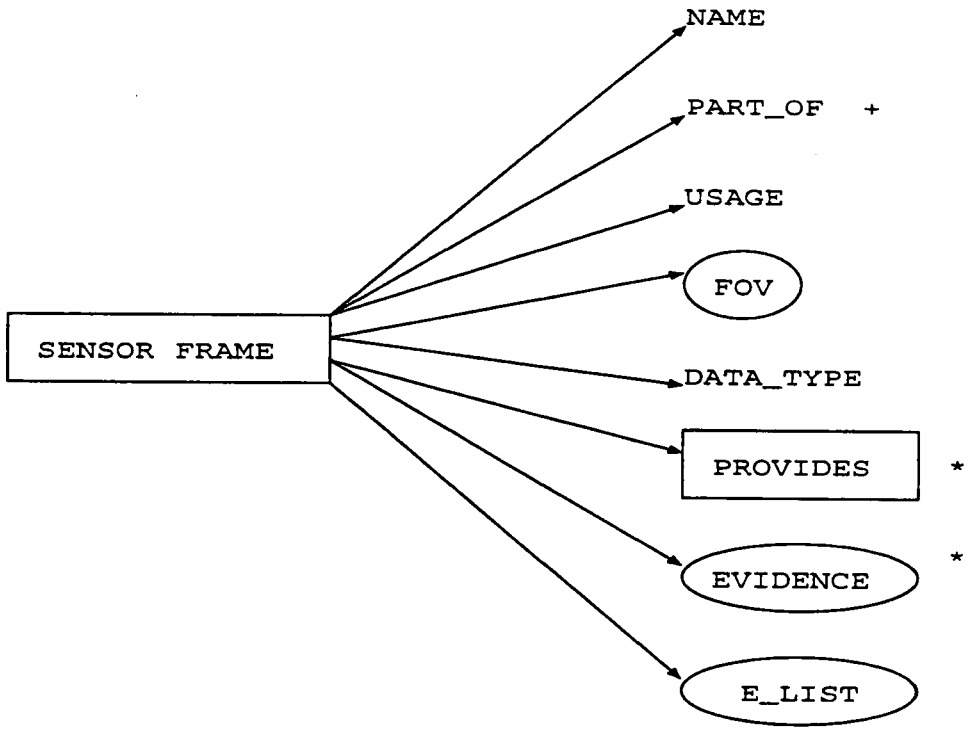

Figure 8: Sensor Frame Attributes

evidence obtained from the robot prior to a failure. CURR_EVIDENCE contains the evidence that was generated by Autonomous Exception Handling when a failure occurred and a first attempt at recover was made. SECOND_EVIDENCE contains values that would be generated if the robot attempted to recovery from the failure a second time. Evidence gets posted to the Interactive Exception Handling panel of the blackboard when the local system is invoked.

There are three types of evidence obtained from a sensor. The types are: 'support', 'against' or 'don't know'. Each type may have values between one and zero. A '0' represents a weak belief and '1' represents a strong belief that a particular set of features set forth by the sensing plan was observed by the robot's sensor.

E_LIST is a list of enhancements available for the data that is obtained from each sensor. Enhancements are important in directing the operator's attention to aspects of an image that may lead the operator to some conclusion about why failure occurred. Some examples of enhancements available are: false color, for the infra-red sensor, and occupancy grid and polar plot, both for the ultrasonics sensor. PROVIDES is a link to the Data Frame.

Data Frame

Figure 9 shows the data frame attributes. **PROVIDED_BY** identifies the sensor that provided the data. **SCENE** determines what scene the data is for. The current scene may change during Autonomous Execution. This attribute ensures the correct data is associated with the current scene. Also, during Interactive Exception Handling, data for a particular scene may be displayed. This attribute ensures that the correct scene is chosen.

IMAGE SIZE may be important. The transmission time may present a bottleneck if the images are very large. The size of an image is determined by horizontal size, vertical size, and depth, in bits. Based on the size of the image, only a relevant portion of the image may get transmitted to the local system. Both **IMAGE** and **NUMERIC DATA** attributes are files that store data obtained from the robot at the time of failure and prior to failure.

Navigation

This section shows examples of how navigation among the frames is accomplished through each level of the knowledge base hierarchy.

Navigating through the Environment, Scene & Object Frames

In Figure 10, an instance of an environment is represented by the lab. In this example, the environment

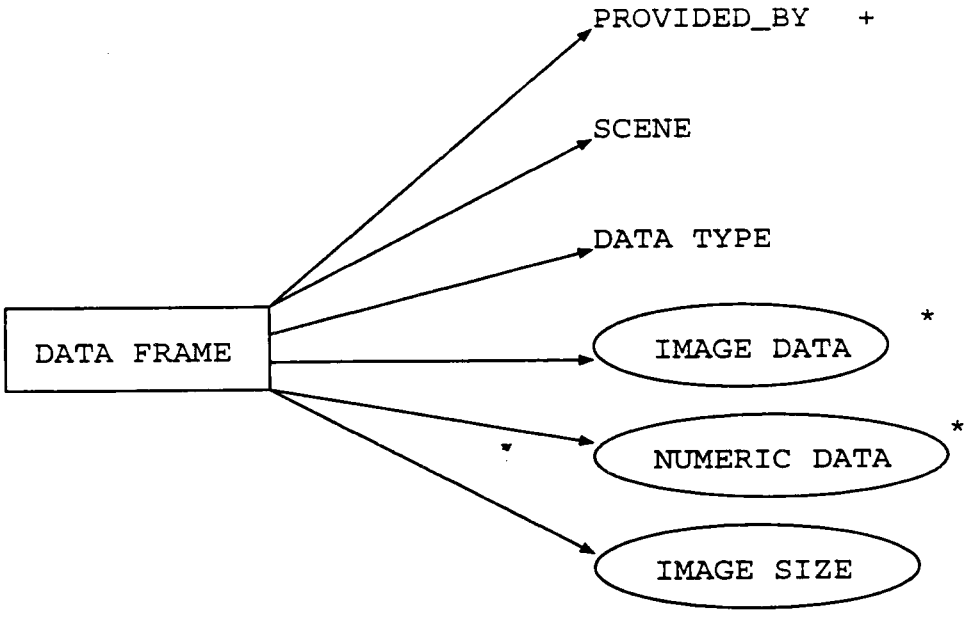

Figure 9: Data Frame Attributes

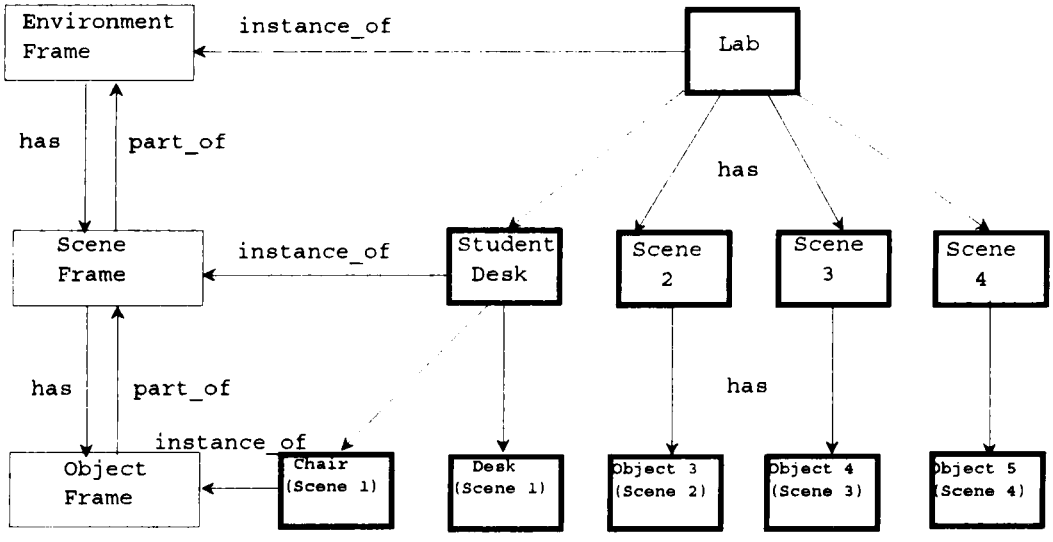

Figure 10: Environment, Scene and Object Frame Navigation

has four different scenes, one of which is the student desk scene. Each scene has a list of object frames. Two objects, a chair and a desk, are shown and both objects are part of the student desk scene (scene 1). Note there is at least one unique object for every scene.

Object Inheritance

Figure 11 shows how objects from the Environment Frames' object list are inherited by a scene. Objects one through four are located in the Lab but only the chair and desk are located in the student desk scene. Note that more than one object may belong to the same scene.

Navigating through Robot, Sensor & Data Frames

In Figure 12, the robot George is an instance of a robot frame, and it has four instances of the sensor frame. Three sensors are shown in the figure. They are: black and white (BW), Infrared (IR), and color sensors. Each sensor has a data frame and provides data that is unique to that sensor, hence there are four instances of the data frame, one for each sensor.

Navigating through the Robot, Environment & Scene Frames

In Figure 13, George is an instance of the Robot Frame. George operates in the Lab, which is an instance of

