September 1985

A Motion Verb Interface to A Task Animation System

Jeffrey Scott Gangel
University of Pennsylvania

Follow this and additional works at: http://repository.upenn.edu/cis_reports

Recommended Citation
http://repository.upenn.edu/cis_reports/729


This paper is posted at Scholarly Commons. http://repository.upenn.edu/cis_reports/729
For more information, please contact repository@pobox.upenn.edu.
A Motion Verb Interface to A Task Animation System

Abstract
TEMUS, an interactive software system designed at the University of Pennsylvania, provides a graphical simulation of human movements within a workstation environment [4, 5, 17]. The existing user interfaces to TEMUS are based on menu selection, alphanumeric commands or data input and graphical interaction. The latter are especially suited to the types of interaction necessary for creating workstation objects and performing body positioning in TEMUS.

Looking toward the future application of TEMUS, however, we see that the long-term goals of human motion modeling will include the analysis and visualization of extensive tasks in space involving one or more individuals working in concert over a period of time. In this context, the TEMUS body positioning capability, though extremely useful in creating and validating a small number of particular body positions, will become somewhat tedious to use. The macro facility helps somewhat, since frequently used positions may be easily applied by executing a stored macro. The difference between body positioning and task execution, though subtle, is important. In the case of task execution, the important information at the user’s level is what actions are to be performed rather than how the actions are performed. Viewed slightly differently, the what is constant over a set of individuals though the how may vary.

Comments

This technical report is available at ScholarlyCommons: http://repository.upenn.edu/cis_reports/729
A Motion Verb Interface
To A Task Animation System

Jeffrey Scott Gangel
MS-CIS-85-35

Department of Computer and Information Science
Moore School/D2
University of Pennsylvania
Philadelphia, PA 19104

September 1985

*Acknowledgement: NASA grant NAS9-16634 and NAS9-17239.
To my family.
ACKNOWLEDGEMENTS

I would like to thank Dr. Norman Badler, for his guidance, support, and relatively free reign in my thesis.

Special thanks to Dr. Tim Finin for his BUP system. It help me get out of a syntactical hole.

Thanks to all the members of the TEMPUS staff, with special thanks to Paul Fishwick. Without his thoughts on the subject, I would have had many more problems.

This work was supported by NASA Grants NAS9-16634 and NAS9-17239, NSF CER Grant MCS-82-19196, and ARO Grant DAAG29-84-K-0061.
# Table of Contents

1. Introduction  
   1.1. Overview  

2. Processing NASA Checklists  
   2.1. Task Representation Requirements  
   2.2. Checklist Procedures  
   2.3. Task Representation  
   2.4. Implementation of the FDF Parser

3. Natural Language Parsing  
   3.1. Introduction  
   3.2. Top Down vs. Bottom Up Parsing

4. Representational Structures  
   4.1. Case Frame  
   4.2. Incomplete Case Frames  
   4.3. A Particular Case  
   4.4. The Verb's Part In Determining A Complete Case Frame

5. The Object Knowledge Representation  
   5.1. The Isa-Parent Structure  
   5.2. Physical Object Structures  
      5.2.1. The Control Structure  
      5.2.2. The Indicator Structure  
      5.2.3. Other Inanimate Structures  
      5.2.4. The Agent Structure  
   5.3. Conclusions

6. Primitive Verbs

7. Code Analysis  
   7.1. Introduction  
   7.2. A Test Panel  
   7.3. Syntax Rules  
      7.3.1. Limitations  
      7.3.2. Features  
         7.3.2.1. Conceptual Categories  
      7.3.3. Agent Variations  
      7.3.4. Performance  
   7.4. Resolve Ambiguities
List of Tables

Table 4-1: Schank's Primitive Action Verbs 19
Table 4-2: Case Frame Roles 21
Table 4-3: Verb Case Frames 24
Table 5-1: The Object Knowledge Base 27
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Organization of the User Interface</td>
<td>3</td>
</tr>
<tr>
<td>2-1</td>
<td>Switch Operation Sequence Structure</td>
<td>8</td>
</tr>
<tr>
<td>2-2</td>
<td>Switch Operations</td>
<td>10</td>
</tr>
<tr>
<td>2-3</td>
<td>Example of Checklist Prior to Parsing</td>
<td>12</td>
</tr>
<tr>
<td>2-4</td>
<td>Expanded Checklist of Figure 2-3.</td>
<td>13</td>
</tr>
<tr>
<td>2-5</td>
<td>Representation of Figure 2-3.</td>
<td>13</td>
</tr>
<tr>
<td>7-1</td>
<td>Panel 1</td>
<td>41</td>
</tr>
<tr>
<td>7-2</td>
<td>Lisp Code For The Verb TURN</td>
<td>49</td>
</tr>
</tbody>
</table>
CHAPTER I
Introduction

TEMPUS, an interactive software system designed at the University of Pennsylvania, provides a graphical simulation of human movements within a workstation environment [4, 5, 17]. The existing user interfaces to TEMPUS are based on menu selection, alphanumeric commands or data input and graphical interaction. The latter are especially suited to the types of interaction necessary for creating workstation objects and performing body positioning in TEMPUS.

Looking toward the future application of TEMPUS, however, we see that the long-term goals of human motion modeling will include the analysis and visualization of extensive tasks in space involving one or more individuals working in concert over a period of time. In this context, the TEMPUS body positioning capability, though extremely useful in creating and validating a small number of particular body positions, will become somewhat tedious to use. The macro facility helps somewhat, since frequently used positions may be easily applied by executing a stored macro. The difference between body positioning and task execution, though subtle, is important. In the case of task execution, the important information at the user's level is what actions are to be performed rather than how the actions are performed. Viewed slightly differently, the what is constant over a set of individuals though the how may vary.

To meet these future expected needs of human task description we began a study of more language-oriented interfaces to TEMPUS. Language can describe how to perform an action, but it is much more efficient and effective in describing what is to be accomplished. Human intelligence, innate movement patterns,
learned actions, and general experience make possible human task performance. TEMPUS attempts to simulate at least some of these, such as the task reach. A language-oriented interface would permit the TEMPUS user to describe the steps and goals of a task in a form much like, or even identical to, the form interpretable by another human being. There is no doubt that this is a modern area of extensive research covered even more broadly by the field of Artificial Intelligence. Thus our purpose will not be to attempt the analysis of arbitrary human communication; rather, we shall concentrate on those areas of physical task execution that are fundamental to the NASA environment.

A problem with many interactive systems is the manner in which the input must be formatted. Often, a user would like to communicate with a system in his or her own language, for example English. In practice, though, communication with a computer through arbitrary (but humanly understandable) English is not feasible. Thus we frequently resort to more limited syntactic, semantic, and lexical domains. Such an interface appears to understand a restricted form of English utterances that nonetheless transmits most of the required information.

In TEMPUS the interface must allow the user to command the (simulated) agents to perform a number of tasks. If the interface is treated as a participant in a conversation (so to speak), then everything entered should be understood by the interface and conveyed to the system. In TEMPUS, however, the interface must do more. The simulated agents have no action intelligence and require a specific type of statement to allow them to perform in the manner that was implied. By allowing the interface to have an insight into the context of a specific environment, the message sent can be conveyed with all the information needed by the simulator. As a result, the interface is an intermediary between the TEMPUS user and its available simulation capabilities, with a relatively complete insulation of the user from the details of the required simulation structure.

HIRES [15], a Hierarchical REasoning System, is a simulation system being
designed to reason about processes involving complex actions using a variety of objects. It views a simulation as a hierarchy of abstract processes, objects, times, and reports. Once finished, HIRES will complete the connection between the user's commands and TEMPUS’ graphical simulation.

1.1. Overview

![Diagram]

Figure 1-1: Organization of the User Interface

We begin by describing some of the characteristics of a language-oriented
user interface (see Figure 1-1), examine some of the representational problems that arise, and consider how they might be solved. Of the two common NASA task specification techniques that exist, operational callouts and commands, this thesis mainly considers the second, commands. However, a review, analysis, and representation of both are shown. Examples are used extensively to illustrate the kinds of task instructions and how they may be parsed and understood by the computer. Finally we show how the structures representing the language input is provided to TEMPUS and its associated movement simulation system, HIRES [15].

As diagrammed in Figure 1-1, the language interface to TEMPUS consists of several components. There are two basic input formats: user input and flight data files, though both result in a common output representation message for the simulation. Chapter 2 contains a general presentation of NASA checklist procedures, used to define tasks in the Space Shuttle. This is only a general overview. As seen in Figure 1-1, it has not been completed. The remaining chapters pertain mainly to a user's English command input. Chapter 3 presents an overview of natural language parsing, the type of input used for the rest of the paper. Chapter 4 presents the type of structure used to define an input statement. Chapter 5 defines the object knowledge base, data about each object in the environment. As seen in Figure 1-1, it is needed in many cases. Chapter 6 defines the primitive verbs used in the system (see Figure 1-1). Chapter 7 presents an analysis of the code written. It presents the methods used to follow Figure 1-1 and any limitations that may occur. Chapter 8 points out many of the areas that still must be researched to provide a complete interface.
CHAPTER II
Processing NASA Checklists

To improve the user interface to the TEMPUS body positioning system, the use of existing NASA checklist procedures was investigated. This chapter discusses a computer-oriented representation for the kind of information contained in NASA checklist procedures. A representational structure that will accurately represent these procedures will consist of action verb case frames compatible with those in the interactive user interface. Many examples are given to aid in describing how the checklist procedures can be parsed into the chosen task representation. The parser will interpret as much information as possible from the checklist procedures, and will relay this information in detail to a simulation system [15] which drives the TEMPUS body motion primitives. This chapter will provide an abbreviated description of NASA checklist procedures. A more complete explanation of NASA checklist procedures is in a NASA report [6].

2.1. Task Representation Requirements

NASA checklist procedures are a complicated system of abbreviations and artificial syntax which is nonetheless intended to be human-readable. They represent predetermined actions to be carried out by the crewmembers from the time the environment (shuttle) is entered until it is exited. To interface between the NASA checklists and the TEMPUS graphic simulation an intermediate structure, called a task representation, is needed.

The task specification will be compiled first by translating the NASA checklist procedures into a task representation. The checklist procedures are in
the form of a terse NASA documents entitled the Flight Data File (FDF). The FDF describes the tasks to be carried out by the crewmembers of the Space Shuttle. The task representation is passed to a simulation which decides how each action will be carried out.

Several problems are involved with designing the task representation. Of major concern is the representation of temporal relationships between actions. It is essential that each action be done at exactly the time and order designated. NASA documents spend a large amount of effort describing and representing the temporal relationships of the actions to be performed. Allen [2] discusses various temporal relationships of actions and how to represent them. Three problems that he feels occur in nearly all existing (computationally-based) models of action are addressed:

1. Actions that involve non-activity.
2. Actions that are not easily decomposed into sub-actions.
3. Actions that occur simultaneously and may interact with each other.

The third point is of particular interest because of the importance of accurately representing sequences of actions, parallel actions, and their interactions. Allen concludes that simultaneous actions can be described directly if the temporal aspects are separated from the causal aspects of a plan and from that it enables us to describe the interactions as well. He feels that problems still remain in building a system that can reason about such interactions while problem solving, but the representation appears suitable for our simulation needs.

More generally, we must address the problem of representing the entire procedure or event (see Chapter 4). The checklists must be represented in a viable structure, containing all of its pertinent information.

Another major problem in translating the checklist procedures into the task
representation structure is understanding the parsing NASA did when initially preparing the Flight Data File documents. Much time and effort went into making FDF's quick and easy for the crewmembers to read. But without proper training and supplementary documents containing abbreviations and standards, they are almost impossible to decipher. In fact, the checklists are often considered cue cards rather than being true instructional devices. In order to design the task representation accurately, a complete understanding of the Flight Data File is essential. (See Figure 2-3 for a sample cue card).

2.2. Checklist Procedures

There are two basic types of operations in the NASA checklist procedures: commands and operational callouts. Figure 2-1 shows how each type is broken down into, or formed from its components. Commands are similar to those used in an user, interactive session, with a implied format of extreme brevity. Operational callouts follow the strict format specified in Space Shuttle Flight Data File Preparation Standards [13].

The operational callout is used to describe functions that are associated with controls and displays. They are composed of steps, crewmember designation, location identification, procedure, and notes. Only the procedure is required at all times, specifying the object to be operated on and the action, the two separated by a dash. The other components are only included as required.

Consider the following examples:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>R2</td>
<td>1</td>
<td>HYD MAIN PUMP PRESS 1 - NORM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>- LD</td>
</tr>
<tr>
<td>C</td>
<td>C3</td>
<td>2</td>
<td>MAIN ENG LIMIT SHUT DN - AUTO</td>
</tr>
<tr>
<td>P</td>
<td>R2</td>
<td>3</td>
<td>HYD MAIN PUMP PRESS 3 - NORM</td>
</tr>
</tbody>
</table>

The far left column contains the crewmember designation. There are two types, general and specific. General crew designations, such as
are associated with procedures in which all crewmembers have received equal training and each is capable of performing the task. Specific crew designations, such as,

- C - Commander
- P - Pilot
- EV1 - EVA crewman 1 (extra-vehicular activity)
- MS - Mission Specialist
- PS - Payload Specialist

are used in cases where certain crewmembers receive specialized training in a given task. In the above example, the first two lines are to be performed by the pilot even though it is only specified in the first line. Only when a new crewmember is needed to perform an operation is its designation specified.

Location identification is used to specify the location of a particular
operation. Identification may be specific, such as a panel or stowage locker number, or general, such as a flight deck, middeck, or airlock. A location is usually specified for each step, unless there is no need to perform the operation in a particular location. The location is specified in a column to the left of the step number. One or more spaces are always left between the location designation and the step number or the first character of the procedure (when step number is not indicated). In the above example the second line contains no location identification. In such cases the convention is that the location (crewmember and object) remain the same from the previous operation. However, the information must be included for step 3 because of the intervening step 2 that has a different location (crewmember and object).