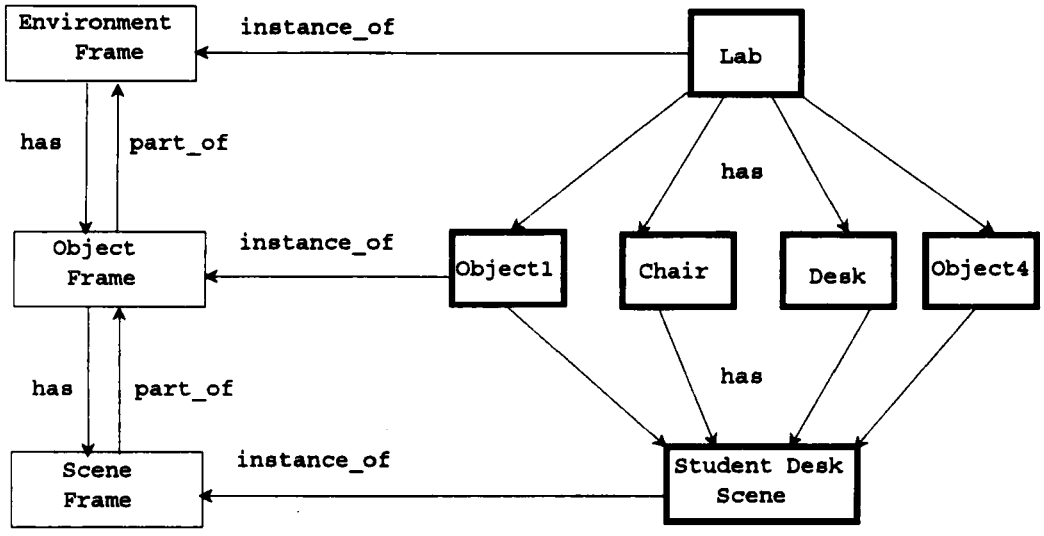

Figure 11: Object Inherited by a Scene

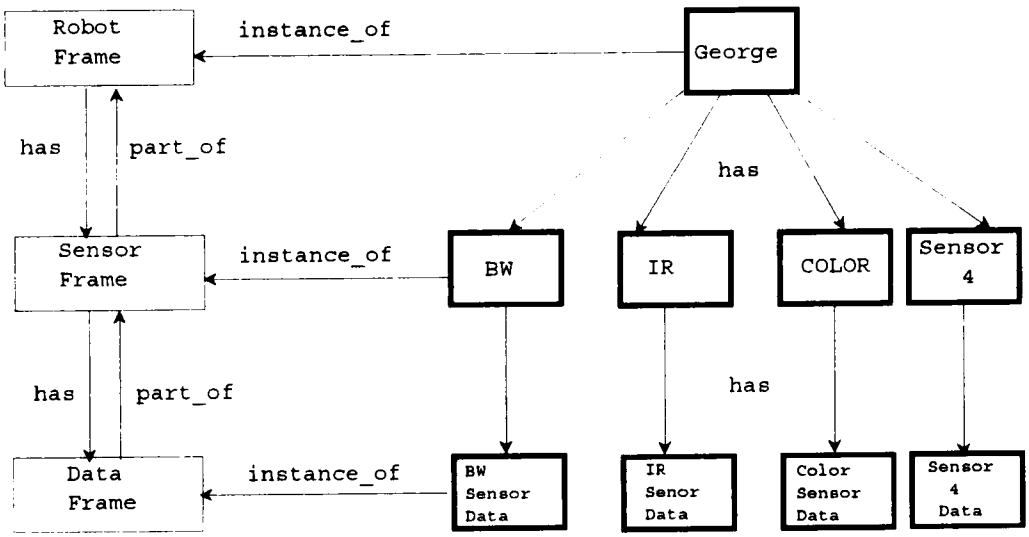

Figure 12: Robot, Sensor and Data Frame Navigation

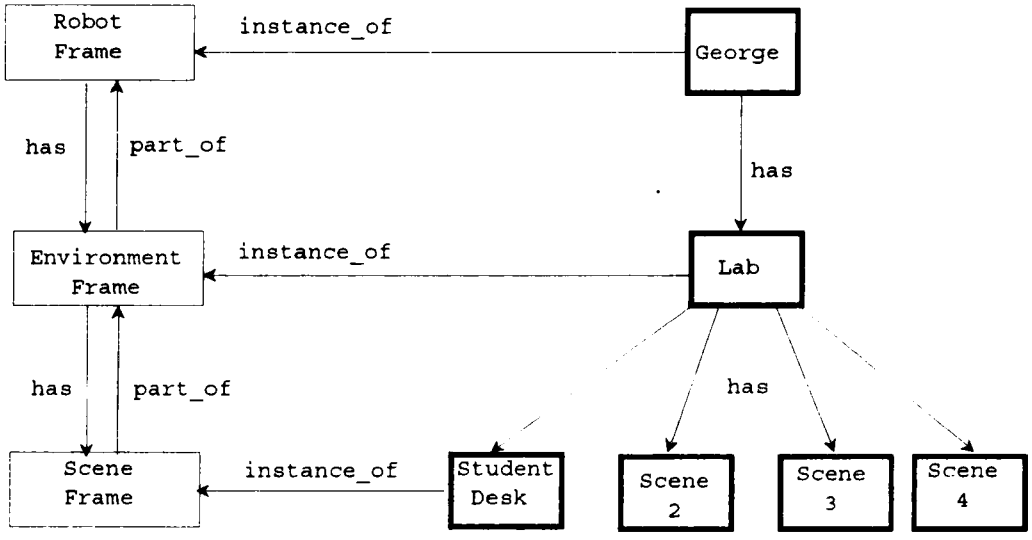

Figure 13: Robot, Environment and Scene Frame Navigation

the Environment Frame. The lab has a scene frame and there may be several scenes in the lab; therefore, scenes one through four are instantiated in the lab, one scene being the student desk scene.

CHAPTER 4

IMPLEMENTATION

Overview

In this section, the programs used to implement the knowledge base are described. Examples of the type of data that may fill each frame's slots are shown and sample output for the knowledge base routines is given. Finally, an evaluation of the knowledge base is made.

Knowledge Base Routines

The knowledge base was implemented using 10 routines, totaling about fifteen hundred lines of C code. These routines are described as follows: FRAMES.H contains the structures that define the knowledge base (see Appendix A). KBASE.H contains the facts about the robot and the environment (see Appendix B). INIT.C (see Appensix C) initializes all the frames in the knowledge base. DATA.C (see Appendix D) instantiates the frames. There are two environments - a lab and a warehouse. The robot frame was instantiated with two robots: George and Clementine. George has five sensors: black and white, color, infra-red, ultrasonics, and an ultraviolet. Clementine has three

sensors: black and white, color and ultrasonics. UPDATE.C (see Appendix E) updates the values of the dynamic attributes when a simulated failure occurs. PRINT.C (see Appendix F) outputs all attribute values. STRING.C (see Appendix G) provides an interface so that information stored in the knowledge base is presented to the user as text strings. MISC.C (see Appendix H) contains miscellaneous routines. GLOBAL.H (see Appendix I) contains global variables, and TEST.C (see Appendix J) executes the above routines.

Frame Instantiations

For the purposes of instantiating the frames with static data, it is assumed that the relevant information is available, and can be hard coded into the knowledge base. On the other hand, when a failure occurs, the dynamic values must be updated. A question arises about how this data is integrated into the knowledge base. In particular, does the robot dump the failure data to a file or does the robot put it directly onto the blackboard?

If it is assumed that the failure data is dumped into a file than teleVIA must do some preprocessing in order for the blackboard contents to reflect the status of the robot. Given this, the communication burden is on the local system.

If the remote sends the failure data directly to the blackboard, then the remote must be reconfigured so that it

can communicate directly with the blackboard's data structures. Now, the communication burden is on the remote.

These issues have not been resolved; however, for the purposes of this work, it has been assumed that the robot places its data in a file. As a result, the blackboard structures can access the failure data after it has been stored in the knowledge base.

For the instantiated frames in this example, a robot, George, is operating in a lab and is acting as a security guard to determine if the student desk scene has changed. A failure has occurred because George is attempting to gather data while facing the wrong scene (the drill press scene). Autonomous Exception Handling was not able to get the robot going again. As a result, the sensors presumed to be causing the failure (black and white and infra-red) have been stored in the knowledge base, along with the type of failure and evidence for the black and white, infra-red and ultrasonic sensors. Given this scenario, the remainder of this section shows the type of data that would fill each slot of the knowledge base frames. The slot names in bold text with an asterisk next to them correspond to the dynamic attribute values mentioned in Chapter 3.

Environment Frame

Figure 14 shows that George operates in a lab with four scenes, the student desk being the current scene. The

dimensions of the lab (in feet) are provided in WIDTH, HEIGHT and LENGTH slots. The actual dimensions and temperature for the lab were not obtained so the values shown here are merely examples of the type of data that

Environment Frame	
NAME	lab
HAS_SCENE	(vcr/monitor-scene, drill-press-scene, student-desk-scene, door-scene)
* CURR_SCENE	student-desk
ASSOC_TASK	security-guard
WIDTH	12 feet
HEIGHT	10 feet
LENGTH	12 feet
* LIGHT_INTENSITY	on
* TEMPERATURE	68.5 farenheit
OBJECT_LIST	(vcr, monitor,cabinet, drill press,desk, chair,door, fire extinguisher)

Figure 14: Instantiated Environment Frame

would fill those slots. The default value for LIGHT_INTENSITY is 'on'. A list of all objects present in the environment is shown. CURR_SCENE is an index into the HAS_SCENE list.

Scene & Object Frames

In Figure 15, the student desk scene contains three objects. These objects are inherited from the OBJECT_LIST slot in Figure 14. An example of the cabinet object frame is shown in Figure 16. The temperature of the object has the default value 'ambient'.

Robot Frame

In the instantiated Robot Frame of Figure 17, George has five sensors. For this sensing plan, the sensors have the following status: the ultrasonics sensor is 'active'; the ultraviolet and color sensors are 'inactive'; the infrared and black and white sensors are 'suspect' sensors in a 'high conflict' failure. The position of the attribute values in the list for HAS_SENSOR and CURR_SENSOR_STATUS correspond one-to-one.

Sensor Frame

In Figure 18, the PROVIDES attribute is a list of data frames for all the sensors available, however; actual data is stored for only currently active and suspect sensors.

Scene Frame	
NAME	student-desk-scene
PART_OF	lab
* CURR_OBJECT	(desk, chair, cabinet)

Figure 15: Instantiated Scene Frame

Object Frame	
NAME	cabinet
COLOR	blue
TEMPERATURE	Ambient

Figure 16: Instantiated Object Frame

ROBOT FRAME	
NAME	George
OPERATES_IN	lab
* **GIVEN_TASK**	security-guard
HAS_SENSOR	(BW, Color, Infrared Ultrasonics, Ultraviolet)
* **CURR_SENSOR_STATUS**	(suspect, inactive, suspect,active, inactive)
* **FAILURE_TYPE**	high-conflict

Figure 17: Instantiated Robot Frame

Sensor Frame	
NAME	BW
PART_OF	George
USAGE	detect-visible-light
HORZ_FOV	23.5 degrees
VERT_FOV	25.0 degrees
DATA_TYPE	image
* PROVIDES	< list of data frames for active and suspect sensors >
* CURR_EVIDENCE	(0.00, 1.00, 0.00)
* SECOND_EVIDENCE	(0.00, 0.00, 0.00)
* PREV_EVIDENCE	(0.00, 0.00, 0.00)
E_LIST	< no enhancements availible >

Figure 18: Instantiated Sensor Frame

NAME is an index into the PROVIDES list and E_LIST, the list of enhancements. The evidence is shown and previous evidence values are zero because it is assumed that the failure has taken place before a history of evidence could be established.

Data Frame

Since there are three sensors involved in the sensing plan (denoted by the active and suspect values in Figure 19), the knowledge base will contain three instantiated data frames. Figure 19 shows example data for the black white sensor. An image of the current scene at the time of failure is contained in a file pointed to by CURR_IMAGE. PREV_IMAGE would contain an image received from the robot just before failure occurred, however it is assumed that no history was established before failure occurred. PREV_NUMERIC and CURR_NUMERIC files would contain data from the ultrasonics sensor.

Data Frame	
PROVIDED_BY	BW
SCENE	student_desk
DATA_TYPE	image
*CURR_IMAGE	<pointer to a file>
*PREV_IMAGE	<pointer to a file>
*CURR_NUMERIC	no-data
*PREV_NUMERIC	no-data
HORZ_SIZE	640 bits
VERT_SIZE	480 bits
DEPTH	8 bits

Figure 19: Instantiated Data Frame

Sample Output

In this section sample output for each of the instantiated frames in the previous section is provided. Figure 20 shows that Scenario 1 was used to simulate a failure. In Scenario 1, the robot George is making observations of the wrong scene.

Figure 21 shows the menu options to print the contents of each frame for two robots, Clementine and George. The output below was obtained by selecting options one, two, and six through nine. The code for these menus is in Appendix F.

```
*****    Choose Scenario    *****
1 - Wrong Scene
2 - Lens Covered - Second Recovery Attempt
3 - Lights Out
4 - Fluctuating Readings
5 - IR Enhancement
6 - None (Quit)

 Select a number corresponding to the failure you
want to simulate: => 1
```

Figure 20: Scenario Menu

```
**********************************************
****     Menu - Print Knowledge Base Frames ***

*****    Choose a Frame to Print     *****

 1 - Environment
 2 - Scene

*** Clementine ***
 3 - Robot
 4 - Sensor
 5 - Data

*** George ***
 6 - Robot
 7 - Sensor
 8 - Data

 9 - None (Quit)

 Select a number corresponding to the frame you want to
print: =>
```

Figure 21: Knowledge Base Menu Options

Environment Frame

Output for the environment frame contents is shown in Figure 22. The list of all scenes and tasks for the environment are shown. Figure 23 shows output for the scene frame and all the objects within the scene.

```
************************************************
*****   Output for Environment Frame Contents *****

Name:              Lab
Width:             12.000000 feet
Height:            10.000000 feet
Length:            12.000000 feet
Light Intensity:   On
Temperature:       68.500000 fahrenheit
Current Scene:     Student-Desk-Scene


*** Scenes for Environment ***
Drill Press Scene
VCR/Monitor Scene
Door Scene
Student Desk Scene

****   Tasks for  Environment ***
Security Guard
```

Figure 22: Output for the Environment Frame

Figure 23 shows output for the Scene Frame. All the objects present in this scene are shown, along with the characteristics for each object. Output for the object frame is lengthy because it lists every object in the environment and its characteristics; therefore, it has been omitted here.

```
*********************************************
*****  Output for Scene Frame Contents *****

Scene              Student-Desk-Scene
Environment:       Lab

        ***  Object in this Scene  ***

        Object:          Cabinet
        Color:           Blue
        Temperature:     Ambient


        Object:          Desk
        Color:           Blue
        Temperature:     Ambient


        Object:          Chair
        Color:           Yellow
        Temperature:     Ambient
```

Figure 23: Output for the Scene Frame

Robot Frame

Figure 24 shows the Robot Frame contents for the George. Only those sensors which are either active or suspect are printed.

```
************************************************
*****    Output for Robot Frame Contents    *****


Robot Name:              George
Environment Name:        Lab
Robot Task:              Security Guard
Failure type:            High Conflict

 ***  Current Sensor List Status: ***

Sensor: Black & White
Status: Suspect

Sensor: Infra-red
Status: Suspect

Sensor: Ultrasonics
Status: Active
```

Figure 24: Output for the Robot Frame

Sensor Frame

In Figures 25 and 26, sample output for the Sensor Frame is shown. In Figure 26, only output for the black and white sensor is shown. Second evidence has been omitted since it is assumed that George did not try to recover a second time before signalling for help.

```
************************************
*****   Sensor Frame Contents  *****

Sensor:          Black & White
Robot:           George
Usage:           Detect Visible Light
Horz_Fov:        23.500000 degrees
Vert_Fov:        25.000000 degrees
Data Type:       Image


Sensor:          Infrared
Robot:           George
Usage:           Detect Visible Light
Horz_Fov:        46.400000 degrees
Vert_Fov:        40.000000 degrees
Data Type:       Image


Sensor:          Ultrasonics
Robot:           George
Usage:           Detect Range
Horz_Fov:        0.000000 degrees
Vert_Fov:        0.000000 degrees
Data Type:       Numeric
```

Figure 25: Output for the Sensor Frame

```
**********************************************
*****   Output for Sensor Frame Contents  *****
                    (cont.)

Evidence for Active and Suspect Sensors

( Support, Against and Don't Know
  is listed from top to bottom )


 Black & White

Current Belief:   0.000000
                  0.100000
                  0.000000

Previous Belief:  0.000000
                  0.000000
                  0.000000


*** List of Available Enhancement for Sensors  ***

 Black & White:          none
 Color:                  none
 Infrared:               false color
 Ultrasonics:            occupancy grid
 Ultrasonics:            polar plot
```

Figure 26: Output for the Sensor Frame, continued

Data Frame

Output for the data frame is shown in Figure 27. The current and previous data files have been omitted in the output. The default file names for the image and numeric data are: 'prev_raw_image', 'curr_raw_image', 'prev_raw_numeric', and 'curr_raw_numeric' (see Appendix A).

Evaluation

For the purposes of this work, the robot operated in a lab. If the robot was to be placed in a more hostile environment i.e., on a remote planet, other environmental factors should be considered. It was assumed that the only known environmental factors that impact the performance of sensors are temperature and light intensity. Other factors such as duration of exposure to atmospheric elements or motion may adversely effect the sensors' performance.

This model captures the concepts within the cooperative teleassistance domain set forth by Rogers and Murphy. Furthermore, the choice of frames provides the functionality needed in this type of problem solving, namely, storage of both static and dynamic data. Frames also allow the knowledge base to have attached procedures ,i.e., enhancement procedures. The choice of frames provides flexibility that may be necessary as the knowledge base grows in size and complexity.

```
*****************************************
*****   Output for Data Frame Contents ****
            (Listed by Sensor)


Sensor:                 Black & White
Scene:                  Student-Desk-Scene
Robot:                  George
Data_Type:              Image
Horz Size:              256 bits
Vert Size:              256 bits
Depth:                  8 bits



Sensor:                 Infra-red
Scene:                  Student-Desk-Scene
Robot:                  George
Data_Type:              Image
Horz Size:              640 bits
Vert Size:              480 bits
Depth:                  8 bits



Sensor:                 Ultrasonics
Scene:                  Student-Desk-Scene
Robot:                  George
Data_Type:              Numeric
Horz Size:              0 bits
Vert Size:              0 bits
Depth:                  0 bits
```

Figure 27: Output for the Data Frame

The knowledge base can be modified if additional concepts or attributes are need. Any attribute that does not link two concepts may be removed or added without compromising the original design.

CHAPTER 5

SUMMARY AND CONCLUSIONS

Summary

This work has described the design of a knowledge base that supports interactive exception handling for a cooperative teleassistance. The major focus was to identify and represent concepts specific to Rogers and Murphy's cooperative teleassistant [Rogers and Murphy, 1993]. In Chapter 2, a brief overview of Rogers and Murphy's cooperative teleassistance system was given to lay the foundation for the concepts that are represented in teleVIA's knowledge base. An explanation was provided for each component of the teleassistant as it relates to the type of information that must be stored in the knowledge base. In Chapter 3, the details of the teleVIA knowledge base design were presented. The choice of frames as a representation schema was explained, and the concepts represented in this schema and their relationships were defined. An explanation for each concept's attributes was given and examples were given for navigating through the knowledge base. Finally, the programs used to implement the knowledge base were described. Examples of the type of data

that may fill each frame's slots was shown and sample output for the knowledge base routines was given. Finally, an evaluation of the knowledge base was made.

Future Work

Future work in Rogers and Murphy's cooperative teleassistance will examine the use of a generic blackboard based on an object-oriented design. Given this, it is conceivable that knowledge base design presented here could be converted to the generic blackboard platform. In doing so, each frame could be represented as an object in the object-oriented system. The concepts represented here and objects, as defined in an object-oriented paradigm, [Cattel, 1991] are quite similar. Figure 28 shows the characteristics of an object in an object oriented paradigm compared to the concepts represented in the teleVIA knowledge base. These similarities suggest that such a conversion may be possible.

Conclusions

This work emphasizes the importance of building domain knowledge into intelligent, cooperative systems. Generally, domain knowledge is specific to a given application; therefore, knowledge cannot be built in until important facts and relationships within the domain are defined and organized. Because of this work, the teleVIA

An Object ...	A teleVIA Robot Frame ...
models real world entities	represents a real world entity
may have relationships	is related to a sensor by the 'has' relationship
may have behavior as well as data	performs tasks and has a name
has attributes	may have a failure
may have procedural attachments	may have an attached enhancement procedure

Figure 28: Objects vs. Concepts

system now has a repository of organized information that it can use to bring researchers one step closer to human and intelligent robot cooperation.

APPENDIX A

This appendix contains the C code for each frame structure in the knowledge base. The code is contained in the file named "zframes.h"

```
#ifndef ZFRAMES_H
#define ZFRAMES_H 1
/*************************************************************

   Name:         zframes.h
   Date:         April 26, 1994
   Purpose:      Templates for knowledge representation
                 frames
   Comments: data_frame, object_frame, scene_frame,
environment_frame, sensor_frame, robot_frame,

*************************************************************/

#include zkbase.h
#include zglobal.h


typedef struct test_s {
     int                 provided_by;
     struct scene       *models;
     int                 scene;
     int                 data_type;
     FILE                *curr_raw_image;
     FILE                *prev_raw_image;
     FILE                *curr_raw_numeric;
     FILE                *prev_raw_numeric;
     int                 horz_size;
     int                 vert_size;
     int                 depth;

} DATA_FRAME
 *DATA_P;


typedef struct {

     int     name;
     int     color;
     int     temperature;

} OBJECT_FRAME;
```

```
typedef struct scene {

     int              name;
     int              part_of;
     DATA_FRAME      *modeled_by;
     int              curr_object[NUM_OBJECTS + 1];

} SCENE_FRAME
 *SCENE_P;


typedef struct {

    int             name;
    SCENE_FRAME     has_scene[NUM_SCENES + 1 ];
    int             curr_scene;
    int             assoc_task[NUM_TASKS + 1];
    double          width;
    double          height;
    double          length;
    int             light_intensity;
    double          temperature;
    OBJECT_FRAME    object_list[NUM_OBJECTS + 1];

} ENVIRONMENT_FRAME;

ENVIRONMENT_FRAME   environment_list[NUM_ENVIRONMENTS + 1];


/*************************************************************

 A function pointer is used to call the enhancement functions.

*************************************************************/

typedef struct func_s {

     int         name;
     char        *function;

} ENHANCEMENT_NODE;


typedef struct {

     ENHANCEMENT_NODE  enhancement[MAX_ENHANCEMENTS];

} E_NODE;


typedef struct{

     int             name;
     int             part_of;
     int             usage;
     double          horz_fov;
     double          vert_fov;
     int             data_type;
     DATA_FRAME     provides[NUM_SCENES + 1];
     double          curr_evidence[NUM_BELIEFS];
     double          prev_evidence[NUM_BELIEFS];
     double          second_attempt_evidence[NUM_BELIEFS];
     E_NODE          e_list[NUM_SENSORS + 1];

} SENSOR_FRAME;
```

```
typedef struct {

     int                name;
     int                operates_in;
     int                given_task;
     SENSOR_FRAME       has_sensor[NUM_SENSORS + 1];
     int                curr_sensors_status[NUM_SENSORS + 1];
     int                failure_type;
     int                second_attempt_failure_type;

} ROBOT_FRAME;

ROBOT_FRAME  robot_list[NUM_ROBOTS + 1];


#endif
```

APPENDIX B

This appendix contains the possible slot values for the frames in the knowledge base.

```
#ifndef ZKBASE_H
#define ZKBASE_H 1
/*************************************************

    Name:           kbase.h
    Date:           April 26, 1994
    Purpose:        Holds definition for all knowledge
                    base information
    Comments:       Environment, Scene, Robot, Sensors


**********************************************************/

            /******* Environment *******/

/* Names */

#define NO_ENVIRONMENT      0
#define NUM_ENVIRONMENTS    1

#define LAB             1
#define WAREHOUSE       2


/* Temperature */

#define     NO_TEMPERATURE      0

#define     BELOW_AMBIENT       1
#define     AMBIENT             2
#define     ABOVE_AMBIENT       3


/* Light Intensity */

#define     NO_INTENSITY          0

#define     ON                  1
#define     OFF                 2
#define     DIM                 3

         /******* Scenes *******/

#define  NO_SCENE                    0
#define  NUM_SCENES                  4
```

```
#define  DRILL_PRESS_SCENE          1
#define  VCR_MONITOR_SCENE          2
#define  DOOR_SCENE                 3
#define  STUDENT_DESK_SCENE         4


/* Objects */

#define  NO_OBJECT              0
#define  NUM_OBJECTS            8

#define  VCR                    1
#define  MONITOR                2
#define  CABINET                3
#define  DRILL_PRESS            4
#define  DESK                   5
#define  CHAIR                  6
#define  DOOR                   7
#define  FIRE_EXTINGUISHER      8


/* Color */

#define      NO_COLOR               0

#define      BLACK                  1
#define      WHITE                  2
#define      RED                    3
#define      ORANGE                 4
#define      YELLOW                 5
#define      GREEN                  6
#define      BLUE                   7
#define      PURPLE                 8
#define      GREY                   9

             /*******  Robot    *******/

/* Names */

#define NO_ROBOT                    0
#define NUM_ROBOTS                  2

#define      CLEMENTINE             1
#define      GEORGE                 2


/* Task List   */


#define NO_TASK                     0
#define NUM_TASKS                   2

#define SECURITY_GUARD              1
#define MONITOR_TASK                2
```

```
    /******* Sensor *******/


/* Sensor Type */

#define NO_SENSOR               0
#define NUM_SENSORS             5
#define NUM_DATA                5

#define     BW                  1
#define     COLOR               2
#define     IR                  3
#define     US                  4
#define UV                      5

/* Sensor Usage    */

#define     NO_USAGE                        0

#define     DETECT_VISIBLE_LIGHT            1
#define     DETECT_RANGE                    2
#define     DETECT_HEAT                     3
#define DETECT_ENVIRONMENTAL_CHANGE         4



/* Data Type */

#define   NO_DATA       0

#define    IMAGE        1
#define    NUMERIC      2


 /* Enhancement */

/*************************************************************

There may be more than one enhancement associated with each
sensor. The maximum number of enhancements for each sensor is
given by MAX_ENHANCEMENTS.  This number is the maximum number
of enhancements that may be associated with each sensor.

*************************************************************/

#define MAX_ENHANCEMENTS   5

#define NO_ENHANCEMENT      0

#define BW_ENHANCE          1    /* BW */

#define COLOR_ENHANCE       2    /* COLOR */

#define FALSE_COLOR         3    /* IR */

#define  OCCUPANCY_GRID     4    /* US */

#define  POLAR_PLOT         5

#define UV_ENHANCE          6    /*  UV */
```

```
/****** Failure ******/

/*  Types of State Failure     */

#define NO_FAILURE        0

#define BELOW_MIN          1
#define MISSING_EVIDENCE   2
#define HIGH_CONFLICT      3
#define HIGHLY_UNCERTAIN   4



/*  Beliefs */
#define NUM_BELIEFS        3

#define SUPPORT            0
#define AGAINST            1
#define DONT_KNOW          2


/*  Constants      */

#define NUM_TRANSDUCERS       24     /*  US constants */

#define DEFAULT_NO_PULSE      250
#define DEFAULT_NOT_FIRING    300
#define DIST_BTW_TRANSDUCERS  10


/* Status Variables */

#define NOT_AVAIL             0

#define      ACTIVE           1
#define      INACTIVE         2
#define    SUSPECT            3

#define      TRUE             1
#define      FALSE            0

#define YES                    1
#define NO                     0

#endif
```