Panel locations are often combined to save space and to consolidate operations into single steps. For example,

\[
\begin{align*}
F_5 & \text{ ADI ATT - INTRL} \\
F_8 & \text{ ADI ATT - INTRL} \\
\end{align*}
\]

into

\[
\begin{align*}
F_6/F_8 & \text{ ADI ATT - INTRL,} \\
\end{align*}
\]

where a slash, /, indicates and. Both statements must be present in the task representation and must be expressed as separate callouts. Alternate panel locations may be indicated by using parentheses, ( and ). As in, \text{F6(F8) ADI ATT - INTRL}, they indicate that the step can be performed at either location.

The final portion of the operational callout is the procedural callout, consisting of object, dash, and action. In order to understand these statements the use of upper and lower case letters, spacing, symbols, and special identifiers must be studied. This style of writing was adopted to aid the crewmembers in quickly identifying and accomplishing tasks. Some examples of switch operations are shown in Figure 2-2. It should be noted that the type of switch, quantity, and clarifying comment are not always required. Valve operations appear the same as switch operations except its abbreviation, \text{vlv}, is used. Similar forms are used for other devices.
To accurately represent the operations as described on the cue cards, conventions used to conserve space and simplify operations must be accounted. One such method is combining switch callouts. Special attention must be given to properly identifying the number of switches being positioned and the particular switches desired from a given group. Series of callouts are condensed as in,

BOILER CNTLR 1 - ON
BOILER CNTLR 2 - ON
BOILER CNTLR 3 - ON

into

BOILER CNTLR (three) - ON

BOILER CNTLR 1 - ON
BOILER CNTLR 3 - ON

into

BOILER CNTLR 1,3 - ON

and

BOILER CNTLR 1 - ON
BOILER CNTLR 2 - ON

or

BOILER CNTLR 1 - ON
BOILER CNTLR 3 - ON

into

BOILER CNTLR 1(2,3) - ON.

The task representation must interpret these combined operations as multiple tasks and return the condensed version back to its original expanded form.

Another simplifying convention for operational callouts that must be expanded is the recreation of deleted leading redundant words, panel numbers, and row numbers. For example,
must be represented as:

\begin{verbatim}
08 LTG PNL R O V HD - O F F
 R INST - OFF
\end{verbatim}

Checklist operations often require clarification or identification of special conditions. Three different forms of information may appear in a procedure: general notes, cautions, and warnings. Although valuable to the crew members, it will not be used in the task representation. More likely they will be passed as text to appear as captions in the graphical simulation.

Checklist commands are used primarily for operations not associated with displays or controls and may be of two types: simple and conditional (see Figure 2-1). Their brief natural language format is similar to user interactive commands.

2.3. Task Representation

The \textit{parser} translates the checklist procedures into a target structure, the task representation. This structure is in turn passed to the simulation system and TEMPUS. The FDF task frame needs at least six relevant slots (compatible with case frame roles in Figure 4-2):

- \textit{NUMBER}: An ordered number for each command frame.
- \textit{AGENT}: The agent in the command given or implied in the callout. If no agent is given or implied, a blank will be passed. The simulator will assign the agent (depending on who is available, who can get there quickest and easiest, and what type of action is needed).
- \textit{OBJECT}: A specific object.
- \textit{ACTION}: The action or position desired as specified by the callout (in a case frame, the action would be the frame's verb).
- \textit{TEMPORALS}: Follows Allen's 13 temporal relationships.
• TIME: The amount of time to carry out a particular action.

To fill the action slot in an operational callout (not a command), the position desired for that particular switch or valve will be obtained as it appears in the callout.

BUS LOSS ACTION
MNC ALC3

R2  BLR CNTRL PWR/HTR 2 - B (1 - B on 102)
    3 - A
L1  FLASH EVAP CNTRL PRI A - OFF
    PRI B - ON
    TOP EVAP HTR NOZ R - A AUTO
A12 APU HTR GAS GEN/FUEL PUMP 2 - A AUTO
    3 - B AUTO
    TK/FU LINE/H2O SYS 2A, 3B (two) - AUTO

Figure 2-3: Example of Checklist Prior to Parsing

To further illustrate the FDF parsing technique Figure 2-3 shows an example cue card from an FDF Orbit Pocket Checklist. Missing from this cue card are the crew designation and step numbers. The step numbers are relatively unimportant since a number will be assigned to each event as it is translated. This example is rather straightforward as far as temporals are concerned; the operations will be performed in the order that they appear. As such, no temporals are involved in this example. The crewmember(s) chosen to perform the operations will depend upon the context in which this cue card is used.

Extrapolating all the hidden information in Figure 2-3 will lead to the expansion of the callout as shown in Figure 2-4. This expansion can then be parsed into the command frame suitable for input to the simulation. Figure 2-5 shows the parse in tabular form.

An additional condition for operational callouts is a check, (√). Preceding a
R2  BLR CNTRL PWR/HTR 2 - B (1 - B on 102)
R2  BLR CNTRL PWR/HTR 3 - A
L1  FLASH EVAP CNTRL PRI A - OFF
L1  FLASH EVAP CNTRL PRI B - ON
L1  TOP EVAP HTR NOZ R - A AUTO
A12 APU HTR GAS GEN/FUEL PUMP 2 - A AUTO
A12 APU HTR GAS GEN/FUEL PUMP 3 - B AUTO
A12 APU HTR TK/FU LINE/H20 SYS 2A - AUTO
A12 APU HTR TK/FU LINE/H20 SYS 3B - AUTO

Where the last two lines are the expanded form of:

A12 APU HTR TK/FU LINE/H20 SYS 2A,3B (two) - AUTO

**Figure 2-4:** Expanded Checklist of Figure 2-3.

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>AGENT</th>
<th>OBJECT</th>
<th>ACTION</th>
<th>TEMPORAL</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R2</td>
<td>BLR CNTRL PWR/HTR 2</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R2</td>
<td>BLR CNTRL PWR/HTR 3</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>L1</td>
<td>FLASH EVAP CNTRL PRI A</td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>L1</td>
<td>FLASH EVAP CNTRL PRI B</td>
<td>ON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>L1</td>
<td>TOP EVAP HTR NOZ R</td>
<td>A AUTO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A12</td>
<td>APU HTR GAS GEN/FUEL PUMP 2</td>
<td>A AUTO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>A12</td>
<td>APU HTR GAS GEN/FUEL PUMP 3</td>
<td>B AUTO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>A12</td>
<td>APU HTR TK/FL LINE/H20 SYS 2A</td>
<td>AUTO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A12</td>
<td>APU HTR TK/FL LINE/H20 SYS 3B</td>
<td>AUTO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2-5:** Representation of Figure 2-3.

procedural callout it indicates verify or check. If before a callout, an implied predicate is indicated. The agent (or simulator) is to check whether the object is in the state given in the callout. If it is in that state, nothing is done. Else, the operation must be performed to obtain that state. For example,

F4E √ CWDS DUMP ISOL VLV - CL

means, *Check whether the condensate tank dump ISOL valve is closed; if not, close it.* The object slot will contain the location identification and the valve title. The action slot will contain check CL.
2.4. Implementation of the FDF Parser

At this time, a complete, working parser for the operational callouts does not exist. A partial parser does exist [12], however further work must be done to complete and expand it to the point where the parser can accept all (or almost all) of the operational callouts that may exist now, or in the future. Its final output must be totally compatible with the rest of the interface system (to be explained) such that its output can eventually be sent to the simulation system (see Figure 1-1).
3.1. Introduction

A parser is a device which can be used to determine whether a sentence conforms to a language's syntax constraints, and often to formalize the input into a semantic representation structure. In this case, we will only consider parsers for a specific type of input, natural language. However, regardless of the type of input, parsers can be divided into two methods, top-down and bottom-up.

3.2. Top Down vs. Bottom Up Parsing

Top-down parsing begins at the top rule (often labeled with the symbol S) and attempts to follow it with other rules until it eventually leads to a complete account of all the constituents (grammatical units) of the sentence. For example, the sentence "Jeff turned the valve" would be top-down parsed in an order similar to:

```
S  -> NP VP
NP -> N
N  -> Jeff
VP -> V NP
V  -> turned
NP -> DET N
DET -> the
N  -> valve.
```

Bottom-up parsing begins with the constituents of the sentence and looks for rules whose right-hand side matches them. The left-hand side of each rule then forms new constituents which are again matched, and so on, until it becomes a
single left-hand side (the symbol S). Using the same example as above, it would be
bottom-up parsed in an order such as:

\[
\begin{align*}
N & \rightarrow \text{Jeff} \\
NP & \rightarrow N \\
V & \rightarrow \text{turned} \\
DET & \rightarrow \text{the} \\
N & \rightarrow \text{valve} \\
NP & \rightarrow \text{DET N} \\
VP & \rightarrow V \ NP \\
S & \rightarrow NP \ VP.
\end{align*}
\]

The result of either parser is:

\[
[S [NP \ [N \ Jeff]]] \\
\ [VP \ [V \ turned]] \\
\ [NP \ [\text{DET the}]] \\
\ [NP \ [\text{DET valve}]]]]
\]

(In this Lisp-like list notation it may be seen that the pairs of items in braces are
associated as an attribute-value pair: thus [N Jeff] implies Jeff is a N (noun);
[NP [N Jeff]] implies [N Jeff] is a NP (noun phrase); etc.)

Although the results are identical: the amount and type of processing done
by the parsers are very different; Top-down parsing is goal-directed processing:
the path is guided by the goals of the process. Bottom-up parsing is data-directed
processing, the path is guided by the given data. Although there exist advantages
and disadvantages to both, a bottom-up parser, BUP [14], was chosen it
happened to be available to us.

Augmentations to a parser allow the addition of certain features to the
grammar rules providing additional information. For example, Jeff is an animate
object and the valve is an inanimate object. Without these augmentations, the
sentence, *The valve turned Jeff* could be parsed by the given rules. If the verb
turn requires that the subject must be animate, only *Jeff turned the valve*
would be allowed. BUP allows such augmentations to be included in the rules.

---

1 adapted from Franz Lisp to Common Lisp by Gangel
Parsers also allow various output structures. The standard output is a *syntactic* structure of the sentence. Other structures contain semantic representations of the input data. In our case, BUP is being used to form a structure called case grammar frame.
We must now address the problem of representing the meaning of an entire concept. Beyond the syntactic structure of the concept, a deep structure, based on the application of the syntactical phrase structure rules and certain lexical semantic features plus semantic components applicable to the constituents, must be used to determine the true meaning of the concept.

4.1. Case Frame

Schank [19] proposed a conceptual dependency grammar (CD) for representing events. Using several basic primitives verbs for action verbs, all actions can be reduced to a sequence of primitive verbs based on the idea of case grammar. The constituents of the event are represented in case slots relevant to the primitive verbs.

His work has several weaknesses but the idea of a case frame, primitive verbs and relevant slots, is the backbone of our analysis of the action verb sentences presented as input. For our purposes, it is sufficient to understand that a case frame is a list of attributes (cases or slots) dependent on an action verb in which varying types of information (values) may be placed.

Case grammar argues that a set of cases can be used to define the deep structure (meaning, as opposed to the superficial sentence structure) of English sentences. Schank’s CD representation system is based on the conceptual relationships between objects and actions. As in all case grammar structures, the
meaning of a sentence is not dependent upon the form of the sentence. The sentences, "Jeff turned the valve" and "The valve was turned by Jeff," both have the same deep structure. Schank's CD extends the case grammar concept one step forward and reduces all verbs into a member of a set of limited primitive verbs (see Table 4-1). As a result, both of the above sentences would be reduced into the case frame:

```
EVENT
ACTOR: Jeff
ACTION: propel
OBJECT: the valve.
```

where ACTOR, ACTION, and OBJECT are filled with their constituents, Jeff, propel, and the valve.

<table>
<thead>
<tr>
<th>Primitive Verb</th>
<th>Meaning</th>
<th>Sentence</th>
<th>CD Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTRANS</td>
<td>physical transfer</td>
<td>Jeff moves to N.Y.</td>
<td>Jeff PTRANS N.Y.</td>
</tr>
<tr>
<td>ATRANS</td>
<td>abstract transfer</td>
<td>Jeff buys a book from the store.</td>
<td>Jeff ATRANS book</td>
</tr>
<tr>
<td>MTRANS</td>
<td>mental transfer</td>
<td>Jeff reads a book.</td>
<td>Jeff MTRANS book</td>
</tr>
<tr>
<td>PROPEL</td>
<td>apply physical force</td>
<td>Jeff throws a ball.</td>
<td>Jeff PROPEL ball</td>
</tr>
<tr>
<td>MBUILD</td>
<td>mental operation</td>
<td>Jeff understands the code.</td>
<td>Jeff MBUILD code</td>
</tr>
<tr>
<td></td>
<td>(old to new info)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTEND</td>
<td>focus attention</td>
<td>Jeff hears a song.</td>
<td>Jeff MBUILD song</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jeff ATTEND song</td>
<td></td>
</tr>
<tr>
<td>GRASP</td>
<td>get physical control</td>
<td>Jeff holds a pen.</td>
<td>Jeff GRASP pen</td>
</tr>
<tr>
<td></td>
<td>(with your hands)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>bodily movement</td>
<td>Jeff drinks coffee.</td>
<td>Jeff MOVE coffee</td>
</tr>
</tbody>
</table>

Table 4-1: Schank's Primitive Action Verbs

Only recently have efforts been made to construct a representation combining concepts and temporals. Besides the efforts cited in a NASA report
relevant work in this area was also done by Waltz [24]. Waltz addresses some of the shortcomings of Schank's CD representation system, one of which is capturing the similarity of meanings in various verbs rather than representing the nuances in meaning. In an environment, such as the Space Shuttle, the meaning of each command must be correctly interpreted or else a wrong action may be performed. Waltz feels that there exist no proper primitives in Schank's representation for many actions. His event shape diagrams deal with action combinations and result in the temporal framework of an event. Although his paper is short and just scratches the surface of this concept, it offers useful ideas in setting up the task representation. His event shape diagrams can be used to represent concurrent processes, causation and other temporal relations.2

4.2. Incomplete Case Frames

As we just discussed, case structures are used to represent the meaning of a sentence independent of its particular surface form. We saw that Schank's CD extends the case grammar by reducing all verbs into a set of limited primitive verbs.