Appendix C

C code is for the INIT.C routine is included in this appendix.

```
/*****************************************************************

   Name:     init.c
   Date:     March 23, 1994
   Purpose:  Initializes all the Televia frames
   Comments: Object,Scene, Environment
             Data, Sensor, Robot

*****************************************************************/

#include <stdio.h>
#include <string.h>
#include "zframes.h"
#include "zkbase.h"
#include "zglobal.h"

extern int i;
extern int j;
extern int k;


extern int prn_frame;          /* selects frame to prn */


/*******************************************************
 *
 *   Initializes the contents of the robot frame
 *
 *******************************************************/

init_robot_frame(){

for( i = NO_ROBOT; i <= NUM_ROBOTS; i++){

       robot_list[i].name = NO_ROBOT;
       robot_list[i].operates_in = NO_ENVIRONMENT;
       robot_list[i].given_task = NO_TASK;
       robot_list[i].failure_type = NO_FAILURE;
       robot_list[i].second_attempt_failure_type = NO_FAILURE;


    for( j = NO_SENSOR; j <= NUM_SENSORS; j++){

       robot_list[i].curr_sensors_status[j] = NOT_AVAIL;


    }
```

```
}
}   /*       End Init_Robot_Frame      */

/*****************************************************
 *                                                   *
 *   Initializes the contents of the sensor frame    *
 *                                                   *
 *****************************************************/

init_sensor_frame(){

for( i = NO_ROBOT; i <=  NUM_ROBOTS; i++){

   for( j = NO_SENSOR; j <= NUM_SENSORS; j++){

      robot_list[i].has_sensor[j].name = NO_SENSOR;
      robot_list[i].has_sensor[j].part_of = NO_ROBOT;

      robot_list[i].has_sensor[j].usage = NO_USAGE;
      robot_list[i].has_sensor[j].horz_fov = 0;
      robot_list[i].has_sensor[j].vert_fov = 0;
      robot_list[i].has_sensor[j].data_type = NO_DATA;


   /* Beliefs */

robot_list[i].has_sensor[j].curr_evidence[SUPPORT]     = 0.00;
robot_list[i].has_sensor[j].curr_evidence[AGAINST]     = 0.00;
robot_list[i].has_sensor[j].curr_evidence[DONT_KNOW]   = 0.00;

robot_list[i].has_sensor[j].prev_evidence[SUPPORT]     = 0.00;
robot_list[i].has_sensor[j].prev_evidence[AGAINST]     = 0.00;
robot_list[i].has_sensor[j].prev_evidence[DONT_KNOW]   = 0.00;

robot_list[i].has_sensor[j].second_attempt_evidence[SUPPORT]   = 0.00;
robot_list[i].has_sensor[j].second_attempt_evidence[AGAINST]   = 0.00;
robot_list[i].has_sensor[j].second_attempt_evidence[DONT_KNOW]= 0.00;

      for( k = NO_ENHANCEMENT; k <= MAX_ENHANCEMENTS; k++){


      robot_list[i].has_sensor[j].e_list[j].enhancement[k].name
NO_ENHANCEMENT;

      robot_list[i].has_sensor[j].e_list[j].enhancement[k].function =
NIL_FUNCTION;

      } /* k*/
   } /* i*/
}  /* j */

}   /*       End Init_Sensor_Frame()   */


/*****************************************************
 *                                                   *
 *   Initializes the contents of the Data Frame      *
 *                                                   *
 *****************************************************/

init_data_frame(){
```

```
for( i = NO_ROBOT; i <= NUM_ROBOTS; i++){
    for( j = NO_SENSOR; j <= NUM_SENSORS; j++){

      for( k = NO_SCENE; k <= NUM_SCENES; k++){

        robot_list[i].has_sensor[j].provides[k].provided_by = NO_SENSOR;
        robot_list[i].has_sensor[j].provides[k].models = NIL_SCENE;
        robot_list[i].has_sensor[j].provides[k].scene= NO_SCENE;
        robot_list[i].has_sensor[j].provides[k].data_type = NO_DATA;
        robot_list[i].has_sensor[j].provides[k].horz_size = 0;
        robot_list[i].has_sensor[j].provides[k].vert_size = 0;
        robot_list[i].has_sensor[j].provides[k].depth = 0;
        robot_list[i].has_sensor[j].provides[k].default_not_firing = 0;
        robot_list[i].has_sensor[j].provides[k].default_no_pulse = 0;
        robot_list[i].has_sensor[j].provides[k].distance_btw_data_points=
0;

      }
    }
}

}



/*****************************************************************************
*
*
*   Purpose:  Initializes the contents of the Environment Frame
*
*
*****************************************************************************
/

init_environment_frame(){

   for (i = NO_ENVIRONMENT; i <=  NUM_ENVIRONMENTS; i++) {

        environment_list[i].name = NO_ENVIRONMENT;

        environment_list[i].curr_scene = NO_SCENE;
        environment_list[i].width  = 0;
        environment_list[i].height = 0;
        environment_list[i].length = 0;
        environment_list[i].light_intensity = NO_INTENSITY;
        environment_list[i].temperature = NO_TEMPERATURE;


    for (j = NO_TASK; j <=  NUM_TASKS ; j++) {
      environment_list[i].assoc_task[j] = NO_TASK;
    }
    }

}
```

```
/*********************************************************************
 *
 *    Purpose: Initializes the contents of the Percept Frame
 *
 *********************************************************************/

init_scene_frame(){

    for (i = NO_ENVIRONMENT; i <= NUM_ENVIRONMENTS; i++) {

       for (j = NO_SCENE; j <= NUM_SCENES; j++) {


          environment_list[i].has_scene[j].name = NO_SCENE;
          environment_list[i].has_scene[j].part_of = NO_ENVIRONMENT;


          environment_list[i].has_scene[j].modeled_by = NIL_DATA;


          for (k = NO_OBJECT; k<=     NUM_OBJECTS; k++) {
             environment_list[i].has_scene[j].curr_object[k] = FALSE;
          }

    }


  }
}


/*********************************************************************
 *
 *    Purpose: Initializes the contents of the Object Frame
 *
 *********************************************************************/
init_object_frame(){

  for (i = NO_ENVIRONMENT; i <= NUM_ENVIRONMENTS + 1; i++) {

   for (j = NO_SCENE; j <= NUM_SCENES + 1; j++){

    for (k = NO_OBJECT; k <= NUM_OBJECTS + 1; k++){

      environment_list[i].object_list[k].name = NO_OBJECT;

      environment_list[i].object_list[k].color = NO_COLOR;
      environment_list[i].object_list[k].temperature = NO_TEMPERATURE;

       }

   }
  }
}
```

APPENDIX D

This appendix contains code for the data.c routine.

```
/*****************************************************************
   Name:            data.c
   Date:            April 26, 1994
   Purpose:         Instantiates all the Knowledge Base Frames
   Comments:        Robot, Sensor, Data,
                    Environment, Percept,

*****************************************************************/

#include <stdio.h>
#include <string.h>
#include "zframes.h"
#include "zkbase.h"
#include "zglobal.h"

extern int i;
extern int j;
extern int k;


extern int prn_frame;          /* selects frame to prn */


/*******************************************************
 *
 *       Instantiates the Robot Frame
 *
 *  The following function, instantiates the robot frame
 *  for Clementine and George.
 *
 *******************************************************/


put_robot_data() {

int x;


/* Clementine's Data */

robot_list[CLEMENTINE].name = CLEMENTINE;
robot_list[CLEMENTINE].operates_in = LAB;
robot_list[CLEMENTINE].given_task = SECURITY_GUARD;

robot_list[CLEMENTINE].curr_sensors_status[BW]     = ACTIVE;
robot_list[CLEMENTINE].curr_sensors_status[COLOR] = ACTIVE;
robot_list[CLEMENTINE].curr_sensors_status[IR]     = NOT_AVAIL;
robot_list[CLEMENTINE].curr_sensors_status[US]     = ACTIVE;
robot_list[CLEMENTINE].curr_sensors_status[UV]     = NOT_AVAIL;
```

```
/* George's Data */

      robot_list[GEORGE].name         = GEORGE;
      robot_list[GEORGE].operates_in =  LAB;
      robot_list[GEORGE].given_task   = SECURITY_GUARD;

robot_list[GEORGE].curr_sensors_status[BW]         = ACTIVE;
robot_list[GEORGE].curr_sensors_status[COLOR]      = ACTIVE;
robot_list[GEORGE].curr_sensors_status[IR]         = ACTIVE;
robot_list[GEORGE].curr_sensors_status[US]         = ACTIVE;
robot_list[GEORGE].curr_sensors_status[UV]         = ACTIVE;

}

/*******************************************************
 *
 *   Instantiate  the Sensor frame
 *
 *******************************************************/

put_sensor_data(){


i = CLEMENTINE;
j = BW;


      robot_list[i].has_sensor[j].part_of = CLEMENTINE;
      robot_list[i].has_sensor[j].name = BW;

      robot_list[i].has_sensor[j].usage = DETECT_VISIBLE_LIGHT;
      robot_list[i].has_sensor[j].horz_fov = 23.5;
      robot_list[i].has_sensor[j].vert_fov = 25.0;
      robot_list[i].has_sensor[j].data_type = IMAGE;


j = COLOR;

      robot_list[i].has_sensor[j].part_of = CLEMENTINE;
      robot_list[i].has_sensor[j].name = COLOR;

      robot_list[i].has_sensor[j].usage = DETECT_VISIBLE_LIGHT;
      robot_list[i].has_sensor[j].horz_fov = 46.4;
      robot_list[i].has_sensor[j].vert_fov = 40.0;
      robot_list[i].has_sensor[j].data_type = IMAGE;



j = US;

      robot_list[i].has_sensor[j].part_of = CLEMENTINE;
      robot_list[i].has_sensor[j].name = US;

      robot_list[i].has_sensor[j].usage = DETECT_RANGE;
      robot_list[i].has_sensor[j].horz_fov = 0;
      robot_list[i].has_sensor[j].vert_fov = 0;
      robot_list[i].has_sensor[j].data_type = NUMERIC;


/*********************************************************************
```

```
  BW Enhancement:

   Insert the enhancement function(s) for BW in enhancement list.
   bw_enhance() is a dummy funciton - it is a place holder for a
   real black and white enhancement function.

*********************************************************************/

/* Input the enhancement for availible each sensor */

robot_list[CLEMENTINE].has_sensor[BW].e_list[BW].enhancement[1].name
=
  BW_ENHANCE;

robot_list[CLEMENTINE].has_sensor[BW].e_list[BW].enhancement[1].function =
 (char*) bw_enhance();


/*********************************************************************

  COLOR Enhancement:

   Insert the enhancement function(s) for COLOR in enhancement list.
   color_enhance() is a dummy funciton - it is a place holder for a
   real black and white enhancement function.

*********************************************************************/


robot_list[CLEMENTINE].has_sensor[COLOR].e_list[COLOR].enhancement[1].name
=
       COLOR_ENHANCE;


robot_list[CLEMENTINE].has_sensor[COLOR].e_list[COLOR].enhancement[1].funct
ion =
      (char*) color_enhance();

/*********************************************************************

  US Enhancement:

   Insert the enhancement function(s) for US in enhancement list.
    There are two enhancements for the ultrasonice camera. They are a
    and occupancy grid and polar plot.      The occupancy grid presents a
    bird's eye view of what the robot has sensed.  The polar plot is a
    graphical representation of the US readings.

*********************************************************************/


robot_list[CLEMENTINE].has_sensor[US].e_list[US].enhancement[1].name
 =
  OCCUPANCY_GRID;

robot_list[CLEMENTINE].has_sensor[US].e_list[US].enhancement[1].function =
  (char*) occupancy_grid();

robot_list[CLEMENTINE].has_sensor[US].e_list[US].enhancement[2].name
 =
  POLAR_PLOT;

robot_list[CLEMENTINE].has_sensor[US].e_list[US].enhancement[2].function =
```

```
 (char*) polar_plot();



i = GEORGE;
j = BW;

      robot_list[i].has_sensor[j].part_of = GEORGE;
      robot_list[i].has_sensor[j].name = BW;

      robot_list[i].has_sensor[j].usage = DETECT_VISIBLE_LIGHT;
      robot_list[i].has_sensor[j].horz_fov = 23.5;
      robot_list[i].has_sensor[j].vert_fov = 25.0;
      robot_list[i].has_sensor[j].data_type = IMAGE;

j = COLOR;

      robot_list[i].has_sensor[j].part_of = GEORGE;
      robot_list[i].has_sensor[j].name = COLOR;

      robot_list[i].has_sensor[j].usage = DETECT_VISIBLE_LIGHT;
      robot_list[i].has_sensor[j].horz_fov = 46.4;
      robot_list[i].has_sensor[j].vert_fov = 40.0;
      robot_list[i].has_sensor[j].data_type = IMAGE;

j = IR;

      robot_list[i].has_sensor[j].part_of = GEORGE;
      robot_list[i].has_sensor[j].name = IR;

      robot_list[i].has_sensor[j].usage = DETECT_HEAT;
      robot_list[i].has_sensor[j].horz_fov = 24;
      robot_list[i].has_sensor[j].vert_fov = 24;
      robot_list[i].has_sensor[j].data_type = IMAGE;

j = US;

      robot_list[i].has_sensor[j].part_of = GEORGE;
      robot_list[i].has_sensor[j].name = US;

      robot_list[i].has_sensor[j].usage = DETECT_RANGE;
      robot_list[i].has_sensor[j].horz_fov = 0;
      robot_list[i].has_sensor[j].vert_fov = 0;
      robot_list[i].has_sensor[j].data_type = NUMERIC;

j = UV;

      robot_list[i].has_sensor[j].part_of = GEORGE;
      robot_list[i].has_sensor[j].name = UV;

      robot_list[i].has_sensor[j].usage = DETECT_ENVIRONMENTAL_CHANGE;
      robot_list[i].has_sensor[j].horz_fov = 40.0;
      robot_list[i].has_sensor[j].vert_fov = 40.0;
      robot_list[i].has_sensor[j].data_type = IMAGE;

/*********************************************************************
```

```
  BW Enhancement:

   Insert the enhancement function(s) for BW in enhancement list.
   bw_enhance() is a dummy funciton - it is a place holder for a
   real black and white enhancement function.

*************************************************************************/


robot_list[GEORGE].has_sensor[BW].e_list[BW].enhancement[1].name        =
  BW_ENHANCE;

robot_list[GEORGE].has_sensor[BW].e_list[BW].enhancement[1].function =
  (char*) bw_enhance();


/*************************************************************

    COLOR enhancements:

   Insert the enhancement function for COLOR images
   in enhancement list.  This function is a dummy function - It is
   a place holder for a real function.

*************************************************************************/

      :
robot_list[GEORGE].has_sensor[COLOR].e_list[COLOR].enhancement[1].name
=
       COLOR_ENHANCE;


robot_list[GEORGE].has_sensor[COLOR].e_list[COLOR].enhancement[1].function
=
      (char*) color_enhance();


/*************************************************************

    IR enhancements:

   Insert the enhancement function for IR images
   in enhancement list.

*************************************************************************/

      :
       robot_list[GEORGE].has_sensor[IR].e_list[IR].enhancement[1].name
=
       FALSE_COLOR;

       robot_list[GEORGE].has_sensor[IR].e_list[IR].enhancement[1].function
=
       (char*) false_color();

/*************************************************************

    US enhancements:

    Insert the enhancement function for
    US data in enhancement list
```

```
**********************************************************************/


      robot_list[GEORGE].has_sensor[US].e_list[US].enhancement[1].name
=
       OCCUPANCY_GRID;
      robot_list[GEORGE].has_sensor[US].e_list[US].enhancement[1].function
=
      (char*) occupancy_grid();

      robot_list[GEORGE].has_sensor[US].e_list[US].enhancement[2].name
=
       POLAR_PLOT;
      robot_list[GEORGE].has_sensor[US].e_list[US].enhancement[2].function
=
      (char*) polar_plot();

/*********************************************************************

    ULTRAVIOLET enhancements:

    Insert the enhancement function for
    ULTRAVIOLET data in enhancement list. This is a dummy function -
    It is a place holder for a real function

**********************************************************************/


      robot_list[GEORGE].has_sensor[UV].e_list[UV].enhancement[1].name
=
       UV_ENHANCE;
      robot_list[GEORGE].has_sensor[UV].e_list[UV].enhancement[1].function
=
      (char*) uv_enhance();


}

/*    End put_sensor data()    */

/*****************************************************
 *                                          *
 *   Instantiates the Data Frame                *
 *                                          *
 *****************************************************/

put_data(){

i = CLEMENTINE;
j = BW;
k = DRILL_PRESS_SCENE;


        robot_list[i].has_sensor[j].provides[k].provided_by = BW;

        robot_list[i].has_sensor[j].provides[k].scene = DRILL_PRESS_SCENE;
        robot_list[i].has_sensor[j].provides[k].data_type = IMAGE;
        robot_list[i].has_sensor[j].provides[k].horz_size = 256;
        robot_list[i].has_sensor[j].provides[k].vert_size = 256;
```

```
        robot_list[i].has_sensor[j].provides[k].depth = 8;


j = COLOR;
k = DRILL_PRESS_SCENE;

        robot_list[i].has_sensor[j].provides[k].provided_by = COLOR;

        robot_list[i].has_sensor[j].provides[k].scene = DRILL_PRESS_SCENE;
        robot_list[i].has_sensor[j].provides[k].data_type = IMAGE;
        robot_list[i].has_sensor[j].provides[k].horz_size = 640;
        robot_list[i].has_sensor[j].provides[k].vert_size = 480;
        robot_list[i].has_sensor[j].provides[k].depth = 8;


j = US;
k = DRILL_PRESS_SCENE;

        robot_list[i].has_sensor[j].provides[k].provided_by = US;

        robot_list[i].has_sensor[j].provides[k].scene = DRILL_PRESS_SCENE;
        robot_list[i].has_sensor[j].provides[k].data_type = NUMERIC;
        robot_list[i].has_sensor[j].provides[k].horz_size = 0;
        robot_list[i].has_sensor[j].provides[k].vert_size = 0;
        robot_list[i].has_sensor[j].provides[k].depth = 0;


i = GEORGE;
j = BW;
k = DRILL_PRESS_SCENE;

        robot_list[i].has_sensor[j].provides[k].provided_by = BW;

        robot_list[i].has_sensor[j].provides[k].scene = DRILL_PRESS_SCENE;
        robot_list[i].has_sensor[j].provides[k].data_type = IMAGE;
        robot_list[i].has_sensor[j].provides[k].horz_size = 640;
        robot_list[i].has_sensor[j].provides[k].vert_size = 480;
        robot_list[i].has_sensor[j].provides[k].depth = 8;


j = COLOR;
k = DRILL_PRESS_SCENE;


        robot_list[i].has_sensor[j].provides[k].provided_by =  COLOR;

        robot_list[i].has_sensor[j].provides[k].scene = DRILL_PRESS_SCENE;
        robot_list[i].has_sensor[j].provides[k].data_type = IMAGE;
        robot_list[i].has_sensor[j].provides[k].horz_size = 640;
        robot_list[i].has_sensor[j].provides[k].vert_size = 480;
        robot_list[i].has_sensor[j].provides[k].depth = 8;


j = IR;
k = DRILL_PRESS_SCENE;


        robot_list[i].has_sensor[j].provides[k].provided_by = IR;
        robot_list[i].has_sensor[j].provides[k].scene = DRILL_PRESS_SCENE;
        robot_list[i].has_sensor[j].provides[k].data_type = 1;
        robot_list[i].has_sensor[j].provides[k].horz_size = 478;
        robot_list[i].has_sensor[j].provides[k].vert_size = 397;
        robot_list[i].has_sensor[j].provides[k].depth = 8;
```

```
j = US;
k = DRILL_PRESS_SCENE;


        robot_list[i].has_sensor[j].provides[k].provided_by =  US;
         robot_list[i].has_sensor[j].provides[k].scene = DRILL_PRESS_SCENE;
         robot_list[i].has_sensor[j].provides[k].data_type = NUMERIC;
         robot_list[i].has_sensor[j].provides[k].horz_size = 0;
         robot_list[i].has_sensor[j].provides[k].vert_size = 0;
         robot_list[i].has_sensor[j].provides[k].depth = 0;


j = UV;
k = DRILL_PRESS_SCENE;

         robot_list[i].has_sensor[j].provides[k].provided_by   = UV;
         robot_list[i].has_sensor[j].provides[k].scene = DRILL_PRESS_SCENE;
         robot_list[i].has_sensor[j].provides[k].data_type = IMAGE;
         robot_list[i].has_sensor[j].provides[k].horz_size = 40.0;
         robot_list[i].has_sensor[j].provides[k].vert_size = 60.0;
         robot_list[i].has_sensor[j].provides[k].depth = 8;

/*************************************************************/



}


/*****************************************************************************
*
*
*   Purpose: Instantiates the  the Environment Frame
*
*
*****************************************************************************/

put_environment_data(){


i = LAB;

         environment_list[i].name = LAB;

         environment_list[i].width    = 12;
         environment_list[i].height = 10;
         environment_list[i].length = 12;
         environment_list[i].light_intensity = ON;
         environment_list[i].temperature = 68.5;
         environment_list[i].curr_scene = DRILL_PRESS_SCENE;


       environment_list[i].assoc_task[SECURITY_GUARD] = TRUE;
       environment_list[i].assoc_task[MONITOR_TASK] = TRUE;


}


/*********************************************************************
 *
```

```
 *     Purpose: Instantiates  the Scene Frame
 *
 ************************************************************************/

put_scene_data(){


i = LAB;
j = DRILL_PRESS_SCENE;

          environment_list[i].has_scene[j].name = DRILL_PRESS_SCENE;
          environment_list[i].has_scene[j].part_of      = LAB;

i = LAB;
j = DOOR_SCENE;

          environment_list[i].has_scene[j].name = DOOR_SCENE;
          environment_list[i].has_scene[j].part_of      = LAB;

i = LAB;
j = VCR_MONITOR_SCENE;

          environment_list[i].has_scene[j].name = VCR_MONITOR_SCENE;
          environment_list[i].has_scene[j].part_of      = LAB;

i = LAB;
j = STUDENT_DESK_SCENE;

          environment_list[i].has_scene[j].name = STUDENT_DESK_SCENE;
          environment_list[i].has_scene[j].part_of      = LAB;


}

/************************************************************************
 *
 *     Purpose: Instantiates the Object Frame
 *
 ************************************************************************/



put_object_data(){

i = LAB;

k = DRILL_PRESS;


      environment_list[i].object_list[k].name = DRILL_PRESS;
      environment_list[i].object_list[k].color = GREY;
      environment_list[i].object_list[k].temperature = ABOVE_AMBIENT;


k = VCR;

      environment_list[i].object_list[k].name = VCR;
      environment_list[i].object_list[k].color = BLACK;
      environment_list[i].object_list[k].temperature = AMBIENT;


k = MONITOR;
```

```
        environment_list[i].object_list[k].name = MONITOR;
        environment_list[i].object_list[k].color = GREY;
        environment_list[i].object_list[k].temperature = AMBIENT;

k = CABINET;
        environment_list[i].object_list[k].name = CABINET;
        environment_list[i].object_list[k].color = BLUE;
        environment_list[i].object_list[k].temperature = AMBIENT;


k = DOOR;

        environment_list[i].object_list[k].name = DOOR;
        environment_list[i].object_list[k].color = BLACK;
        environment_list[i].object_list[k].temperature = AMBIENT;


k = FIRE_EXTINGUISHER;



        environment_list[i].object_list[k].name = FIRE_EXTINGUISHER;
        environment_list[i].object_list[k].color = RED;
        environment_list[i].object_list[k].temperature = AMBIENT;


k = DESK;


        environment_list[i].object_list[k].name = DESK;
        environment_list[i].object_list[k].color = BLUE;
        environment_list[i].object_list[k].temperature = AMBIENT;

k = CHAIR;

        environment_list[i].object_list[k].name = CHAIR;
        environment_list[i].object_list[k].color = YELLOW;
        environment_list[i].object_list[k].temperature = AMBIENT;

}


/*********************************************************************

    NOT USED

    The following two Link(*) functions, provides the connection
    between the scene and data frames.      Every data frame that
    contains data for some scene, a link is made via "models" pointer.
    For every scenes frame, that has some data, a link is made via the
    "modeled_by" pointer.

*********************************************************************/

link_data_with_scene() {

int scene_index;
int envir_index;

/* Init varaibles */

envir_index = LAB;
```

```
   for (i = 1; i <= NUM_ROBOTS; i++) {
      for (j = 1; j <= NUM_SENSORS; j++) {
      for (k = 1; k<= NUM_SCENES; k++) {


/* Find the scene index */

scene_index = robot_list[i].has_sensor[j].provides[k].scene;


    /* If scene_index = 0 then do not link data and scene frames */

    if(scene_index != 0) {


       /* Link data and scene frame */
      robot_list[i].has_sensor[j].provides[k].models =
            &(environment_list[envir_index].has_scene[scene_index]);

     } /* End if */
   } /* for i */
 } /* for j */
} /* for k */

} /* End Link_ data_with_Percept() */

/*****************************************************************/

link_scene_with_data(){

int scene_index;
int envir_index;

/* Init varaibles */

envir_index = LAB;


   for (i = 1; i <= NUM_ROBOTS; i++) {
      for (j = 1; j <= NUM_SENSORS; j++) {
      for (k = 1; k<= NUM_SCENES; k++) {

       scene_index = robot_list[i].has_sensor[j].provides[k].scene;

/* If scene_index = 0 then do not link scene with data frame */

    if(scene_index != 0) {

      environment_list[envir_index].has_scene[scene_index].modeled_by =
       &(robot_list[i].has_sensor[j].provides[scene_index]);

   }
  }
 }
}
}
```