BUP produces an output similar to the case frame seen earlier:

```plaintext
((ACTION (turned))
 (AGENT (Jeff))
 (OBJECT (the valve))).
```

However, this case frame is still lacking in some key aspects. The ACTION slot contains an action verb, not a primitive verb. In addition, pertinent information needed to produce a full explanation of the action is still needed. As will be seen later, once the verb is analyzed, all of its frame roles are filled, and the verb is reduced to a primitive form, a complete understanding of the user's intentions is well represented. At the present time, though, the BUP output may be considered an incomplete case frame.

2 Its similarity to the TEMPUS track-oriented animation system is worth noting [5].
Continuing with the case frame concept, every case frame contains a list of roles, each of which is dependent upon the verb. Roles, as in Table 4-2, allow the concepts within a sentence to be classified into their contextual meanings.

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT</td>
<td>the instigator of an action</td>
</tr>
<tr>
<td>OBJECT</td>
<td>an obligatory case found or implied with each verb</td>
</tr>
<tr>
<td>INSTRUMENT</td>
<td>the object used to perform the action</td>
</tr>
<tr>
<td>LOCATIVE</td>
<td>the location (source - destination) of the object</td>
</tr>
<tr>
<td>DIRECTIVE</td>
<td>the direction (source - destination) of the object</td>
</tr>
<tr>
<td>STATIVE</td>
<td>the state (initial - final) of the object</td>
</tr>
<tr>
<td>TIME</td>
<td>the time (initial - final) used for the action</td>
</tr>
<tr>
<td>TEMPORAL</td>
<td>temporal conditions imposed on the action</td>
</tr>
</tbody>
</table>

Table 4-2: Case Frame Roles

4.3. A Particular Case

For example, let the object, valve-x, be a rotary valve with six states (or vertices) labeled 1, 2, 3, 4, 5, and 6:

```
3 4
2--o 5
1 6
```

The current value of valve-x is 2.

The sentence, "Ask Jeff to turn valve-x from 2 to 5" would be parsed by BUP into the frame,

```
((ACTION (ask))
 (AGENT (you))
 (OBJECT (Jeff))
 (INSTRUMENT
  ((ACTION (turn))
   (AGENT (Jeff))
   (OBJECT (valve-x))
   (STATIVE (INITIAL (2)))
   (STATIVE (FINAL (5))))))
```

where, according to a transformational grammar rule entitled object-controlled equi [20] the sentence is transformed into two sub-statements, "Ask Jeff" and
"Jeff to turn valve-x from 2 to 5." They, in turn, are changed into a case frame specified by the action verb. Following certain syntactic rules and semantic features of the constituents, their roles are filled accordingly.

The sentence "Ask Jeff to turn valve-x to the right," although similar to the previous sentence in that state 5 is "to the right" of its present state, would be parsed into a different case frame:

```
((ACTION (ask))
 (AGENT (you))
 (OBJECT (Jeff))
 (INSTRUMENT
   ((ACTION (turn))
    (AGENT (Jeff))
    (OBJECT (valve-x))
    (DIRECTIVE (DESTINATION (the right))))))
```

The first sentence provided a full account of what action is to be done (in this case by the simulator). The simulator knows (or eventually will know once the case frame is reduced to a suitable message format for the simulator) the state change of valve-x, and as long as the action is viable, will be able to perform the event. On the other hand, the second sentence only provides general locative information. "To the right" usually does not provide enough information to enact a specific action. If valve-x cannot be rotated from state 6 to state 1 (the path is not a cycle), then "to the right" can account for any of the four remaining states to the right of the current state 2, states 3, 4, 5, or 6. If state 6 and 1 are connected, then "to the right" can account for all of the states except for the current state 2.3,4

---

3 In other cases, "to the right" is specific. If state 6 and 1 do not form an edge and the current state is 5, then "to the right" would imply the state 6. For other objects, such as a toggle-switch, where only two states exist, if the movement of the object can only be done in a left-right direction, "to the right" is both valid and specific. Finally, if the directional movement of the object is not left-right, "to the right" is not valid.

4 Certain directionals, such as up or down, may be insensitive to an object's directional movement. To turn a valve or a toggle-switch up, implies an increase in the state of the object. Thus an object with a left-right direction movement would allow the direction up to be valid.
Although the case frame contains the information stated in the sentence, there is often a certain amount of information that is still needed for simulation. The frame is incomplete in the sense that it does not produce a complete representation of the statement's meaning. Specific roles pertaining to the action may not have been given. To complete, many verb specific roles must still be filled and/or resolved and possibly an error analysis may be performed.

### 4.4. The Verb's Part In Determining A Complete Case Frame

Every verb has an associated case frame(s). It specifies which roles the verb may take, their preferred ordering, and whether or not any of the roles are syntactically obligatory. For example,

- **GIVE**: [ (agent) (benefactive) object ]
- **KILL**: [ (agent) object (instrument) ]

where the roles in parentheses are optional and the order listed is the preferred order, are two motion (or action) verbs. They assert a change of state or an activity instigated by an agent. As can be seen, the syntactic roles of the two are different. *Give*, as in the sentence "John gives Mary the book," has a benefactive, Mary. But *kill* does not. "John kills Mary the book" is obviously incorrect. *Kill* needs an object, not a benefactive. Thus "John kills Mary" is correct (structurally, not morally!).

In the TEMPUS environment most of the verbs we see are action verbs. Since this is an interface for a simulation, we want to assert a change of state for the object(s) or a motional activity for the agent(s). Of the lengthy lists of action verbs available (see Appendix A and [18, 3]), the verbs we chose to investigate are *put, turn, rotate, open, close,* and *move,* as well as some of their multi-word verb forms, *put on, put aside, put down, put out, turn on, turn off, turn up, turn down, turn out, turn over, open up,* and *close down.* In addition, of the numerous definitions for each verb, we chose the definitions that are pertinent to the given environment (See Appendix B). Many other definitions exist for each verb, but
only those pertinent to motional changes are valid in our case. This limitation, besides being a necessity, also reduces the choices and therefore the analyses needed to determine the user's wants.

Table 4-3: Verb Case Frames

These verbs also have key case frames which follow the main (most used) definition of the verb (see Table 4-3). There may be, however, other case frames for the verb. Many action verbs, as well as other types of verbs, have instances of polysemy (multiple meanings). Usually, the semantical features of certain constituents, the entire sentence, and often an entire concept is needed to determine the true meaning in a given instance.

5Turn off has such definitions as: A. to stop the flow of (water, gas, etc.), as by closing a faucet or valve, B. to extinguish (a light), C. to drive a vehicle or walk onto (a side road) from the main road, and D. slang: to cause (someone) to lose interest; to bore or discourage. A and B are definitely pertinent. C may be pertinent, but not really in such an enclosed environment as the shuttle. D is definitely not pertinent. Nor is it really valid to use it in a command (eg. "Tell John to turn on"). Thus only A and B would be used.
The object of a statement, in these cases an object, often helps in determining the verb’s meaning. For example, consider the verb turn. "Turn the valve on" may imply a rotary movement of the valve to the on position. "Turn the switch on" may imply a linear movement of the switch to the on position. "Turn the light on" may imply that the switch for the light, not the light itself, is "turned on." In addition, "Turn out the light" is a valid sentence. But, "Turn out the switch" or "Turn out the valve" are not valid sentences. The verb, turn out, is only valid for a specific type of indicator. By knowing specific information about an object, proper definitions for the verb can often be chosen. And if errors occur, informative error messages can be made pertaining to the verb and object.

Finally, as shown in Section 4.3, prepositional phrases (also called conceptual categories) can help determine the user’s intentions by adding additional information about the action. In fact, many of the necessary slots are filled by prepositional phrases.

As a result, the meaning of a statement can often be found only after the verb and its roles have been analyzed. BUP changes the sentence into an incomplete case frame. The next step is to analyze the incomplete frame and create a true case frame. Both the verb and its filled case roles, plus any default information from former discourse that may be pertinent to the present statement, must now be used to fill and/or correct the case frame. A complete understanding of the action intended by the user must be determined, else incorrect actions may occur.
The Object Knowledge Representation

As stated earlier, a command issued by the user may not provide enough information to enact an action. Even if enough information is given, there needs to be a way to check whether the information is correct. Conceptual dependency frames are, as the name implies, dependent upon the concepts of the sentence. A key aspect of it is the instrumental object of the sentence. Without a clear understanding of the object, the most that could be expected would be a simple, strictly syntactical breakdown of the command. We need more, obviously.

An object frame is a structure, with slots (roles) and values, describing specific details about an object's characteristics; its type, states, motion, and so on. The knowledge needed to infer an object's operation, motion, or change must be embedded in a well defined structure to allow such inferences. Presently the frame structure for each object, including the agents, are shown in Table 5-1. The isa-parent, control, and indicator frames are complete; their use can be seen in Appendix C. However, moving object frames are not complete. Future research will alleviate that problem.
Table 5-1: The Object Knowledge Base

- **If Object is a *ISA-PARENT***
  - Children of the parent (i.e., the objects that can be called by the parent object.)
  - General location of the group of objects

Else...

- **Name of the object**
- **Texture mapping**
  - A texture map pointer or a pointer to and from the object’s SurfsUP description
  - Object center coordinates
  - Object extent box
- **Message passer**
  - Sends messages specific to the object
Table 5-1, Continued

- If Object is a CONTROL
  - Type of control,
    - Switch, Circuit-Breaker (CB), Valve, Latch, etc.
  - Sub-type of control,
    - Switch: pushbutton, pushbutton-with-light, thumbwheel, valve, toggle, display-select-switch
    - Cb: toggle, push/pull
    - Valve: lever, rotary
    - Latch:
      - etc.
  - The control's location
  - The possible directions for the control's state-change
    - Left-Right
    - Up-Down
    - Release-Push
    - Push-Push
    - Push-Pull
    - Counterclockwise-Clockwise
    - etc.
  - Ordered labeling of the control's states
  - Amount of movement from one state to the next state, or vice versa
    - discrete or continuous range
    - type of measurement (i.e., degree or mm.)
    - type of movement (i.e., rotary or linear)
    - an account for each state and the next
      - the states
      - the amount of travel needed to reach the next state
      - the amount of force needed to enact the movement
  - Current state (variable)
Table 5-1, Continued

- **If Object is an **INDICATOR**
  - Type of indicator
    - meter, discrete-event, tones
  - Sub-type of indicator,
    - Meter: bar, digital, rotary
    - Discrete-Event: talkback-gray, talkback-legend, talkback-barber-pole, light, off-flag
    - Tones:
  - The indicator's location
  - The way in which a person would perceive its state change
  - Ordered labeling of the indicator's states
  - Connections to a control, containing lists of the quadruples:
    - a primitive verb
    - an object
    - the object's locative
    - the state the object must be at (or nil)
  - Current state (variable)

- **If Object is a **TOOL**
  - For now, consider it a solid object
  - Is it static or dynamic? (i.e., can it be moved?)
  - Miscellaneous slots
    - the tool's use
    - the area of the tool to be held
    - preferred grasp configuration
    - etc.

- **If Object is a **CLOTHING**
  - For now, consider it a solid object
  - Miscellaneous slots
    - the clothing's use
    - the clothing's size
    - folding transformations
    - etc.
Table 5-1, Concluded

- **If Object is a STOWAGE**
  - With or without restraint?
    - If with restraint, a pointer (in SurfsUP) to the stowage-restraint frame
  - Pointer (in SurfsUP) to the objects contained in the stowage
    - Objects would change as they are removed from or replaced into the stowage area

- **If Object is a STOWAGE-RESTRAINT**
  - i.e., handle, latches, bungees, etc.
  - If latch or handle, it would be part of SurfsUP’s object hierarchy.
  - Else, it would be a separate object with a separate frame, still accessible from a pointer to SurfsUP

- **If Object is a CRT**
  - Viewing center
  - Pointer to SurfsUP geometric description
    - Output image on the CRT (perhaps only cosmetic)

- **If Object is an AGENT**
  - Personal Attributes
    - **name**
    - **nicknames**
    - **gender**
    - **role** (Pilot, Commander, Mission Specialist, etc.)
    - **preferred handedness** (left, right, or ambidextrous)
    - **responsibility** (geometric area, panel, etc.)
    - **pointer** to ADB entry and associated anthropometric parameters
    - **reach** model (workspaces generated from TEMPUS)
    - **strength** model (from specific task or general data)
  - A Special Message Passer
    - Unlike inanimate objects, an agent would be intelligent. As such, it would pass and receive messages in a more intelligent, commutative method than the other objects. Besides sending and receiving messages in a straightforward fashion (as in the other objects), the agents can also pass messages to one another. The result should alleviate certain problems that may occur during multi-agent planning for a common goal.
Two specific types of object frames exist: virtual objects, represented as an *isa-parent*, and physical objects, represented by any of the other frames.

### 5.1. The Isa-Parent Structure

The *isa-parent* frame is used as a typical *isa* taxonomic hierarchy. Containing its *children*, it allows a user to specify a general object. For example, in Figure 7-1, *twF-1, twF-2, twF-3, and twF-4* are children of the *isa-parent, twF*. If a user states a command such as, "turn twF to 3," although a more specific object is needed, the set of choices is now limited to only the four. The command is not declared invalid. In addition, the general locative of all of its specific objects is known.

### 5.2. Physical Object Structures

The other frames specify a single object. Although each of these frames are very different, certain roles are common to each. The *name* of the object provides the full version of the object's abbreviated name. For example, *twF-1* is actually "thumbwheel F-1." The *texture mapping* provides a graphical mapping of the object to SurfsUP. Finally, the *message passer* allows a message to be included in the object's structure. The message can be used to provide or obtain certain information that is specific to only that object. Other roles in the frame depend on the type of object.