APPENDIX E

This appendix contains code for the update.c routine.

```
/*****************************************************************
    Name:           update.c
    Date:           March 30, 1994
    Purpose:        Updates knowledge base when there is an a repaired
                    or replaced plan or when failure occurs.
    Comments:       get_new_sensors(), get_new_scene(), get_failure()
                  update_objects(), scenario()
*****************************************************************/

#include <stdio.h>
#include <string.h>
#include "zframes.h"
#include "zkbase.h"
#include "zglobal.h"


/*******************************************************
      The procedure ensures that the correct objects are
listed in the current object list whenever the current scene changes.
*******************************************************************/
update_object(i,scene)
    int i, scene;
{


      if (scene == 1) {
            environment_list[i].has_scene[scene].curr_object[1] = FALSE;
            environment_list[i].has_scene[scene].curr_object[2] = FALSE;
            environment_list[i].has_scene[scene].curr_object[3] = FALSE;
            environment_list[i].has_scene[scene].curr_object[4] = TRUE;
            environment_list[i].has_scene[scene].curr_object[5] = FALSE;
            environment_list[i].has_scene[scene].curr_object[6] = FALSE;
            environment_list[i].has_scene[scene].curr_object[7] = FALSE;
            environment_list[i].has_scene[scene].curr_object[8] = FALSE;



}
      if (scene == 2) {

            environment_list[i].has_scene[scene].curr_object[1] = TRUE;
            environment_list[i].has_scene[scene].curr_object[2] = TRUE;
            environment_list[i].has_scene[scene].curr_object[3] = FALSE;
```

```
            environment_list[i].has_scene[scene].curr_object[4] = FALSE;
            environment_list[i].has_scene[scene].curr_object[5] = FALSE;
            environment_list[i].has_scene[scene].curr_object[6] = FALSE;
            environment_list[i].has_scene[scene].curr_object[7] = FALSE;
            environment_list[i].has_scene[scene].curr_object[8] = FALSE;

}
      if (scene == 3) {

            environment_list[i].has_scene[scene].curr_object[1] = FALSE;
            environment_list[i].has_scene[scene].curr_object[2] = FALSE;
            environment_list[i].has_scene[scene].curr_object[3] = FALSE;
            environment_list[i].has_scene[scene].curr_object[4] = FALSE;
            environment_list[i].has_scene[scene].curr_object[5] = FALSE;
            environment_list[i].has_scene[scene].curr_object[6] = FALSE;
            environment_list[i].has_scene[scene].curr_object[7] = TRUE;
            environment_list[i].has_scene[scene].curr_object[8] = TRUE;
}
      if (scene == 4) {

            environment_list[i].has_scene[scene].curr_object[1] = FALSE;
            environment_list[i].has_scene[scene].curr_object[2] = FALSE;
            environment_list[i].has_scene[scene].curr_object[3] = TRUE;
            environment_list[i].has_scene[scene].curr_object[4] = FALSE;
            environment_list[i].has_scene[scene].curr_object[5] = TRUE;
            environment_list[i].has_scene[scene].curr_object[6] = TRUE;
            environment_list[i].has_scene[scene].curr_object[7] = FALSE;
            environment_list[i].has_scene[scene].curr_object[8] = FALSE;

}

} /* End Update Object */

/*************     Examples of failure  **********/

/*************************************************************

    Collected data while facing the wrong part of the room.

*************************************************************/

scenario1() {

int robot, j, k;

robot = GEORGE;


/* Simulating the retreival of activation conditions
   in place prior to the failure  */

    robot_list[robot].curr_sensors_status[BW]       = SUSPECT;
    robot_list[robot].curr_sensors_status[COLOR]       = INACTIVE;
    robot_list[robot].curr_sensors_status[IR]       = SUSPECT;
    robot_list[robot].curr_sensors_status[US]       = ACTIVE;
    robot_list[robot].curr_sensors_status[UV]       = INACTIVE;


    /* Type of failure */

    robot_list[robot].failure_type = HIGH_CONFLICT;
```

```
    /* Beliefs obtained upon failure */

    robot_list[robot].has_sensor[BW].curr_evidence[SUPPORT]    = 0.00;
    robot_list[robot].has_sensor[BW].curr_evidence[AGAINST]    = 1.00;
    robot_list[robot].has_sensor[BW].curr_evidence[DONT_KNOW]  = 0.00;

    robot_list[robot].has_sensor[IR].curr_evidence[SUPPORT]    = 0.50;
    robot_list[robot].has_sensor[IR].curr_evidence[AGAINST]    = 0.00;
    robot_list[robot].has_sensor[IR].curr_evidence[DONT_KNOW]  = 0.50;

    robot_list[robot].has_sensor[US].curr_evidence[SUPPORT]    = 0.24;
    robot_list[robot].has_sensor[US].curr_evidence[AGAINST]    = 0.01;
    robot_list[robot].has_sensor[US].curr_evidence[DONT_KNOW]  = 0.75;


}

/*********************************************************************

    Simulated a sensor malfunction by taking plastic over the
    lens of the BW camera.  In this scenario, the initial sensing
    plan is replaced with a backup plan which includes the color
    sensor in.
*********************************************************************/

 scenario2(){

int j;

int robot = GEORGE;


/* Simulating the retreival of activation conditions for the
   backup plan */

    robot_list[robot].curr_sensors_status[BW]       =    SUSPECT;
    robot_list[robot].curr_sensors_status[COLOR]       = SUSPECT;
    robot_list[robot].curr_sensors_status[IR]       =    ACTIVE;
    robot_list[robot].curr_sensors_status[US]       =    ACTIVE;
    robot_list[robot].curr_sensors_status[UV]       =    INACTIVE;



  /* Type of failure */
   robot_list[robot].failure_type = BELOW_MIN;
   robot_list[robot].second_attempt_failure_type = HIGH_CONFLICT;


    /* Beliefs */


    robot_list[robot].has_sensor[BW].curr_evidence[SUPPORT]    = 0.44;
    robot_list[robot].has_sensor[BW].curr_evidence[AGAINST]    = 0.56;
    robot_list[robot].has_sensor[BW].curr_evidence[DONT_KNOW]  = 0.00;

    robot_list[robot].has_sensor[IR].curr_evidence[SUPPORT]    = 0.50;
    robot_list[robot].has_sensor[IR].curr_evidence[AGAINST]    = 0.00;
    robot_list[robot].has_sensor[IR].curr_evidence[DONT_KNOW]  = 0.50;

    robot_list[robot].has_sensor[US].curr_evidence[SUPPORT]    = 0.24;
    robot_list[robot].has_sensor[US].curr_evidence[AGAINST]    = 0.01;
```

```
    robot_list[robot].has_sensor[US].curr_evidence[DONT_KNOW] = 0.75;

/*  SECOND ATTEMPT  EVIDENCE */

robot_list[robot].has_sensor[BW].second_attempt_evidence[SUPPORT]   =0.44;
robot_list[robot].has_sensor[BW].second_attempt_evidence[AGAINST]   =0.56;

robot_list[robot].has_sensor[BW].second_attempt_evidence[DONT_KNOW]=0.00;

robot_list[robot].has_sensor[COLOR].second_attempt_evidence[SUPPORT] =0.68;
robot_list[robot].has_sensor[COLOR].second_attempt_evidence[AGAINST]
=0.32;

robot_list[robot].has_sensor[COLOR].second_attempt_evidence[DONT_KNOW]
=0.00;

robot_list[robot].has_sensor[IR].second_attempt_evidence[SUPPORT] =0.50;
robot_list[robot].has_sensor[IR].second_attempt_evidence[AGAINST] =0.00;
robot_list[robot].has_sensor[IR].second_attempt_evidence[DONT_KNOW] = 0.50;

robot_list[robot].has_sensor[US].second_attempt_evidence[SUPPORT] = 0.25;
robot_list[robot].has_sensor[US].second_attempt_evidence[AGAINST] = 0.00;
robot_list[robot].has_sensor[US].second_attempt_evidence[DONT_KNOW] = 0.75;

}


/*****************************************************************

    Lights were turned out to simulate an environmental chnage.

*****************************************************************/


scenario3() {

int j;
int robot = GEORGE;


/* Simulating the retreival of activation conditions in place
   prior to the failure  */

      robot_list[robot].curr_sensors_status[BW]      = ACTIVE;
      robot_list[robot].curr_sensors_status[COLOR]     = INACTIVE;
      robot_list[robot].curr_sensors_status[IR]      = ACTIVE;
      robot_list[robot].curr_sensors_status[US]      = ACTIVE;
      robot_list[robot].curr_sensors_status[UV]      = SUSPECT;


   /* Type of failure */

    robot_list[robot].failure_type = MISSING_EVIDENCE;


    /* Beliefs */

    robot_list[robot].has_sensor[BW].curr_evidence[SUPPORT]    = 0.00;
    robot_list[robot].has_sensor[BW].curr_evidence[AGAINST]    = 0.00;
    robot_list[robot].has_sensor[BW].curr_evidence[DONT_KNOW] = 1.00;

    robot_list[robot].has_sensor[IR].curr_evidence[SUPPORT]    = 0.50;
```

```
    robot_list[robot].has_sensor[IR].curr_evidence[AGAINST]    = 0.00;
    robot_list[robot].has_sensor[IR].curr_evidence[DONT_KNOW] = 0.50;

    robot_list[robot].has_sensor[US].curr_evidence[SUPPORT]    = 0.24;
    robot_list[robot].has_sensor[US].curr_evidence[AGAINST]    = 0.01;
    robot_list[robot].has_sensor[US].curr_evidence[DONT_KNOW] = 0.75;


}

/*********************************************************************

    US causing problem.  One transducer giving spurious
    readings.

*********************************************************************/

scenario4() {

int j;
int robot = CLEMENTINE;


/* Simulating the retreival of activation conditions in place prior to
   the failure     */

     robot_list[robot].curr_sensors_status[BW]       = ACTIVE;
     robot_list[robot].curr_sensors_status[COLOR]       = ACTIVE;
     robot_list[robot].curr_sensors_status[IR]       = NOT_AVAIL;
     robot_list[robot].curr_sensors_status [US]      = SUSPECT;
     robot_list[robot].curr_sensors_status[UV]       = NOT_AVAIL;


   /* Type of failure */

    robot_list[robot].failure_type = HIGHLY_UNCERTAIN;

}

/*********************************************************************

 Choice of enhancement is based on knowledge about the sensor.

*********************************************************************/


scenario5() {

int j;
int robot = GEORGE;


/* Simulating the retreival of activation conditions in place
   prior to the failure  */

    robot_list[robot].curr_sensors_status[BW]       = ACTIVE;
    robot_list[robot].curr_sensors_status[COLOR]       = INACTIVE;
    robot_list[robot].curr_sensors_status[IR]       = SUSPECT;
    robot_list[robot].curr_sensors_status[US]       = ACTIVE;
    robot_list[robot].curr_sensors_status[UV]       = INACTIVE;


    /* Type of Failure */
```

```
    robot_list[robot].failure_type = HIGHLY_UNCERTAIN;

}
```