#### 5.2.1. The Control Structure

A *control* is an object with many characteristics, some totally unique. It is used to *control* certain events in the environment. The *types* of controls such as switches, circuit-breakers, valves, etc., all contain a number of sub-types. These types and sub-types help to provide general information about a control. For example, a rotary valve imparts the knowledge that the control moves in a
circular movement. However, that is not enough in most cases. Other roles are needed. The location of the control helps in resolving any ambiguities that may occur; if two or more controls at different locations (panels) have the same name, a specified location would help in determining which one was meant by the user.

Besides general information about a control, very specific information is needed too. Exact knowledge about the movement of a control is very important.

Since each control contains some sort of movement, the set of directions it can possibly move toward are important. Besides providing the direction of movement, it may also help in determining a specific state change. But first, we must describe the state list. The states of the control are the vertices obtainable by changing the position of the control. Provided the control has discrete states (to be discussed shortly), each discrete position of the control is a state. They are labelled in a particular order, from lowest to highest position. The directionals are labelled accordingly. The first direction points toward the lowest state; the last toward the highest. As such, if no state was provided in the input command, but a directional was, the list of possible states can usually be limited, sometimes to only one (see Chapter 4.3, Footnote 3 and 4).

The next characteristic is very involved. It is used to determine the exact movement needed to follow a path from one state to another.

The first part involves the type of movement, either discrete or continuous. A discrete movement implies that the control has individual state positions. If depicted as a graph, the positions would be represented as vertices (states). However, continuous movement does not provide such information. The most it may have are two states, such as off and on. The rest of the movement can only be stated as a percentage; somewhere between off which is $0\%$ and on which is $100\%$. Since this knowledge representation is being used for a simulation, a sort of generality will be used. Although a control may be continuous, virtual states will
be listed. The percentage given will be rounded to a specific state. For example, a continuous control may now have the virtual state list, \( \text{off} \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ \text{on} \). Thus 50% would imply state 5.

Next is an indication of what type of measurement is used to go from one state to another. If the control follows a rotary path, such as a rotary valve, the type would be \textit{degrees}. If it follows a linear path, it would be \textit{mm's}.

Although it seems that the type of measurement would correctly imply the type of movement, it is not always the case. A thumbwheel is moved using a \textit{mm} measurement. However, it is actually a rotary device. The reason for the discrepancy stems from the fact that the rotary movement is hidden from the user. All that is seen is the current value of the thumbwheel. As such, the object is moved by sliding a finger up or down (or left or right). This is measured in millimeters. But, the values are actually shown on a wheel, moving around an axis. To correctly portray the control, the proper type of movement is \textit{rotary}. Most other types follow the \textit{degree to rotary, mm to linear} movement connection.

The final part shows the "workings" of the control. Looking at the state list above, there must be some way to get from one state to another. Using the terminology of vertices (states) and edges (adjoining states), an undirected graph can be created. Each edge depicts two adjoined states. The path it forms can be followed in either direction. However, certain costs are present in the traversal of edges. A certain amount of movement, the length from one vertex to another, provides one cost. It is measured following the type of measurement given earlier. In addition, to move a control to another vertex requires a certain amount of force. This force, at the present time, is an arbitrary number. As a result, changing states can be calculated as a simple path-finding problem. The shortest path, the path costing the least, is chosen.

Finally, the current state of the control, the value seen by the user, must be
contained in the structure. In addition, it must be a variable, since it may be changed often.

5.2.2. The Indicator Structure

An indicator is an object unlike any other; it is not physically moved. Instead it is used to display information about some other event. Although movement does not occur, it does change states, similar to other objects. As such, many of its roles are the same as those in the control frame.

Type and sub-types impart general information about the indicator, the location slot helps in ambiguities, and the current value slot contains the current slot. However the other slots do have unique meanings to an indicator. The state list, besides the usual meaning described in the control frame, also is used as a mapping list for the indicator. To explain this, me must also explain the connection slot.

Since an indicator's state change is due to a change of state in another inanimate object, such as a control, movement is not needed. But, a connect slot is necessary. This slot provides an implicit connection between an indicator and its control. The connection is implicit since its use cannot be done by a physical connection. Instead, it's a virtual connection. As in the case where a person is asked to change the state of an indicator (i.e., "Turn the light on."), the person must have knowledge about which control will allow the change. Once that information is found, its control (i.e., switch) can be moved to its proper state which corresponds to the indicator's state change.

A bijective mapping between an indicator's state list and its connect slot cells exists. If a specific state is specified in the command, its position in the state list corresponds to the position of a member of the connect slot. The correspondence is ordered. If the first state is chosen, the first item in the connect
slot will be chosen, and so on. As can be seen, this is necessary to enact the proper state change in an indicator.

Each position in the connect slot contains a number of lists, each with the four values: primitive, object, locative, and state. The number of lists, \((P_1 O_1 L_1 S_1), (P_2 O_2 L_2 S_2), \ldots, (P_n O_n L_n S_n)\), depend on what motions are needed to perform the implied action. The first states which primitive verb is to be used for the movement of the control. The second states which object, usually a control, is to be used. The third states the general location of the object; it removes any possible ambiguities, as mentioned earlier. The fourth states the proper state change needed for the object to perform the state change indicated for the indicator.

As a result, a command such as, "turn the light on," can be enacted. First, the state on in found in the light's state list. After noting its position in regard to the other states, the same position is found in the connection slot. Finally, the lists at this position are individually evaluated and the proper state change for the object (switch) is enacted. The result is a state change for the object and a corresponding state change for the indicator. The indicator is now at the state the user wanted.

The other role that has not been discussed is the perception role. State changes for objects other than an indicator are perceived in only one method; they are seen by the user (and artificially seen by the simulated agent). Certain indicators may be perceived by another method. There exists a type of indicator called tone. Its state change is signified by the emission of certain sounds. To perceive the state change, it must be heard, not seen. In addition, there may exist indicator's that do both: produces visible and audible state changes. This slot labels the type of perception that is needed to note its state change.
5.2.3. Other Inanimate Structures

Tools, Clothing, Stowages, Stowage-Restraints, Crts, etc. all require separate frames. Unfortunately, we do not have any full structures completed. Part of the reason stems from the fact that a mapping from the knowledge representation to SurfsUP does not exist, yet. Since all of these objects have certain characteristics that involve more than a simple state change, the mapping is necessary. Some of the objects are not fixed to a stationary location such as a panel. Some have very complicated physical structures; they have many moving points. And some have both. This will researched at a later time.

5.2.4. The Agent Structure

Although an agent does have many characteristics similar to the above, unstructured objects, certain roles are understood. An agent's personal attributes provide specific information about the entity. Names, nicknames, genders, and specific roles all help in identifying the agent. The other personal attributes are useful for simulation. The message passer, although used in the inanimate objects, is more complex. Since the agent is "intelligent," it must be able to both send and receive messages, including to and from another agent. It would also be very useful for the simulation.

At this point, without a true working simulation, the full structured frame does not exist. Only the parts used for identification are used. Later on it will be completed.
5.3. Conclusions

For a real person, much, but not all, of the information about the objects is implicit. Many of the facts in these frames would be learned through a person's senses. *Sight* allows one to determine the amount of movement. *Touch* allows one to determine the amount of pressure needed, whether or not the object has discrete or continuous positions, and whether or not any or all states are attached. And, *language comprehension* substitutes implied connections. A simulated person, however, cannot do the same. But by creating a database for each object, the system can appear to know the same kind of information.
CHAPTER VI
Primitive Verbs

Miller [18] states that many motion verbs can be defined by a generic verb, generally stated as *travels*, plus some preposition to indicate the direction of the motion. Schank also believes that motion verbs can be redefined. Of the eight possible primitive verbs seen in Table 4-1, only a few can be used for physical movement, *Ptrans, Propel, Grasp*, and *Move*.

In our system, we are involved with only a set number of movements, e.g.,
1. Agents can walk (or more aptly, fly in the air) from one location to another.
2. They can physically move objects from one location to another.
3. They can write. This involves movements of their arm and hand.
4. They can "talk." This involves movements of their mouths.
5. They can look at something. This involves movements of their head, body, and eyes.

All of these involve movement of some type. After all, our concern involves motion. Although they all could be reduced to a single primitive verb, our decision was to divide the primitives a little less drastically. At this point only two primitives are used, *PMove* and *PLook*. *PMove* accounts for a physical movement of some entire object, or the entire part that can be moved. 1 and 2 would use *PMove*. *PLook* would be used for 5. 3 and 4 have not yet been reduced to a primitive. As a result, action verbs such as move, turn, put, open, and close would be replaced by *PMove*. A verb like check, when used to verify some object's state,
would be reduced to \textit{PLook}. The simulator would now have to be concerned with only a small set of primitive verbs.
CHAPTER VII
Code Analysis

7.1. Introduction

The interface system was written in VAX Lisp [21], a derivative of Common Lisp. See Appendix E for a list of the functions. Its input is a command from a user in an interactive mode. From that point, it follows the order (illustrated in Figure 1-1):

1. The input command is parsed by BUP into an incomplete case frame.

2. The slots of the case frame are validated and any discrepancies are resolved. The result is another incomplete case frame, with all of its filled slots being totally valid.

3. The new case frame is now made complete. Any of the necessary unfilled slots are now filled, either from the existing information given and the object's knowledge base or from a query to the user. The result is now a complete case frame containing a primitive verb and all the slots needed.

   • If the object is an indicator, its connection slot is evaluated and the objects that control its state change are made into complete case frames.

4. The complete case frame is sent to the primitive verb's function. It evaluates the information and returns a formatted message.

5. The message is sent to the simulator.

   • At this point, only a mock simulator exists. This simulator records state changes and returns statements, but no graphical output.

   • Once HIRES is finished, it will replace the mock simulator (gladly). True simulated interactions will occur.
7.2. A Test Panel

Figure 7-1 shows a test panel designed to exemplify a number of objects found on the space shuttle. It is used as a test case for the code. See Appendix C for each object's knowledge representation structure.

- $tbA$, $tbM-1$, and $tbM-2$ are talkback indicators that indicate an on or off state.
- $vlvB$ and $vlvD$ are rotary valves.
- $pbC$ is a push button.
- $meterE$ is a bar meter indicator.
- $twF$ are thumbwheel switches.
- $push/pullG$ is a switch that moves in the Z axis.
- $pb-ltlH$ and $pb-ltlI$ are push-buttons with an indicator light.
- $tglJ-1$, $tglJ-2$, $tglJ-3$, $tglK$, $tglL-1$, and $tglL-2$ are toggle switches.
tbA, vlvB, and pbC are all internally connected.
  * vlvB determines which tbA is on.
  * pbC is a momentary switch. Push is on, release is off.
  * To turn a tbA on, rotate vlvB to a new value and push pbC.

vlvD and meterE are internally connected.
  * meterE displays the position on vlvD.

twF has four separate thumbwheels, each having states 0 through 9.

push/pullG is a push/pull type of circuit breaker. Push is on, pull is off.

pb-ltH and pb-ltI are push buttons with lights. When either is pushed, the light changes its present state.

tglJ-1, tglJ-2, and tglJ-3 are 3 distinct toggle switches. Each moving in an down-up direction.

tglK is a toggle switch moving in a left-right direction.

tglL-1, tbM-1 and tglL-2, tbM-2 are internally connected. If the toggle is in the up position, the talkback is on.
7.3. Syntax Rules

7.3.1. Limitations

The type of sentences any natural language system can accept is always limited. Since the parsing of the sentences require rules, and the linguistic community still has many questions at to whether or not some rules are valid [20], natural language systems are not able to determine the proper syntax for every possible sentence a person may write (or speak).

The same problem exists in this interface system. The type of commands allowed by the system (BUP) are very limited. Only certain specific statements will be accepted. Although this may be a drastic hindrance, in that a totally general English statement will not be allowed, at this point its the best we could do. Further research and programming will allow many more types of statements, although not all possibilities.

The syntax rules used in BUP comprise two specific cases:

1. Commands that begin with the verb *ask* or *tell*, such as, "Ask Jeff to turn the switch on."

2. Commands that start with an action verb, such as, "Turn the switch on."

In the first case, "ask Jeff to" indicates the agent to perform the action. In the second, an agent is implied to be the same as one previously called; a default value. If no agent has been called yet, a query to the user is made. In either case, it is the instrumental role of the sentence (see Table 4-2) that is most important. Both cases have virtually identical instrumental roles, the only difference being that the second case may have a different agent.

The words allowed by the syntax rules are also very limited. *Agents* must be the name of a defined representational structure or one of its other names, also
defined. *Objects* must also be the names of defined structures, or at least abbreviated versions of the structure. Likewise, *locatives* must be defined too. *States* must be a member of the object’s states. If an object contains the state list, (0 1 2 3 4 5 6) (see Appendix C), **only** a member of the state list can be used. Finally, *directionals* may be any object, agent, or directional word, as long as it is included in the syntax vocabulary. It is not always limited to the object’s direction slot (see Chapter 4.3, Footnote 4).

Once the limitations are understood, many commands can be parsed. In addition to specific objects, pronouns are allowed. *He* or *she* can be used to indicate an agent and *it* can be used to indicate the previous object.

The abbreviation part of the system allows the user to use partial abbreviations for an object’s name. It is a limited function in that it only allows abbreviations that consist of the first *n* letters of the real name. But, it is useful. If the user enters an abbreviation such as, *tgl*, in our test panel that can stand for *tglJ-1*, *tglJ-2*, *tglJ-3*, *tglK*, *tglL-1*, or *tglL-2*. As such, the user would be queried for one of these possibilities.