APPENDIX F

This appendix contains code for the print.c routine.

```
/*************************************************************************
    Name:           print.c
    Date:           April 9, 1994
    Purpose:        Prints instances of the knowledge base frames

    Comments:       Menus and print functions

*************************************************************************/


#include <stdio.h>
#include <string.h>
#include "zframes.h"
#include "zkbase.h"
#include "zglobal.h"

int scene;

int  robot;                /* selects the current robot frames */

int i;                           /* loop counters */
int j;
int k;
int l;

int prn_frame;                    /* selects frame to prn */
int scenario;                     /* selects failure */

int display_current_robots;     /* sets display flags */
int display_current_sensors;
int display_current_data;
int display_current_environment;
int display_current_scenes;
int display_current_objects;

/*************************************
 *
 *   Name:   choose_frame
 *   Purpose:      Selects a frame to print
 *   PreCond:      None
 *
 *************************************/

choose_frame() {

prn_frame = 0;                          /*  Initialize the print variables
*/
```

```
while (prn_frame > 9 || prn_frame < 1) {

display_current_robots       = FALSE;              /* initialize display
flags */
display_current_sensors     = FALSE;
display_current_data         = FALSE;
display_current_environment = FALSE;
display_current_scenes       = FALSE;

     printf("\n\n*****    Choose a Frame to Print    *****\n\n");


     printf(" 1 - Environment\n");
     printf(" 2 - Scene\n\n");

     printf("*** Clementine ***\n");

     printf(" 3 - Robot\n");
     printf(" 4 - Sensor\n");
     printf(" 5 - Data\n\n");

     printf("*** George ***\n");

      printf(" 6 - Robot\n");
     printf(" 7 - Sensor\n");
     printf(" 8 - Data\n\n");

     printf(" 9 - None (Quit)\n");


     printf("\n Select a number corresponding ");
     printf("to the frame you want to print: => ");

     scanf("%d",&prn_frame);

} /* while */

if(prn_frame == 9) exit(1);

} /*  End Choose Frame    */


/*************************************************************

    Simulates failure in the teleSFX system for the 5
    scenarios given in the paper by Robin and Erika.
    This function is an interface that allows you to select
    a failure to simulate.

*************************************************************/

choose_scenario() {


scenario = 0;


while (scenario > 6 || scenario      < 1) {

     printf("\n\n*****    Choose Scenario    *****\n");

     printf("1 - Wrong Scene\n");
     printf("2 - Lens Covered - Second Recovery Attempt\n");
```

```
      printf("3 - Lights Out\n");
      printf("4 - Fluctuating Readings\n");
      printf("5 - IR Enhancement\n");
      printf("6 - None (Quit)\n");


      printf("\n Select a number corresponding ");
      printf("to the failure you want to simulate: => ");

      scanf("%d",&scenario);

      if (scenario == 1) {
        scenario1();
      }

      if (scenario == 2) {

       scenario2();

      }
     if (scenario == 3) {

       scenario3();

      }
      if (scenario == 4) {
       scenario4();
      }

      if (scenario == 5) {
       scenario5();
      }

     if (scenario == 6) break;


} /* End While loop */

} /*  End Choose Scenario      */



 /*************************************************************
 *
 *   Name:   Set_Current_Instances()
 *   Purpose:      Selects instances to print
 *   PreCond:      No instances for prn_frame
 *          variable chosen.  prn_frame = 1,2,3
 *
 *************************************************************/

set_current_instances() {

if (prn_frame == 3)  robot = CLEMENTINE;


if (prn_frame == 4)  robot = CLEMENTINE;


if (prn_frame == 5 ) robot = CLEMENTINE;


if (prn_frame == 6)  robot = GEORGE;
```

```
if (prn_frame == 7)  robot = GEORGE;

if (prn_frame == 8)  robot = GEORGE;

}   /*      End Set_Current_instances() */


prn(){

/***********************************************
 *                                             *
 *          Prints the Environment Frame   *
 *                                             *
 ***********************************************/

if (prn_frame == 1) {

printf("\n************************************************************");

printf("\n*****   Environment Frame Contents *****\n");

   for (i = 1; i <= NUM_ENVIRONMENTS; i++) {


      printf("Name:
");printf("%s\n",deter_envir(environment_list[i].name));
      printf("Width:          ");printf("%f",environment_list[i].width
);printf(" feet\n");
      printf("Height:          ");printf("%f",environment_list[i].height
);printf(" feet\n");
      printf("Length:
");printf("%f",environment_list[i].length);printf(" feet\n");
      printf("Light Intensity:
");printf("%s\n",deter_light(environment_list[i].light_intensity));
      printf("Temperature:
");printf("%f",environment_list[i].temperature);printf(" farenheit\n");
      printf("Current Scene:
");printf("%s\n",deter_scene(environment_list[i].curr_scene));

      printf("\n");


printf("\n*** Scenes for Environment ***\n");

          for (j = 1; j <= NUM_SCENES; j++) {

              printf("%s\n",deter_scene(environment_list[i].has_scene[j].n
ame));
          }


printf("\n****   Tasks for  Environment ***\n");

          for (j= 1; j<= NUM_TASKS; j++){

               if( environment_list[i].assoc_task[j]== TRUE )
               printf("%s\n",(deter_task(j)));
```

```
          }

    } /* i */
}  /* prn_frame = 1 */

/*****************************************************
 *                                                   *
 *          Prints the Scene Frame   Contents    *
 *                                                   *
 *****************************************************/

if (prn_frame == 2 ) {

     printf("\n************************************************************");
     printf("\n*****   Scene Frame Contents *****\n");
        for( i = 1 ; i <= NUM_ENVIRONMENTS; i++) {
             for (j =    1; j <= NUM_SCENES;      j++){
 printf("\nScene              %s",deter_scene(j));
 printf("\nEnvironment:
%s",deter_envir(environment_list[i].has_scene[j].part_of));
  /* for each scene, update the current object list */
 update_object(i,j);
/* Print the objects that correspond to the scene */
   printf("\n\n        ***  Object in this Scene  *** \n");

       for (k =    1; k <= NUM_OBJECTS;     k++){
      if (environment_list[i].has_scene[j].curr_object[k] == TRUE) {
 printf("\n        Object:
%s",deter_object(environment_list[i].object_list[k].name));
 printf("\n        Color:
%s",deter_color(environment_list[i].object_list[k].color));
 printf("\n        Temperature:
%s\n\n",deter_temp(environment_list[i].object_list[k].temperature));
       }  /* k */




             } /* j */
        } /* i */
} /* prn = 2 */
```

```
/*********************************************
 *                                           *
 *   Prints the contents of the Robot Frame  *
 *                                           *
 *********************************************/

if (prn_frame == 3 || prn_frame == 6) {

i = robot;


printf("\n\n************************************************");
printf("\n*****    Robot Frame Contents    *****\n\n");


for (i = robot; i <= robot; i++) {


   printf("\n\n");
   printf("Robot Name:              %s\n",deter_robot(robot_list[i].name));

   printf("Environment Name:
%s\n",deter_envir(robot_list[i].operates_in));
   printf("Robot Task:
%s\n",deter_task(robot_list[i].given_task));
   printf("Failure type:            %s\n",
deter_failure(robot_list[i].failure_type));
   printf("Second Attempt Failure: %s\n\n",
deter_failure(robot_list[i].second_attempt_failure_type));

printf(" ***  Current Sensor List Status: ***  \n ");

        for (j = 1; j<= NUM_SENSORS; j++) {

             /* If the robot does not have a given sensor then don't print
*/

             if( robot_list[i].curr_sensors_status[j] != NOT_AVAIL) {

                printf("\nSensor: %s",deter_sensor(j));

                printf("\nStatus:
%s\n",deter_status(robot_list[i].curr_sensors_status[j]));

            } /* not avail */

        } /* j */



} /* i */

} /* end prn_frame = 3/6 */


/*********************************************
 *                                           *
 *    Prints the contents of the Sensor Frame  *
 *                                           *
 *********************************************/
```

```
if (prn_frame == 4  || prn_frame == 7) {

i = robot;


printf("\n\n*********************************************************");
printf("\n*****   Sensor Frame Contents  *****\n\n");

for( i = robot; i <= robot; i++) {

      for( j = BW; j<= NUM_SENSORS; j++){


if (robot_list[i].curr_sensors_status[j] != NOT_AVAIL) {

          printf("Sensor:
");printf("%s\n",deter_sensor(robot_list[i].has_sensor[j].name));
          printf("Robot:
");printf("%s\n",deter_robot(robot_list[i].has_sensor[j].part_of));


         printf("Usage:
%s\n",deter_usage(robot_list[i].has_sensor[j].usage));
         printf("Horz_Fov:
%f",robot_list[i].has_sensor[j].horz_fov);printf(" degrees\n");
         printf("Vert_Fov:
%f",robot_list[i].has_sensor[j].vert_fov);printf(" degrees\n");
         printf("Data Type:
%s\n\n\n",deter_data_type(robot_list[i].has_sensor[j].data_type));

       } /* j */
 }
printf("\n\n*****  Sensor Beliefs for Active and Suspect Sensors
*****\n\n");

printf("            ( Support, Against and Don't Know) \n\n");

        for (j = 1; j <= NUM_SENSORS; j++) {

            if (robot_list[i].curr_sensors_status[j] == SUSPECT) {

                  printf("**");printf(" %s\n",deter_sensor(j));
                  printf("_____________________________\n");
                  printf(" Current Belief: %s");


                  for (k = SUPPORT; k <= DONT_KNOW; k++) {

                        if (k == SUPPORT)
                        printf("
%f",robot_list[i].has_sensor[j].curr_evidence[k]);

                       if (k > SUPPORT) {
                       printf("\n");
                       printf("
%f",robot_list[i].has_sensor[j].curr_evidence[k]);
                       }

                       if (k == DONT_KNOW) printf("\n\n");

                  } /* Current Beliefs */


                  /* Print the previous beliefs for the suspect sensor */
```

```
                    printf(" Previous Belief: %s");

                    for (k = SUPPORT; k <= DONT_KNOW; k++) {
                          if (k == SUPPORT)
printf("%f",robot_list[i].has_sensor[j].prev_evidence[k]);
                          if (k > SUPPORT){
                          printf("\n");
                           printf("
%f",robot_list[i].has_sensor[j].prev_evidence[k]);
                          }
                          if (k == DONT_KNOW) printf("\n\n");
                    }     /* Previous  Beliefs */

             } /* if suspect */

             else if (robot_list[i].curr_sensors_status[j] == ACTIVE ) {

                    printf("\n %s\n",deter_sensor(j));
                    printf("_____\n");
                    /* Print the current beliefs */

                     printf(" Current Belief: %s");

                          for (k = SUPPORT; k <= DONT_KNOW; k++) {
                              if (k == SUPPORT)
                                printf("
%f",robot_list[i].has_sensor[j].curr_evidence[k]);

                                if (k > SUPPORT) {
                                     printf("\n");
                                 printf("
%f",robot_list[i].has_sensor[j].curr_evidence[k]);
                                 }

                                if (k == DONT_KNOW) printf("\n\n");
                           }    /* Beliefs */


                       printf(" Previous Belief: %s");

                          for (k = SUPPORT; k <= DONT_KNOW; k++) {
                              if (k == SUPPORT)

printf("%f",robot_list[i].has_sensor[j].prev_evidence[k]);

                                if (k > SUPPORT) {
                                     printf("\n");
                                 printf("
%f",robot_list[i].has_sensor[j].prev_evidence[k]);
                                 }

                                if (k == DONT_KNOW) printf("\n\n");

                          }     /* Previus Beliefs */



             }
```

```
        }   /* j */

/*robot_list[i].second_attempt_failure_type = HIGH_CONFLICT;*/
if(robot_list[i].second_attempt_failure_type != NO_FAILURE) {
printf("\n\n***** -  SECOND RECOVERY ATTEMPT -  *****\n\n");
printf("       (Support, Against, and Don't Know) \n\n");

        for (j = 1; j <= NUM_SENSORS; j++) {
            if (robot_list[i].curr_sensors_status[j] == SUSPECT) {
                printf("**");printf(" %s",deter_sensor(j));
                printf("\n_____\n");
                 printf(" Second Attempt: %s");


                for (k = SUPPORT; k <= DONT_KNOW; k++) {
                       if (k == SUPPORT){
                       printf("
%f",robot_list[i].has_sensor[j].second_attempt_evidence[k]);
                      }
                      if (k > SUPPORT) {
                      printf("\n");
                      printf("
%f",robot_list[i].has_sensor[j].second_attempt_evidence[k]);
                      }
                      if (k == DONT_KNOW) printf("\n\n");
                } /* Second Attempt Beliefs */


            } /* if suspect */
            else if (robot_list[i].curr_sensors_status[j] == ACTIVE ) {
                printf("\n %s\n",deter_sensor(j));
                printf("_____\n");
                /* Print the Second attempt beliefs */
                      printf(" Second Attempt: %s");
                       for (k = SUPPORT; k <= DONT_KNOW; k++) {
                           if (k == SUPPORT)
                             printf("
%f",robot_list[i].has_sensor[j].second_attempt_evidence[k]);
                             if (k > SUPPORT) {
                                  printf("\n");
                             printf("
%f",robot_list[i].has_sensor[j].second_attempt_evidence[k]);
                             }
                             if (k == DONT_KNOW) printf("\n\n");
                       }   /* Second Attempt Beliefs */
```

```
            }
          } /* end if second attempt */

  } /* j */

      printf("\n\n *** List of Availible Enhancement for Sensors  ***
\n");
      for( j = BW; j<= NUM_SENSORS; j++){

         for (k = 1; k<= MAX_ENHANCEMENTS; k++) {

         /* If the robot does not have a given sensor, then don't print
         the enhancement */

         if( robot_list[i].curr_sensors_status[j] !=    NOT_AVAIL) {

            /* if there is no enhancement then dont print the name */

            if (robot_list[i].has_sensor[j].e_list[j].enhancement[k].name
!= NO_ENHANCEMENT) {

            printf("\n %s:",deter_sensor(j)); printf("          %s",
deter_enhancement(robot_list[i].has_sensor[j].e_list[j].enhancement[k].name
));

             }
          }
/*************************************************************/




      } /* k */
 }/* j */
 /* } */
}   /* i */


} /* prn_frame = 4/7 */

/*********************************************
 *                                           *
 *          Prints the Data Frames           *
 *                                           *
 *********************************************/


if (prn_frame == 5 || prn_frame == 8) {

i = robot;

printf("\n\n*********************************************************");
printf("\n*****   Data Frame Contents (Listed by Sensor) *****\n\n");


for( i = robot; i<= robot; i++){
       for (k = 1; k<= NUM_SCENES; k++){
      for (j = 1; j<= NUM_SENSORS; j++){
```

```
/* If no data for the current scene print a message */
if  (robot_list[i].has_sensor[j].provides[k].scene == NO_SCENE) {


  /*  printf("No %s sensor data for the
%s\n",deter_sensor(j),deter_scene(k)); */
}
else {

printf("\nSensor:
%s",deter_sensor(robot_list[i].has_sensor[j].provides[k].provided_by));
printf("\nScene:
%s",deter_scene(robot_list[i].has_sensor[j].provides[k].scene));
printf("\nRobot:                  %s",deter_robot(i));
printf("\nData_Type:
%s",deter_data_type(robot_list[i].has_sensor[j].provides[k].data_type));
printf("\nHorz Size:
%d",robot_list[i].has_sensor[j].provides[k].horz_size);printf(" bits");
printf("\nVert Size:
%d",robot_list[i].has_sensor[j].provides[k].vert_size);printf(" bits");
printf("\nDepth:
%d",robot_list[i].has_sensor[j].provides[k].depth);printf(" bits\n\n\n");

}
}     /* i */

}   /* j */

}   /* k */

} /* prn_frame = 5/8 */


} /* end prn() */
```