### 7.3.2. Features

As can be seen in Appendix D, many of the rules contain features. The features are the semantic conditions, the augmentations, used to check many aspects of parsed constituents. For example, if an agent is correctly parsed, it adds the feature *animate* to the list. Other rules checked as the parsing continues up must allow such a feature. More specifically, a verb such as *open*, contains, as one of its features, *transitive*. Transitive implies that the sentence must have an object as well as a subject. The sentence, "Jeff opened the door" shows an example of subject and object. "Jeff" is the subject and "the door" is the object. It is a grammatically correct sentence. However, "Jeff opened" is **not** correct. Since the verb is not intransitive, it is missing an object. An intransitive verb,
such as turn does allow a missing object. "Jeff turned" is fine. In reality, the true meaning is, "Jeff turned Jeff." In addition, since turn is also transitive, "Jeff turned the knob" is also correct. Other features like locative, directive, and stative are also very useful in determining whether prepositional phrases are correctly stated; whether the words are what they should be.

7.3.2.1. Conceptual Categories

Since the present environment involves a single panel, where each object is involved in a simple state change, only certain categories are present. These categories are, as stated above, locative, directive, and stative. Once a prepositional phrase is determined as one of these, a particular feature is imparted upon it: L for locative, D for directive, or S for stative. For example, "...on panel1" has the feature L, "...to the right" has the feature D, and "...to 3" has the feature S. If the original sentence was, "Turn vlvD on panel1 to the right to 3," the verb, turn having, among others, the feature LDS, would allow these conceptual categories. A number of other verbs would, too. But, a verb like turn up would not. It only allows a combination of statives and/or locatives; no directives. If completely parsed, it is somewhat definite that the sentence is grammatically correct. The verb's case frame should be valid too.

7.3.3. Agent Variations

As can be seen in Table 5-1, an agent may have nicknames, a role name and a gender. To allow a certain amount of flexibility, any of these can be used to identify a specific agent. If either a nickname or role is used, the structure it represents is found and its official name is returned. If a gender is used (he or she), one of three case occur.

1. If the previous agent has the same gender, it is chosen.

2. If there exists only one agent of that gender, it is chosen.

3. Else, the user is queried as to which of the agents with that gender is meant.
The result is an official name for an agent structure.

7.3.4. Performance

Although BUP is not very fast, it does do a relatively good job. Its syntax rules are limited, but adequate for a test example. The output, determined by the syntax rules, is an incomplete case frame, as discussed earlier.

7.4. Resolve Ambiguities

In this part of the system, the command has already been parsed. It now is in the form of an incomplete case frame. However, there may be certain slots whose values are not specific. There also may be certain unfilled slots that can be filled by either default information or a user query. Finally, certain information may be incorrect.

Once resolved, any discrepancies that may have occurred will have been dealt with. Another incomplete case frame will exist, however it will now contain specific information, as well as certain default information as needed.

7.4.1. Non-Specific Entities

An isa-parent object is not specific. It only specifies a number of *connected* objects. A user query is made to provide the specific object. Often an abbreviated name for an object is also not specific. It may point to a list of possibilities. Both cases will be discussed below.
7.4.2. Default Information

After the first command has been entered, certain information is saved for future commands. As in the FDF checklists (see Chapter 2), the most previous agent, object, and location are saved as default values. This allows a sentence like "Turn it to the right" to have a meaningful representation. The agent slot of the case frame is filled with its default. It is replaced by the default object, and the missing locative is also filled by the default.

7.4.3. User Query

In certain cases, a query to the user is in order. If this is the first command, no agent default information exists. The user must choose between any of the members of the crew list. If an isa-parent object has been used, it is implied that the user wants one of its children as the object. The user is queried as to which of the children. If the object has been entered in an abbreviated form, and more than one object matches, the user must make the final choice. If a location has been entered and it does not match the object, a query is made. Likewise, if more than one location exists for an object’s name, and no previous location matches, a query is made. If the directional is an object, as in the sentence, "Turn the switch toward the vlv," and the directional object is abbreviated, again, the user may be queried about the choices. Finally, if a choice exists among the possible states of an object, the user must decide.

7.4.4. Incorrect Information

There is always the chance that incorrect information was entered by the user. Errors such as states that are not members of an object’s state list, directions that are not valid to the object, locations that are not valid to the object, and so on. Unlike many other systems, often the user is given a chance to correct the errors. The user is queried about the error and is given the chance to
remedy it. In addition, as in all queries given by the interface, the user always has the option to *abort* and return to the top level of the interface loop.

### 7.5. Action Verbs

As stated in Chapter 4.2, a complete case frame represents an understanding of a user's intentions. However, the user's command may not explicitly state what action is to be done. Often generalities are used versus an exact state change explanation. In the case of a movable object, the mapping coordinates are seldom provided by the user. After all, that is part of the interface's job. In other cases, the final state needed for an immovable object's state change is generalized. Directionals are used instead of specific states. These generalities must be evaluated and changed into specific state change information. To do so, the action verb must be discussed.

#### 7.5.1. Action Verb Conditions

Each verb, as discussed in Chapter 4.4, has its own case frame. Each verb has special conditions pertinent to only itself.

As seen in Figure 7-2, a verb may have a number of cases. *Turn* is both a transitive and intransitive verb. It allows *self referencing*. The first case in the code is intransitive; an agent *turns* itself. What is needed is a specific mapping for the agent to turn (see Subsection 7.5.2). The next two cases concern a control or indicator. In these cases, the specific state needed for the state change is wanted (see Subsection 7.5.3). Finally, the last case involves one agent *turning* another. As can be seen this is not yet possible (again, see Subsection 7.5.2). Returning to the case involving an indicator, a condition exists. As in many verbs, certain conditions are imperative. In this case, either a final state must be present or a certain directional, *on, off, or out*. Providing one of these exists in the case frame, the calculations can be continued. For an indicator, what is really needed
(DEFUN turn-verb
  (agent object locative s-initial s-final d-source d-dest temporal)
  "The action verb, turn."
  (cond ((null object)
    (apply-primitive
     (list 'PMove
         agent
         agent
         (determine-moving-object-mapping
          'turn-verb agent 'directive d-dest locative)
         temporal)))
    ((control-p (eval object))
     (apply-primitive
      (list 'PMove
        agent
        object
        locative
        (or (determine-state object 'stative s-final '/=)
            (determine-state object 'directive d-dest '/=)
            (determine-state object nil nil '/=)
            (query-user
             object (determine-subset object T '/=) 'state))
        temporal)))
    ((indicator-p (eval object))
     (if (or (s-final
       (member d-dest '(on off out)))
       (apply-primitive
        (specific-indicator-connections
         agent
         object
         locative
         (or (determine-state object 'stative s-final '/=)
             (determine-state object 'directive d-dest '/=)
             (determine-state object nil nil '/=)
             (query-user
              object (determine-subset object T '/=) 'state))
        temporal))
      (if (null d-dest)
        (error-msg
         "The indicator, " object ", needs a directive or state.")
        (error-msg
         "The indicator, " object ", can not be turned " d-dest))))
    ((agent-p (eval object))
     (error-msg "At this point, we can't turn " object))))

Figure 7-2: Lisp Code For The Verb TURN

is a command to its controls. To find the exact controls and their proper state
changes, the precise state change chosen for the indicator must be either verified or determined. Then, the controls can be moved and the indicator can return the state the user intended (see Chapter 5.2.2 for a further discussion of indicator state changes).

7.5.2. Movable Object State Changes

A movable object, such as an agent, may be instructed to be moved. What is needed is the mapping to SurfsUP. As stated earlier, however, at this point this cannot be done. The output of such a command is not able to be simulated. Instead, a pseudo message is returned, without further computation. This problem will eventually be remedied once HIRES is completed.

7.5.3. Immovable Object State Changes

Immovable objects, in this case, controls and indicators, can be used. Their specific state change information can be found from the object's structure.

7.5.3.1. Determining Final States

The given information about a final state, the state that the object should be moved into, can be implied in one of three ways:

1. The final state can be given in the command.

2. A directional destination can be given in the command. The destination implies the final state, whereas a source destination would imply an initial state; the current state.

3. The verb can imply the final state.
7.5.3.2. Given State

Often, the state given in the command will be a member of the object's state list. Provided it is valid, in respect to the verb's meaning, it is returned. Two other states are usually not members of the state list. On implies a state greater than the current. If the object contains only two states, on implies the latter. Off, on the other hand, implies a bit more. It implies the first state in the list, regardless of the number of states. Initially this may seem confusing, but the word on only says that the state should not be the first state; the off position. If more than two states exist, there is a question as to which of the states is meant. Off says that the object should be in its first, or lowest state; the equivalent of "off." No matter how many states may exist in the object, off points to the first.

7.5.3.3. Given Directional

Determining states from directionals can be a "sticky" situation. Many cases exist. They will be dealt with one at a time. In all cases, though, the function contains a qualifier, >, <, or /=, determining what subset of the object's states are to be considered.

If the object is a control, and its direction slot contains identical directions (see Appendix C, object pbC), depending on the verb and whether the control has only two states (usually the case if all directions are identical), a state can be determined. For example, an control that would have these conditions would be a momentary pushbutton switch. Push it in and the state changes. A verb like, turn would just return the state not currently used. Turn up would return the last state. Turn down would return the first.

For any immovable object, if the directional is a member of the object's

---

6A verb, such as turn, provides not indication as to which state is implied. As long as the given state is not equal to the current state value, it is valid. However, some verbs, such as open, close, turn up, etc., do imply a certain subset of the possible states in the object. Open implies any state greater than the current. Close implies any state less than the current. And so on.
direction list, it may be used. If it is equal to the first direction, the state is one of the states less than the current state. If it is the last, the state is greater than the current. Determining one of these states will be discussed shortly.

Sometimes a user gives a directional that is not a member of the object's direction list. Such general directionals can still point to or toward the proper state. However, these directionals are limited to only certain objects. Up and down are valid for any object. They mean either increase the state, or lower it. Right and left are more limited. Either one of them is a member of the object's directions, in which case it would probably be noted above, or the object has a rotary movement. A toggle may have them as their direction list, but if not, they would be invalid. Rotary valves can use them since in a rotary mode; right implies raise and left implies lower. Clockwise and counterclockwise are valid only for rotary movement objects. The reasons are obvious. In can only be used if it is a member of the object's directions. But, out can be used for other objects. If the object is a discrete-event indicator, out implies off. It is valid in this case. Once the directionals are validated, they all follow a set pattern: up, right, clockwise, and in limit the possibilities to states greater than the current state and the others limit them to states lower than the current.

Finally, if another object is used as the directional, its coordinates must be determined. At this point nothing can be done.

7.5.3.4. Verb Implications

Occasionally, no states and no directionals are included in the command. The only information that can point toward the proper final state is the verb. We stated earlier that verbs may imply a limitation on the state possibilities. Verbs implying no limitations other than the subset cannot contain the current state can only be useful if only two states exist. If so, the other is chosen. Other verbs may direct the search toward the higher or lower states, but usually not to a specific state. If the verb directs toward a lower state and the current state is next to the
first state (the current state is in the second position), the first state is chosen. If the subset determined by the verb has more than one possibility, the user must be asked to provide the final state. A similar result occurs toward a higher state.

7.5.4. User Query

If no amount of computation will result in a specific final state, the user must be asked to provide this information. The list of possibilities presented to the user is limited, though, by the verb's implications. The user can then choose to pick one of the possibilities, or abort the command altogether. If aborted, the interface returns to its highest level.

7.6. Indicators

Indicators comprise a very special case in regard to a verb. Whether the user is aware or not, indicators cannot be moved by the agent. Action verbs used for an indicator do not usually imply the movement of the indicator, but instead, the result of an action. Thus the verb used in a command for an indicator has little to do with the action verb that would be used to command its controls. However, the verb used is needed to provide the specific state change of the indicator. It is this specific final state that provides the key to determining the indicator's controls and states.

As discussed in Chapter 5, there is a mapping from the state to the connection slot. Once the specific connection is found, certain information is again needed. The connection provides the primitive verb, the object (control), its location (panel) and its final state. But, each list in the connection is missing values for a number of slots. However, since those missing values are the same as those in the control's case frame, no problems exist. The empty slots are filled, and each new case frame resulting from the connection is analyzed.
7.7. Primitive Verbs

As stated earlier (Chapter 6), our system is involved with only a set number of movements, all involving action. Right now we are concerned with only two, \textit{PMove} and \textit{PLook}. And of these two, the main concern involves the first, \textit{PMove}. As can be seen in Figure 7-2, the verb is usually changed easily from its action verb to a primitive verb. Occasionally there may be a problem deciding which primitive is better suited, but not often. The primitive verb is now part of the case frame and it is complete. The case frame must now be analyzed to produce the message needed for the simulator.

7.8. Complete Case Frame Analysis

At this point, the only primitive verb that is truly used is \textit{PMove}. \textit{PMove} implies a movement of an object from its present state to a new, final state. Again, being that the only movable objects that we can use presently are controls, the path of the movement is well defined. Controls have no collision problems and their knowledge representation details the movement needed to reach the new state.

To explain how an object is moved, we will use control \textit{vlvB} (see Figure 7-1). Presently, \textit{vlvB} is in state 6 out of the eight possible states. Assume that the user commanded an agent to "turn \textit{vlvB} to state 3." By analyzing the movement slot (see Appendix C), it can be seen that two choices exist:

1. Move clockwise from state 6 to 7, 7 to 8, 8 to 1, 1 to 2, and 2 to 3. The total \textit{cost} would be $5 \times 45$ degrees (the \textit{cost} of each movement), or 225 degrees. The pressure would be 5.

2. Move counterclockwise from 6 to 5, 5 to 4, and 4 to 3. The total \textit{cost} would be 135 degrees and again, a pressure of 5.

The \textit{best path} function determines which direction is "cheaper" and follows that path.

---

\footnote{The control, \textit{vlvB}, contains a simple cycle; the edge (8 1) exists.}
What results from the primitive verb's function is a formatted statement compatible with HIRES. It is a *stock* message containing slots to be filled with their respective values (*the underlined names are the slots*):

```
ASK agent TO MOVE object ON locative
    movement measurement direction
TO STATE < state >
```

In the above example, the message would be (assuming *Norm* is the agent):

```
ASK Norm TO MOVE vlvB ON panel1
    135 degree counterclockwise
TO STATE < 3 >
```

The filled message is then sent to the simulator.

### 7.9. The Simulator

Until HIRES, a hierarchical reasoning simulation system, has been finished, the interface will only be able to provide a *mock* simulation. This mock simulator receives messages like the one above and dissects it into particular segments. Since no graphical mapping exists, yet, between the interface and TEMPUS, no graphical output can be produced. Instead, statements are produced to describe what has been done. Also, the object's states are changed in accordance to the user's commands.

### 7.10. Errors

The interface system provides a relatively concise error check; often the errors can be remedied. If the error is not fatal, the user is queried for a new, correct value for some variable. If it is fatal, an informative error message is given; the user is not left *in the air.* In any case, once a command is evaluated, the user is returned to the top level to continue.
7.11. Object Description

To provide the user with a concrete indication of what exists in an object's structure, there exists a command, describe. Depending on the type of structure, agent, isa-parent, control, or indicator, a complete explanation of the object is provided. No simulation is done, however it does provide the user with a knowledge of the object. Since no graphical output can be seen at this point, describe is very useful in affirming an object's state.

7.12. Creation Of An Agent

Agents may be changed often. As such, there exists a function, create-agent, that does just what it indicates; it creates a new agent with numerous slot values. As stated earlier, not all of the slots are used, but just those useful for the mock simulation. Later, the complete agent structure will be created.
7.13. A Sample Session

What follows is a session using the natural language interface.

- *Italicized* words are the user's input.
- Underlined words are comments added later for emphasis.
- The **bold** words, **BLANK LINE**, are used to indicate the end of the session.

See Figure 7-1 for an illustration of the objects being used and their positions.

---

Creating three agents (crewmembers): Norm, Jeff, and Sally

Lisp> (create-agent 'norm)
Any other names used for NORM?
Enter the name(s) or a blank line : norman badler
What is NORM's role? (enter a single word) commander
What is NORM's sex? (Male or Female) m
Which hand does NORM prefer, Left, Right, or Ambidextrous? r
T

Lisp> (create-agent 'jeff)
Any other names used for JEFF?
Enter the name(s) or a blank line : jeffrey gangel
What is JEFF's role? (enter a single word) pilot
What is JEFF's sex? (Male or Female) m
Which hand does JEFF prefer, Left, Right, or Ambidextrous? l
T

Lisp> (create-agent 'sally)
Any other names used for SALLY?
Enter the name(s) or a blank line : Ride
What is SALLY's role? (enter a single word) ms-1
What is SALLY's sex? (Male or Female) f
Which hand does SALLY prefer, Left, Right, or Ambidextrous? r
T

Lisp> *crew-list*
The crew-list, including other names that may be used for the same agent:
(((SALLY RIDE MS-1)
  (JEFF JEFFREY GANZEL PILOT)
  (NORM NORMAN BADLER COMMANDER))

---
Using The Interface

Lisp> (talk)

You are now entering the TEMPS user-friendly interface. You may start your 'conversation' shortly, but first, let me put down some of the guidelines.

1. Enter a BLANK LINE to end the conversation.
2. To see the list of verbs you may use, type: LIST VERBS
3. To see a description of any object or agent, type: DESCRIBE object
4. Parentheses are NOT needed

>>> list verbs
(Move) (Put) (Put On) (Put Aside) (Put Down) (Put Out) (Turn) (Turn Up)
(Turn Down) (Turn On) (Turn Off) (Turn Out) (Turn Over) (Open) (Open Up)
(Close) (Close Down) (Rotate)

>>> describe tba-6

Tba-6 is a discrete-event at locative Panel1 with two states: Off and On
The current value is <On>.

>>> turn on tba

A general object was specified!
Please choose a more specific object from
   Tba-1, Tba-2, Tba-3, Tba-4, Tba-5, Tba-6, Tba-7, or Tba-8
or enter NIL to stop the processing: 1

There is more than one possible Agent.
Please choose ONE Agent from
   Sally, Jeff, or Norm
or enter NIL to stop the processing: norm

ASK NORM TO PMOVE VLVB ON PANEL1 135 DEGREE CLOCKWISE TO STATE < 1 >
Norm PMoved Vlvb on Panel1 135 degree clockwise to state < 1 >

ASK NORM TO PMOVE PBC ON PANEL1 3 MM IN TO STATE < ON >
Norm PMoved Pbc on Panel1 3 mm in to state < on >

ASK TBA-1 ON PANEL1 TO ACKNOWLEDGE THE CHANGE
Tba-1 on Panel1 was changed to state < on >

ASK PBC ON PANEL1 TO ACKNOWLEDGE THE CHANGE
Pbc on Panel1 was changed to state < off >
>>> describe tba-1

Notice the new current value.
Tba-1 is a discrete-event at locative Panel1 with two states:
  Off and On
The current value is <On>.

>>> describe tba-6

Notice the new current value.
Tba-6 is a discrete-event at locative Panel1 with two states:
  Off and On
The current value is <Off>.

>>> describe vlvb

Vlvb is a rotary valve at locative Panel1 with eight states:
  1, 2, 3, 4, 5, 6, 7, and 8
  in the (counterclockwise clockwise) direction with NO endpoint.
The current value is <1>.

>>> describe pbc

Pbc is a linear switch at locative Panel1 with two states:
  Off and On
  in the (in in) direction.
The current value is <Off>.

>>> rotate vlvb

There is more than one possible State for Vlvb.
Please choose ONE State from
  2, 3, 4, 5, 6, 7, or 8
or enter NIL to stop the processing: 8

ASK NORM TO PMOVE VLVB ON PANEL1 45 DEGREE COUNTERCLOCKWISE TO STATE < 8 >
Norm PMoved Vlvb on Panel1 45 degree counterclockwise to state < 8 >

>>> ask her to rotate pbc

Only rotary motion objects can be rotated.

>>> move pbc in

ASK SALLY TO PMOVE PBC ON PANEL1 3 MM IN TO STATE < ON >
Sally PMoved Pbc on Panel1 3 mm in to state < on >

ASK TBA-8 ON PANEL1 TO ACKNOWLEDGE THE CHANGE
Tba-8 on Panel1 was changed to state < on >

ASK PBC ON PANEL1 TO ACKNOWLEDGE THE CHANGE
Pbc on Panel1 was changed to state < off >
>>> describe vlvd

Vlvd is a rotary valve at locative Panel1 with seven states:
   0, 1, 2, 3, 4, 5, and 6
   in the (counterclockwise clockwise) direction.
The current value is <1>.

>>> close vlvd

ASK SALLY TO PMOVE VLVD ON PANEL1 45 DEGREE COUNTERCLOCKWISE TO STATE < 0 >
Sally PMoved Vlvd on Panel1 45 degree counterclockwise to state < 0 >

ASK METERE ON PANEL1 TO CHANGE TO 0
Meter on Panel1 was changed to state < 0 >

>>> ask him to open it

There is more than one possible Agent.
Please choose ONE Agent from
   Jeff or Norm
or enter NIL to stop the processing: jeff

There is more than one possible State for Vlvd.
Please choose ONE State from
   1, 2, 3, 4, 5, or 6
or enter NIL to stop the processing: 6

ASK JEFF TO PMOVE VLVD ON PANEL1 270 DEGREE CLOCKWISE TO STATE < 6 >
Jeff PMoved Vlvd on Panel1 270 degree clockwise to state < 6 >

ASK METERE ON PANEL1 TO CHANGE TO 6
Meter on Panel1 was changed to state < 6 >

>>> turn out meterE

Meter must be a discrete event indicator, not a Meter indicator.

>>> turn vlvd counterclockwise

There is more than one possible State for Vlvd.
Please choose ONE State from
   0, 1, 2, 3, 4, or 5
or enter NIL to stop the processing: 0

ASK JEFF TO PMOVE VLVD ON PANEL1 270 DEGREE COUNTERCLOCKWISE TO STATE < 0 >
Jeff PMoved Vlvd on Panel1 270 degree counterclockwise to state < 0 >

ASK METERE ON PANEL1 TO CHANGE TO 0
Meter on Panel1 was changed to state < 0 >
>> turn twf up

A general object was specified!
Please choose a more specific object from
  Twf-1, Twf-2, Twf-3, or Twf-4
or enter NIL to stop the processing: 1

There is more than one possible State for Twf-1.
Please choose ONE State from
  1, 2, 3, 4, 5, 6, 7, 8, or 9
or enter NIL to stop the processing: 9

ASK JEFF TO PMOVE Twf-1 ON PANEL1 10 mm up to STATE < 9 >
Jeff PMoved Twf-1 on Panel1 10 mm up to state < 9 >

>> tell norm to move push/pullG

ASK NORM TO PMOVE PUSH/PULLG ON PANEL1 7 MM IN TO STATE < ON >
Norm PMoved Push/Pullg on Panel1 7 mm in to state < on >

>> move tglk

ASK NORM TO PMOVE TGLK ON PANEL1 20 MM LEFT TO STATE < OFF >
Norm PMoved Tglk on Panel1 20 mm left to state < off >

>> turn off tglk

ASK NORM TO PMOVE TGLK ON PANEL1 0 MM LEFT TO STATE < OFF >
Norm PLooked at Tglk on Panel1
  and saw that it is in state < off >

>> turn it off

ASK NORM TO PMOVE TGLK ON PANEL1 0 MM LEFT TO STATE < OFF >
Norm PLooked at Tglk on Panel1
  and saw that it is in state < off >

>> open tbm-1

Only a valve can be opened.

>> move tbm-1

Tbm-1 cannot be moved.
Only objects with moving parts can be moved.

>> turn it off

ASK NORM TO PMOVE TGLL-1 ON PANEL1 20 MM DOWN TO STATE < OFF >
Norm PMoved Tgll-1 on Panel1 20 mm down to state < off >

ASK TBH-1 ON PANEL1 TO CHANGE TO OFF
Tbm-1 on Panel1 was changed to state < off >
>>> ask her to turn out tglL-2

The Control, TglL-2, is not a correct object for the verb TURN OUT.

>>> turn it out

The directive, OUT, is not valid for the object.

>>> turn tbm-2 out

The indicator, Tbm-2, can not be turned Out.

>>> turn out tbm-2

ASK SALLY TO PMOVE TGLL-2 ON PANEL1 20 MM DOWN TO STATE < OFF >
Sally PMoved TglL-2 on Panel1 20 mm down to state < off >

ASK TBM-2 ON PANEL1 TO CHANGE TO OFF
Tbm-2 on Panel1 was changed to state < off >

>>> ask her to move

Sally is a "movable" object.
As such, it would need SurfsUP coordinates to analyze.
(Fmove Sally Sally (to Some Unknown 3 Space Mapping) ())

>>> ask her to movetglK to the right

ASK SALLY TO PMOVE TGLK ON PANEL1 20 MM RIGHT TO STATE < ON >
Sally PMoved Tglk on Panel1 20 mm right to state < on >

Syntax errors; Undefined words.

>>> ask her to fly

Fly is an undefined word.

>>> ask superman to fly to the moon

Moon, Fly, and Superman are undefined words.

>>> BLANK LINE

GOOD-BYE

The session is over.
CHAPTER VIII

Further Research

Many aspects of the natural language processing have not yet been researched. Also, some of the methods used so far have possible flaws that need to be fixed.

- Temporals, an aspect of many action commands have not been included in the present research. As mentioned in Chapter 2, certain actions may occur in parallel, or may interact with each other. Presently they are ignored, but they will be needed at a later point.

- A complete knowledge base for every object in the environment has also not been completed. Although much of this may be dependent upon the simulator, HIRES, some is independent. Research must continue toward bringing the knowledge base and SurfsUP coordinates together.

- The present parsing system is not really adequate for many commands. Containing only one hundred and one rules, it too must be further researched and extended to a much greater scope.

- As mentioned earlier, right now all we have been using were controls and indicators. This is fine for a demonstration, but in reality, moving objects would probably be used the most. The code was designed to allow a relatively easy insertion of other objects, but it still adds many more lines of code, most not easily written.

- We have not really worried about identical object names in different locations. There is always the case (actually it is a fact) that certain objects with the same names can be found in more than one location (such as different panels). Common Lisp contains a very useful system consisting of packages and modules. Identical names in different packages are considered different, alleviating such problems. This system can, and should, be incorporated into the interface. However, a number of functions in the existing interface would have to be changed accordingly.
• The FDF compiler must be finished. It must allow both simple operational callouts, as well as the more involved ones, with conditionals.

• Only eighteen action verbs exist, including the multi-word versions. Many more are both useful and more importantly, needed to provide proper movements in many cases. Some of these verbs may also need to be reduced to more than one primitive action. In addition, it may be more practical to use a set format for each verb. Similar to the BUP rules, a set number of slots may allow a better understanding of what is needed to enact a message.

• Finally, more primitive verbs must be included in the system. Although PMove and PLook are adequate for many commands, more must be included to not result in a unexplained command.
CHAPTER IX

Conclusion

As was demonstrated in the interactive session Chapter 7.13, a useful user interface is possible for the TEMPUS environment. Although further research and a working simulation system is needed, the results show true possibilities. Once further research is done, the results should be very impressive. Simple commands, whether from a file or from a user, can be used to direct exacting results. The final interaction between the interface, HIRES, and TEMPUS will result in simulated, graphical output; much more pleasing than a simple English statement.
BIBLIOGRAPHY


### APPENDIX A

**Action Verbs Used By NASA**

The following list of *action* or *motion* verbs has been obtained from numerous NASA FDF books. Although any action verb would be valid, these provide an insight into the types used in a Space Shuttle.