APPENDIX G

This appendix contains code for the string.c routine.

```
/*************************************************************************
 *
 *      Name:         string.c
 *      Purpose:    Determines the string values for the define
 *      Date:           March 16, 1994
 *
 *************************************************************************/

#include <stdio.h>
#include "zframes.h"
#include "zkbase.h"

void invalid() ;


char *deter_robot(integer)
int integer;

{
switch (integer) {
    case 0:
      return("No Robot");
      break;
    case 1:
      return("Clementine");
      break;
    case 2:
      return("George");
      break;
    default:
      invalid();
    }
}

char *deter_envir(integer)
int integer;
{
switch (integer) {
   case 0:
      return("No Environment");
      break;
   case 1:
      return("Lab");
      break;
    case 2:
      return("Warehouse");
      break;
    default:
```

```
        invalid();
    }
}

char *deter_temp(integer)
int integer;
{
switch (integer) {
    case 0:
      return("No Temperature");
      break;
    case 1:
      return("Below Ambient");
      break;
    case 2:
      return("Ambient");
      break;
     case 3:
      return("Above Ambient");
      break;
    default:
      invalid();
    }
}


char *deter_light(integer)
int integer;
{
switch (integer) {
    case 0:
      return("No Intensity");
      break;
    case 1:
      return("On");
      break;
    case 2:
      return("Off");
      break;
     case 3:
      return("Dim");
      break;
    default:
      invalid();
    }
}

char *deter_scene(integer)
int integer;
{
switch (integer) {
    case 0:
      return("No scene");
      break;
    case 1:
      return("Drill Press Scene");
      break;
    case 2:
      return("VCR/Monitor Scene");
      break;
     case 3:
      return("Door Scene");
      break;
    case 4:
      return("Student Desk Scene");
```

```
        break;
      default:
        invalid();
      }
}

char *deter_object(integer)
int integer;
{
switch (integer) {
    case 0:
      return("No Object");
      break;
    case 1:
      return("VCR");
      break;
    case 2:
      return("Monitor");
      break;
    case 3:
      return("Cabinet");
      break;
    case 4:
      return("Drill Press");
      break;
   case 5:
      return("Desk");
      break;
   case 6:
      return("Chair");
      break;
  case 7:
      return("Door");
      break;
  case 8:
      return("Fire Extinguisher");
      break;
  default:
      invalid();

      }

}

char *deter_color(integer)
int integer;
{

switch (integer) {
    case 0:
      return("No Color");
      break;
    case 1:
      return("Black");
      break;
    case 2:
      return("White");
      break;
     case 3:
      return("Red");
      break;
    case 4:
      return("Orange");
      break;
    case 5:
```

```
        return("Yellow");
        break;
      case 6:
        return("Green");
        break;
     case 7:
        return("Blue");
        break;
     case 8:
        return("Purple");
        break;
      case 9:
        return("Grey");
        break;
      default:
        invalid();
     }
}



char *deter_location(integer)
int integer;
{

 switch (integer) {
    case 0:
      return("No Location");
      break;
    case 1:
      return("On Ground");
      break;
    case 2:
      return("Elevated");
      break;
    case 3:
      return("Beside");
      break;
    case 4:
      return("In Front");
      break;
    case 5:
      return("Behind");
      break;
    default:
      invalid();
    }
}


char *deter_task(integer)
int integer;
{

switch (integer) {
   case 0:
      return("No Task");
      break;
    case 1:
      return("Security Guard");
      break;

    case 2:
      return("Monitor");
```

```
        break;
      default:
        invalid();
      }
}

char *deter_sensor(integer)
int integer;

{

 switch (integer) {

    case 0:
      return("No Sensor");
      break;
    case 1:
      return("Black & White");
      break;
    case 2:
      return("Color");
      break;
    case 3:
      return("Infrared");
      break;
    case 4:
      return("Ultrasonics");
      break;
    case 5:
      return("Ultraviolet");
      break;
    default:
      invalid();
    }
}

char *deter_enhancement(integer)
int integer;

{

 switch (integer) {

    case 0:
      return("No Enhancement");
      break;
    case 1:
      return("bw enhance");
      break;
    case 2:
      return("color enhance");
      break;
    case 3:
      return("false color");
      break;
    case 4:
      return("occupancy grid");
      break;
    case 5:
      return("polar plot");
      break;
```

```
   case 6:
      return("uv enhance");
      break;
    default:
      invalid();
    }
}



char *deter_failure(integer)
int integer;
{

 switch (integer) {
    case 0:
      return("No Failure");
      break;
    case 1:
      return("Below Minimum");
      break;
    case 2:
      return("Missing Evidence");
      break;
    case 3:
      return("High Conflict");
    case 4:
      return("Highly Uncertain");
      break;
  default:
      invalid();
    }
}


char *deter_usage(integer)
int integer;
{


 switch (integer) {
    case 0:
      return("No Usage");
      break;
    case 1:
      return("Detect Visible Light");
      break;
    case 2:
      return("Detect Range");
      break;
    case 3:
      return("Detect Heat");
      break;
    case 4:
      return("Detect Environmental Change ");
      break;
  default:
      invalid();
    }
}
```

```
char *deter_data_type(integer)
int integer;

{


 switch (integer) {
    case 0:
      return("No Data");
      break;
    case 1:
      return("Image");
      break;
    case 2:
      return("Numeric");
      break;
  default:
      invalid();
    }
}

char *deter_status(integer)
int integer;
{


 switch (integer) {

    case 0:
      return("NOT_AVAIL");
      break;
    case 2:
      return("Inactive");
      break;
    case 1:
      return("Active");
      break;
    case 3:
      return("Suspect");
      break;
    default:
      invalid();
    }
}

char *deter_obj_name(integer)
int integer;
{

 switch (integer) {
    case 0:
      return("No Object");
      break;
     case 1:
      return("VCR");
      break;
    case 2:
      return("Monitor");
      break;
     case 3:
      return("Cabinet");
      break;
    case 4:
      return("Drill Press");
      break;
```

```
      case 5:
        return("Desk");
        break;
      case 6:
        return("Chair");
        break;
      case 7:
        return("Door");
        break;
      case 8:
        return("Fire Extiguisher");
        break;
      default:
        invalid();
    }
}

void invalid() {
      printf("\n Invalid Number");
      exit(1);
}
```

APPENDIX H

This appendix contains code for the misc.c routine.

```
/*****************************************************************************

      Name:         misc.c
      Purpose:      March 25, 1994
      Comments:     Contains misc routines to allocate nodes and insert
                    nodes

*****************************************************************************

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "zframes.h"
#include "zkbase.h"
#include "zglobal.h"


/*********************************************************** ******

  Repair or Replace the Current Sensing Plan.

  < This seems too simple to have as a function by itself but
  I am not sure what else will be  needed in order to get the
  information from teleSFX to update the knowledge base for a
  repaired or replaced plan so, I will leave it like this for
  now >.
*************************************************************/


update_current_sensors() {

int robot = CLEMENTINE;

int j;


/* Simulating the retreival of activation conditions from teleSFX */

    activation_cond_avail_sensors[0] =      TRUE;
    activation_cond_avail_sensors[1] =      FALSE;
    activation_cond_avail_sensors[2] =      TRUE;
    activation_cond_avail_sensors[3] =      FALSE;
    activation_cond_avail_sensors[4] =      TRUE;
```

```
/* Update the sensor status in the knowledge base */

        for( j = 1; j <= NUM_SENSORS; j++){


       robot_list[robot].curr_sensors_status[j] =
       activation_cond_avail_sensors[j - 1];


    }

}


/*****************************************************************

    When a New plan is obtained, the scene and the sensors may
    change.  Therefore both the status of the availible sensors
    and the current scene must be updated.

*****************************************************************/

update_scene() {


int i = LAB;


/* Simulates the retreival of the current scene from teleSFX */

            current_scene = STUDENT_DESK_SCENE;


/* Updates the current scene in the knowledege base */

            environment_list[i].curr_scene = current_scene;


}


new_plan() {

        update_current_sensors();
        update_scene();

}

get_failure() {


}


 /*********************************************************

Defines the enhancement functions for the data obtained from
black and white, color, IR , Ultrasonics and Ultraviolet sensors.
These are "empty" functions because they are defined elsewhere.
I have simply provided a store for them in
the knowledge base so they may be added to the
enhacement list under the appropriate sensor heading.

Additionally,
```

There are 3 dummy function below: bw_enhance(), color_enhance() and uv_enhance(). The names don't stand for real functions but been included to show how a list of enhancement may associated with each sensor.

Generally, the enhancement is based on the type of data provided by a sensor. However, bw_enhance() and color_enhance() are used as examples to provide a store for enhancement procedures that may be unique to the bw and color sensor even though the bw and color sensors provide the same type of data.

**********/

```
/* BW enhancement */

bw_enhance() {

}

/* Color Enhancement */

color_enhance() {

}

/*  IR enhancement */

false_color() {

}

/* Ultrasonics enhancements */

occupancy_grid() {

}

polar_plot() {

}

/* Ultraviolet enhancement */

uv_enhance() {

}
```

APPENDIX I

This appedix contains global variables.

```
#ifndef ZGLOBAL_H
#define ZGLOBAL_H 1
/* This  file contains the global varibles used in the televia kbase */

#include "zframes.h"

#define NIL_FUNCTION            ((char*) 0)
#define NIL_DATA        ((DATA_P) 0)
#define NIL_SCENE       ((SCENE_P) 0)


/*
DATA_P    data_ptr;
SCENE_P  scene_ptr;
  */

/* variable hold dynamic data from teleSFX */

int     activation_cond_avail_sensors[NUM_SENSORS + 1];
int     current_scene;

#endif
```

APPENDIX J

This appendix contains code for the test.c routine.

```
/*************************************************************************
    Name:           test.c
    Date:           March 30, 1994
    Purpose:        Performs the knowledge base initializations,
                    instantiations and print routines
    Comments:       Calls init_frames and prn_routine

*************************************************************************/


#include <string.h>
#include <stdio.h>
#include "zkbase.h"
#include "zframes.h"


main() {

/* Initializes all frames in the knowledge base */

init_frames();


/* Instantiates all frames in the knowledge base */

put_robot_data();
put_sensor_data();
put_data();
put_environment_data();
put_scene_data();
put_object_data();


/* Updates Knowledge base for Simulated failure */

  /*
choose_scenario();
*/

/* Updates Knowledge Base to maintain consistency when Simulated
   Reconfiguration or Backup Plan takes place */


/*update_kbase();*/
```

```
/*  Prints the frame contents */
prn_routine();

}   /*       End Main     */




/***********************************************
 *                                             *
 *     Initialize contents of the frames       *
 *                                             *
 ***********************************************/

init_frames() {

init_robot_frame();
init_sensor_frame();
init_data_frame();
init_environment_frame();
init_scene_frame();
init_object_frame();

}   /*       End Init_Frames()    */


 /***********************************************
 *                                             *
 *     Prints the contents of the frames       *
 *                                             *
 ***********************************************/


prn_routine(){

     for(;;)       {

        choose_frame();
        set_current_instances();
        prn();

     }

} /*  End Print Routine  */
```

WORKS CITED

Barr, A., and Feigenbaum, E. eds., 1982. The Handbook of Artificial Intelligence. Vol. I. California: William Kaufmann, Inc.

Cattell, R.G.G. 1991. Object Data Management Object Oriented and Extended Relational Database Systems. New York: Addison-Wesley Publishing Company.

Chavez, T. Greg, 1993. Exception Handling for Intelligent Sensor Fusion. Masters thesis, Colorado School of Mines.

Coiffet, P. and Gravez, P. 1991. Man-Robot Cooperation: Toward an Advanced Teleoperation Mode. Tzafestas, S.G, ed. Intelligent Robot Systems. New York: Marcel Dekker, Inc. 593-636.

Rich, E., and Knight, K. 1991. Artificial Intelligence. New York: McGraw-Hill, Inc.

Rogers, E. 1992. Visual Interaction: A Link Between Perception and Problem-Solving. Tech Report No. GIT-CC-92/59. Georgia Institute of Technology.

Murphy, R.R. 1992. An Architecture for Intelligent Robotic Sensor Fusion. Tech Report No. GIT-CC-92/59, College of Computing Georgia Institute of Technology.

Rogers, E and Murphy, R.R. 1994. Teleassistance for Semi-Autonomous Robots. Proc. of AIAA Conference in Intelligent Robots in Field, Factory, Service and Space, NASA Conference Publication 3251, 500-508.

Schenker, Paul, S. 1991. Intelligent Robots for Space Application Systems. Tzafestas, S.G. ed. Intelligent Robotics Systems. New York: Marcel Dekker, Inc. 545-591.

Waterman, D.A. 1986 A Guide to Expert Systems. New York: Addison Wesley Publishing Company.