<table>
<thead>
<tr>
<th>Action Verb</th>
<th>Action Verb</th>
<th>Action Verb</th>
<th>Action Verb</th>
<th>Action Verb</th>
<th>Action Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVATE</td>
<td>DISPOSE</td>
<td>HOOK</td>
<td>PREP</td>
<td>SLIDE</td>
<td></td>
</tr>
<tr>
<td>ADJUST</td>
<td>DOFF</td>
<td>INCREASE</td>
<td>PRESS</td>
<td>SNAP</td>
<td></td>
</tr>
<tr>
<td>ALIGN</td>
<td>DON</td>
<td>INITIALIZE</td>
<td>PULL</td>
<td>STAIN</td>
<td></td>
</tr>
<tr>
<td>ASSEMBLE</td>
<td>DRAW</td>
<td>INJECT</td>
<td>PUSH</td>
<td>STEP</td>
<td></td>
</tr>
<tr>
<td>ASSIST</td>
<td>DROP</td>
<td>INSERT</td>
<td>RAISE</td>
<td>STOW</td>
<td></td>
</tr>
<tr>
<td>ATTACH</td>
<td>EGRESS</td>
<td>INSTALL</td>
<td>RECONFIGURE</td>
<td>SWAB</td>
<td></td>
</tr>
<tr>
<td>BACKOUT</td>
<td>ENGAGE</td>
<td>INSURE</td>
<td>RECORD</td>
<td>SWAP</td>
<td></td>
</tr>
<tr>
<td>BALANCE</td>
<td>ENSURE</td>
<td>LACE</td>
<td>REEDGE</td>
<td>SWITCH</td>
<td></td>
</tr>
<tr>
<td>BECOME</td>
<td>ENTER</td>
<td>LATCH</td>
<td>RELEASE</td>
<td>TAKE</td>
<td></td>
</tr>
<tr>
<td>BITE</td>
<td>EVALUATE</td>
<td>LIFT</td>
<td>REPLACE</td>
<td>TAPE</td>
<td></td>
</tr>
<tr>
<td>CALIBRATE</td>
<td>EXCHANGE</td>
<td>LOAD</td>
<td>REMOVE</td>
<td>TENSE</td>
<td></td>
</tr>
<tr>
<td>CENTRIFUGE</td>
<td>EXECUTE</td>
<td>LOCATE</td>
<td>REPEAT</td>
<td>TIGHTEN</td>
<td></td>
</tr>
<tr>
<td>CHECK</td>
<td>EXTEND</td>
<td>LOCK</td>
<td>REPLACE</td>
<td>TRACK</td>
<td></td>
</tr>
<tr>
<td>CLAMP</td>
<td>EXTRACT</td>
<td>LOG</td>
<td>REPORT</td>
<td>TRANSFER</td>
<td></td>
</tr>
<tr>
<td>CLOSE</td>
<td>FILL</td>
<td>LOOSEN</td>
<td>REPOSITION</td>
<td>TURN</td>
<td></td>
</tr>
<tr>
<td>COLLECT</td>
<td>FIND</td>
<td>LOWER</td>
<td>RESTOW</td>
<td>TURNOFF</td>
<td></td>
</tr>
<tr>
<td>COMPLETE</td>
<td>FIX</td>
<td>MARK</td>
<td>RESTRAIN</td>
<td>TWIST</td>
<td></td>
</tr>
<tr>
<td>CONFIG</td>
<td>FOCUS</td>
<td>MATCH</td>
<td>RETORQUE</td>
<td>UNFOLD</td>
<td></td>
</tr>
<tr>
<td>CONNECT</td>
<td>FOLD</td>
<td>MOUNT</td>
<td>RETURN</td>
<td>UNHOOK</td>
<td></td>
</tr>
<tr>
<td>CONSULT</td>
<td>GEL</td>
<td>MOVE</td>
<td>ROTATE</td>
<td>UNLATCH</td>
<td></td>
</tr>
<tr>
<td>CONTACT</td>
<td>GENERATE</td>
<td>NOTE</td>
<td>RUN</td>
<td>UNSCREW</td>
<td></td>
</tr>
<tr>
<td>COPY</td>
<td>GET</td>
<td>NOTIFY</td>
<td>SCREW</td>
<td>UNSNAP</td>
<td></td>
</tr>
<tr>
<td>COUGH</td>
<td>GIVE</td>
<td>OBSERVE</td>
<td>SCRUB</td>
<td>UNSTOW</td>
<td></td>
</tr>
<tr>
<td>CUTO</td>
<td>GOTO</td>
<td>OPEN</td>
<td>SECURE</td>
<td>UNWIND</td>
<td></td>
</tr>
<tr>
<td>DECREASE</td>
<td>GRAB</td>
<td>OPERATE</td>
<td>SELECT</td>
<td>VERIFY</td>
<td></td>
</tr>
<tr>
<td>DESCEND</td>
<td>GRASP</td>
<td>FEELBACK</td>
<td>SET</td>
<td>ZOOM OUT</td>
<td></td>
</tr>
<tr>
<td>DETACH</td>
<td>HANDLOosen</td>
<td>PERFORM</td>
<td>SETUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>HANdTIGHTEN</td>
<td>PLACE</td>
<td>SHAKE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISENGAGE</td>
<td>HOLD</td>
<td>POSITION</td>
<td>SHOW</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX B

Selected Verb Definitions

The following are definitions for the action verbs, put, turn, rotate, open, close, move, put on, put aside, put down, put out, turn on, turn off, turn up, turn down, turn out, turn over, open up, and close down. Their definitions are limited to those pertaining to physical motion.

PUT
[V.T.]
- to bring into a specified position
- send, thrust
- to throw with an upward pushing motion
[V.I.]
- to take a specified course

TURN
[V.T.]
- to move or cause to move around an axis or center
- to twist so as to effect a desired end
- to change or cause to change position
  by moving through an arc of a circle
- to cause to move around a center so as to show another side of
- to reverse the sides or surfaces of
- to set in another esp. contrary direction
- to change one’s course or direction
- transfer
- to go around

ROTATE
[V.T.]
- to turn about an axis or a center

OPEN
[V.T.]
- to change or move from a shut position
- to make or become functional
- to make openings in
CLOSE
[V.T.]
- to bar passage through
- to bring together the parts or edges of

MOVE
[V.T.]
- to go or cause to go from one point to another
- to change or cause to change place, position, or posture

PUT ON
[V.I.]
- to dress oneself with (an article or articles of clothing)
- to cause to be performed, as a show

PUT ASIDE
[V.I.]
- to store up; save

PUT DOWN
[V.I.]
- to write down; register; record
- to suppress; check

PUT OUT
[V.I.]
- to extinguish, as a fire

TURN ON
[V.I.]
- to cause (water, gas, etc.) to flow, as by opening a valve
- to switch on (a light)
- to put into operation; activate

TURN OFF
[V.I.]
- to stop the flow of (water, gas, etc.)
- to extinguish (a light)

TURN UP
[V.I.]
- to uncover; find
- to intensify or increase
- to appear; arrive

TURN DOWN
[V.I.]
- to deintensify or decrease

TURN OUT
[V.I.]
- to extinguish (a light)
- to produce as a result of labor
TURN OVER
[V.I.]
• to move from one side to another
• to put in reverse position; invert
• to transfer; give

OPEN UP
[V.I.]
• to go into action, especially to begin firing
• to increase speed or the speed of (a vehicle)

CLOSE DOWN
[V.I.]
• to terminate the operation of; discontinue
APPENDIX C

Representation Structures For Panel 1

The following structures are Common Lisp defstructs; a specific type of data structure. They represent the object knowledge representation for each of the objects (controls and indicators) on the example, Panel 1 (Figure 7-1).

---

**PANEL1**

```lisp
(ISA-PARENT
 :children (TBA VLB METERE PBC
 VLVD METERE TWF
 PUSH/PULLG PB-LTH PB-LTI
 TGLJ TGLL TGLL)
 :locative PANEL1)
```

**TBA**

```lisp
(ISA-PARENT
 :children (TBA-1 TBA-2 TBA-3 TBA-4 TBA-5 TBA-6 TBA-7 TBA-8)
 :locative PANEL1)
```

**TBA-1**

```lisp
(INJECTOR
 :name *TALKBACK-GRAY A-1*
 :texture-map ()
 :message ()
 :type-of DISCRETE-EVENT
 :sub-type TB-GRAY
 :locative PANEL1
 :perception SIGHT
 :states (OFF ON)
 :connect ( () ((PMOVE VLB PANEL1 1) (PMOVE PBC PANEL1 ON)))
 :current OFF)
```

---

73
TBA-2
(INDICATOR
 :name "TALKBACK-GRAY A-2"
 :texture-map ()
 :message ()
 :type-of DISCRETE-EVENT
 :sub-type TB-GRAY
 :locative PANEL1
 :perception SIGHT
 :states (OFF ON)
 :connect ( () ((PMOVE VLB PANEL1 2) (PMOVE PBC PANEL1 ON)))
 :current OFF)

TBA-3
(INDICATOR
 :name "TALKBACK-GRAY A-3"
 :texture-map ()
 :message ()
 :type-of DISCRETE-EVENT
 :sub-type TB-GRAY
 :locative PANEL1
 :perception SIGHT
 :states (OFF ON)
 :connect ( () ((PMOVE VLB PANEL1 3) (PMOVE PBC PANEL1 ON)))
 :current OFF)

TBA-4
(INDICATOR
 :name "TALKBACK-GRAY A-4"
 :texture-map ()
 :message ()
 :type-of DISCRETE-EVENT
 :sub-type TB-GRAY
 :locative PANEL1
 :perception SIGHT
 :states (OFF ON)
 :connect ( () ((PMOVE VLB PANEL1 4) (PMOVE PBC PANEL1 ON)))
 :current OFF)
TBA-5

(INJECTOR

:name "TALKBACK-GRAY A-5"
:type-of DISCRETE-EVENT
:sub-type TB-GRAY
:locative PANEL1
:perception SIGHT
:states (OFF ON)
:connect ( () ((PMOVE VLB PANEL1 5) (PMOVE PBC PANEL1 ON)))
:current OFF)

TBA-6

(INJECTOR

:name "TALKBACK-GRAY A-6"
:type-of DISCRETE-EVENT
:sub-type TB-GRAY
:locative PANEL1
:perception SIGHT
:states (OFF ON)
:connect ( () ((PMOVE VLB PANEL1 6) (PMOVE PBC PANEL1 ON)))
:current ON)

TBA-7

(INJECTOR

:name "TALKBACK-GRAY A-7"
:type-of DISCRETE-EVENT
:sub-type TB-GRAY
:locative PANEL1
:perception SIGHT
:states (OFF ON)
:connect ( () ((PMOVE VLB PANEL1 7) (PMOVE PBC PANEL1 ON)))
:current OFF)
**TBA-8**

(INDICATOR
  :name "TALKBACK-GRAY A-8"
  :texture-map ()
  :message ()
  :type-of DISCRETE-EVENT
  :sub-type TB-GRAY
  :locative PANEL1
  :perception SIGHT
  :states (OFF ON)
  :connect ( () ((PMOVE VLVB PANEL1 8) (PMOVE PBC PANEL1 ON))
  :current OFF)

**VLVB**

(CONTROL
  :name "VALVE B"
  :texture-map ()
  :message ()
  :type-of VALVE
  :sub-type ROT
  :locative PANEL1
  :direction (COUNTERCLOCKWISE CLOCKWISE)
  :states (1 2 3 4 5 6 7 8)
  :movement (DISCRETE DEGREE ROTARY
               ((1 2) 45 5) ((2 3) 45 5) ((3 4) 45 5)
               ((4 5) 45 5) ((5 6) 45 5) ((6 7) 45 5)
               ((7 8) 45 5) ((8 1) 45 5))
  :current 6)
**PBC**

```lisp
(CONTROL
  :name "PUSHBUTTON C"
  :texture-map ()
  :message (IF (EQUAL (CONTROL-CURRENT PBC) 'ON)
    ((COMMAND
      (MAPCAR #'(LAMEDA
          (CUR)
          (SETF
            (INDICATOR-CURRENT (EVAL CUR)) 'OFF))
      (ISA-PARENT-CHILDREN TBA)))
    (COMMAND
      (SETQ OBJ
        (CONCAT-SYMBOLS 'TBA-
          (STRUC-VALUE
            'VLVB
            'CURRENT))))
    (COMMAND
      (SETF (INDICATOR-CURRENT (EVAL OBJ)) 'ON))
    (SEND '(ASK OBJ ON TO ACKNOWLEDGE THE CHANGE))
    (COMMAND (SETF (CONTROL-CURRENT PBC) 'OFF))
    (SEND '(ASK PBC ON TO ACKNOWLEDGE THE CHANGE))
    (COMMAND (MAKUNBOUND 'OBJ))))

:type-of SWITCH
:sub-type PB
:locative PANEL1
:direction (IN IN)
:states (OFF ON)
:movement (DISCRETE MM LINEAR ((OFF ON) 3 5))
:current OFF)
```
VLVD
(CONTROL
: name "VALVE D"
: texture-map ()
: message (IF (NOT (EQUAL (CONTROL-CURRENT VLVD) (INDICATOR-CURRENT METERE)))
(SEND 'ASK METERE ON PANEL1 TO CHANGE TO (CONTROL-CURRENT VLVD)))

: type-of VALVE
: sub-type ROT
: locative PANEL1
: direction (COUNTERCLOCKWISE CLOCKWISE)
: states (0 1 2 3 4 5 6)
: movement (CONTINUOUS DEGREE ROTARY)
((0 1 45 6) ((1 2) 45 6) ((2) 45 6))
((3 4) 45 6) ((4 5) 45 6) ((5 6) 45 6))
: current 1)

METERE
(INDICATOR
: name "METER E"
: texture-map ()
: message ()
: type-of METER
: sub-type BAR
: locative PANEL1
: perception SIGHT
: states (0 1 2 3 4 5 6)
: connect (((PMOVE VLVD PANEL1 0))
((PMOVE VLVD PANEL1 1))
((PMOVE VLVD PANEL1 2))
((PMOVE VLVD PANEL1 3))
((PMOVE VLVD PANEL1 4))
((PMOVE VLVD PANEL1 5))
((PMOVE VLVD PANEL1 6)))
: current 1)

TWF
(ISA-PARENT : children (TWF-1 TWF-2 TWF-3 TWF-4) : locative PANEL1)
**TWF-1**
(CONTROL
  :name "THUMBWHEEL F-1"
  :texture-map ()
  :message ()
  :type-of SWITCH
  :sub-type TW
  :locative PANEL1
  :direction (UP DOWN)
  :states (0 1 2 3 4 5 6 7 8 9)
  :movement (DISCRETE MM ROTARY
    ((0 1) 10 3) ((1 2) 10 3) ((2 3) 10 3)
    ((3 4) 10 3) ((4 5) 10 3) ((5 6) 10 3)
    ((6 7) 10 3) ((7 8) 10 3) ((8 9) 10 3)
    ((9 0) 10 3))
  :current 0)

**TWF-2**
(CONTROL
  :name "THUMBWHEEL F-2"
  :texture-map ()
  :message ()
  :type-of SWITCH
  :sub-type TW
  :locative PANEL1
  :direction (UP DOWN)
  :states (0 1 2 3 4 5 6 7 8 9)
  :movement (DISCRETE MM ROTARY
    ((0 1) 10 3) ((1 2) 10 3) ((2 3) 10 3)
    ((3 4) 10 3) ((4 5) 10 3) ((5 6) 10 3)
    ((6 7) 10 3) ((7 8) 10 3) ((8 9) 10 3)
    ((9 0) 10 3))
  :current 3)

**TWF-3**
(CONTROL
  :name "THUMBWHEEL F-3"
  :texture-map ()
  :message ()
  :type-of SWITCH
  :sub-type TW
  :locative PANEL1
  :direction (UP DOWN)
  :states (0 1 2 3 4 5 6 7 8 9)
  :movement (DISCRETE MM ROTARY
    ((0 1) 10 3) ((1 2) 10 3) ((2 3) 10 3)
    ((3 4) 10 3) ((4 5) 10 3) ((5 6) 10 3)
    ((6 7) 10 3) ((7 8) 10 3) ((8 9) 10 3)
    ((9 0) 10 3))
  :current 1)
TWF-4
(CONTROL
:name "THUMBHEEL F-4"
:texture-map ()
:message ()
:type-of SWITCH
:sub-type TW
:locative PANEL1
:direction (UP DOWN)
:states (0 1 2 3 4 5 6 7 8 9)
:movement (DISCRETE MM ROTARY
((0 1) 10 3) ((1 2) 10 3) ((2 3) 10 3)
((3 4) 10 3) ((4 5) 10 3) ((5 6) 10 3)
((6 7) 10 3) ((7 8) 10 3) ((8 9) 10 3)
((9 0) 10 3))
:current 4)

PUSH/PULL
(CONTROL
:name "PUSH/PULL G"
:texture-map ()
:message ()
:type-of CB
:sub-type PUSH/PULL
:locative PANEL1
:direction (OUT IN)
:states (OFF ON)
:movement (DISCRETE MM LINEAR ((OFF ON) 7 10))
:current OFF)

PB-LTH
(CONTROL
:name "PUSHBUTTON-WITH-LIGHT H"
:texture-map ()
:message (IF (EQUAL (CONTROL-CURRENT PB-LTH) 'ON)
(COMMAND (PRINT "The light is <ON>")
(COMMAND (PRINT "The light is <OFF>"))

:type-of SWITCH
:sub-type PB-LT
:locative PANEL1
:direction (IN IN)
:states (OFF ON)
:movement (DISCRETE MM LINEAR ((OFF ON) 3 5))
:current ON)
PB-LTI

(CONTROL
    :name "PUSHBUTTON-WITH-LIGHT I"
    :texture-map ()
    :message (IF (EQUAL (CONTROL-CURRENT PB-LTI) 'ON)
        (COMMAND (PRINT "The light is <ON>"))
        (COMMAND (PRINT "The light is <OFF>")))
    :type-of SWITCH
    :sub-type PB-LT
    :locative PANEL1
    :direction (IN IN)
    :states (OFF ON)
    :movement (DISCRETE MM LINEAR ((OFF ON) 3 5))
    :current ON)

TGLJ

(ISA-PARENT :children (TGLJ-1 TGLJ-2 TGLJ-3) :locative PANEL1)

TGLJ-1

(CONTROL
    :name "TOGGLE J-1"
    :texture-map ()
    :message ()
    :type-of SWITCH
    :sub-type TGL
    :locative PANEL1
    :direction (DOWN UP)
    :states (OFF ON)
    :movement (DISCRETE MM LINEAR ((OFF ON) 20 5))
    :current OFF)

TGLJ-2

(CONTROL
    :name "TOGGLE J-2"
    :texture-map ()
    :message ()
    :type-of SWITCH
    :sub-type TGL
    :locative PANEL1
    :direction (DOWN UP)
    :states (OFF ON)
    :movement (DISCRETE MM LINEAR ((OFF ON) 20 5))
    :current OFF)
TGLJ-3
(CONTROL
: name "TOGGLE J-3"
: texture-map ()
: message ()
: type-of SWITCH
: sub-type TGL
: locative PANEL1
: direction (DOWN UP)
: states (OFF ON)
: movement (DISCRETE MM LINEAR ((OFF ON) 20 5))
: current OFF)

TGLK
(CONTROL
: name "TOGGLE K"
: texture-map ()
: message ()
: type-of SWITCH
: sub-type TGL
: locative PANEL1
: direction (LEFT RIGHT)
: states (OFF ON)
: movement (DISCRETE MM LINEAR ((OFF ON) 20 5))
: current ON)

TGLL
(ISA-PARENT :children (TGLL-1 TGLL-2) :locative PANEL1)

TGLL-1
(CONTROL
: name "TOGGLE L-1"
: texture-map ()
: message (IF (NOT (EQUAL (CONTROL-CURRENT TGLL-1)
                          (INDICATOR-CURRENT TBM-1)))
            (SEND 'ASK
              TBM-1
              ON
              PANEL1
              TO
              CHANGE
              TO
            ,(CONTROL-CURRENT TGLL-1))))
: type-of SWITCH
: sub-type TGL
: locative PANEL1
: direction (DOWN UP)
: states (OFF ON)
: movement (DISCRETE MM LINEAR ((OFF ON) 20 5))
: current ON)
TGLL-2
(CONTROL
  :name "TOGGLE L-2"
  :texture-map ()
  :message (IF (NOT (EQUAL (CONTROL-CURRENT TGLL-2)
                          (INDICATOR-CURRENT TBM-2)))
             (SEND '(ASK
                          TBM-2
                          ON
                          PANEL1
                          TO
                          CHANGE
                          TO
                          ,(CONTROL-CURRENT TGLL-2))))

  :type-of SWITCH
  :sub-type TGL
  :locative PANEL1
  :direction (DOWN UP)
  :states (OFF ON)
  :movement (DISCRETE WINDOW LINEAR ((OFF ON) 20 5))
  :current ON)

TBM
(ISA-PARENT :children (TBM-1 TBM-2) :locative PANEL1)

TBM-1
(INDIICATOR
  :name "TALKBACK-GRAY M-1"
  :texture-map ()
  :message ()
  :type-of DISCRETE-EVENT
  :sub-type TB-GRAY
  :locative PANEL1
  :perception SIGHT
  :states (OFF ON)
  :connect (((PMOVE TGLL-1 PANEL1 OFF))
             (((PMOVE TGLL-1 PANEL1 ON)))
  :current ON)
**TBM-2**

(INJECTOR

:name "TALKBACK-GRAY M-2"
:texture-map ()
:message ()
:type-of DISCRETE-EVENT
:sub-type TB-GRAY
:locative PANEL1
:perception SIGHT
:states (OFF ON)
:connect ((PMOVE TGLL-2 PANEL1 OFF))

((PMOVE TGLL-2 PANEL1 ON)))

:current ON)
APPENDIX D
The Syntax Rules

The following syntax rules are designed for use in BUP. The (-if) symbol determines if the rule is valid, provided the sentence's constituent(s) match the word(s) in the rule. The (-f) symbol enters the rule's features to the list in the parser. It is also used to determine if other rules are valid; features must match between higher rules. The (#>n) symbol, where n is a position pointer to a non-terminal in the rule, is replaced with the final result of the non-terminal, once it reaches a terminal.

L = Locative
D = Directive
S = Stative
A = Animate
IA = InAnimate

...........................
...........................
...........................

.................................................................
............... (DET the 'the)
............... (DET a  'a)
............... (DET an  'an)

...........................
...........................

.................................................................
(PRONOUN him dummy-var
  -if (cond ((equal (struc-value *agent* 'sex) 'male)
    (setq dummy-var *agent*))
  (let ((agents (search-for-gender 'male)))
    (cond ((= (length agents) 1)
      (setq dummy-var (car agents)))
    (> (length agents) 1)
      (setq dummy-var
        (query-user nil agents 'agent)))
    (T (p " There are no male crew members."
      nil)))))
  -f '(A male))
(PRONOUN her dummy-var
  -if (cond ((equal (struc-value *agent* 'sex) 'female)
     (setq dummy-var *agent*))
     ((let ((agents (search-for-gender 'female)))
       (cond (= (length agents) 1)
         (setq dummy-var (car agents)))
       (> (length agents) 1)
         (setq dummy-var
           (query-user nil agents 'agent))
       (T (p "There are no female crew members."
           nil))))
  -f '(A female))

(PRONOUN it *object*
  -if (or *object* (and (p "No object has been defined yet."
                        nil))
  -f '(IA))

; any defined agent, specific or not
(SPECIFIC-OBJECT (<word>) (official-name #>1)
  -if (member #>1 (flatten *agent-list*))
  -f '(A D))

; the abbreviated form of any non-bound object
(OBJECT <word>) #>1
  -if (AND (NOT (numberp #>1))
            (NOT (boundp #>1))
            (abbrev-form #>1 *all-objects*))
  -f '(IA D))

; any specific defined isa-parent object
(OBJECT <word>) #>1
  -if (AND (NOT (numberp #>1))
            (boundp #>1)
            (isa-parent-p (eval #>1))
            (not (struc-valuep #>1 'locative #>1))
            (abbrev-form #>1 (struc-value #>1 'member-of)))
  -f '(IA D))
any specific defined object or isa-parent object
(SPECIFIC-OBJECT (<word>) #>=1
 -if (AND (NOT (numberp #>=1))
 (boundp #>=1)
 (object-p (eval #>=1))
 (setq *object* #>=1))
 -f '(IA D))

any specific defined panel
(SPECIFIC-OBJECT (<word>) #>=1
 -if (AND (NOT (numberp #>=1))
 (boundp #>=1)
 (isa-parent-p (eval #>=1))
 (struc-valuep #>=1 'locative #>=1))
 -f '(IA D L))

(OBJECT right 'right -f '(D))
(OBJECT left 'left -f '(D))

(ADV (ADVERB) '(', #>=1))
(ADV (ADVERB ADVERB) '(', #>=1, #>=2))

(ADVERB quickly '(TEMPORAL (quickly))) -f '(T))
(ADVERB slowly '(TEMPORAL (slowly))) -f '(T))
(ADVERB here '(LOCATIVE (here))) -f '(L))
(ADVERB there '(LOCATIVE (there))) -f '(L))
(ADVERB off '(STATIC (FINAL (off))) -f '(S))
(ADVERB on '(STATIC (FINAL (on))) -f '(S))
(ADVERB down '(DIRECTIVE (DESTINATION (down))) -f '(D L))
(ADVERB up '(DIRECTIVE (DESTINATION (up))) -f '(D L))
(ADVERB out '(DIRECTIVE (DESTINATION (out))) -f '(D L))
(ADVERB in '(DIRECTIVE (DESTINATION (in))) -f '(D L))
(ADVERB right '(DIRECTIVE (DESTINATION (right))) -f '(D))
(ADVERB left '(DIRECTIVE (DESTINATION (left))) -f '(D))
(ADVERB out '(DIRECTIVE (DESTINATION (out))) -f '(D))
(ADVERB over '(DIRECTIVE (DESTINATION (over))) -f '(D))
(ADVERB around '(DIRECTIVE (DESTINATION (around))) -f '(D))
(ADVERB clockwise '(DIRECTIVE (DESTINATION (clockwise))) -f '(D))
(ADVERB counterclockwise '(DIRECTIVE (DESTINATION (counterclockwise))) -f '(D))

...... ANY COMBINATION OF L, D, OR S MAY BE USED
(VERB ask '(ACTION (ask)) -f '(command transitive))
(VERB tell '(ACTION (tell)) -f '(command transitive))
(VERB put '“(ACTION (put))
   -f ’(transitive L))

(VERB move '“(ACTION (move))
   -f ’(transitive intransitive LDS LS LD L D S NoMod))

(VERB turn '“(ACTION (turn))
   -f ’(transitive intransitive LDS LS LD L D S NoMod))

(VERB rotate '“(ACTION (rotate))
   -f ’(transitive intransitive LDS LS LD L D S NoMod))

(VERB open '“(ACTION (open))
   -f ’(transitive LDS LS LD L D S NoMod))

(VERB close '“(ACTION (close))
   -f ’(transitive LDS LS LD L D S NoMod))

(VERB (turn up) '“(ACTION (turn up))
   -f ’(transitive LS L S NoMod))

(VERB (turn down) '“(ACTION (turn down))
   -f ’(transitive LS L S NoMod))

(VERB (turn on) '“(ACTION (turn on))
   -f ’(transitive LS L S NoMod))

(VERB (turn off) '“(ACTION (turn off))
   -f ’(transitive LS L S NoMod))

(VERB (turn out) '“(ACTION (turn out))
   -f ’(transitive L NoMod))

(VERB (turn over) '“(ACTION (turn over))
   -f ’(transitive intransitive S NoMod))

(VERB (put on) '“(ACTION (put on))
   -f ’(transitive NoMod))

(VERB (put aside) '“(ACTION (put aside))
   -f ’(transitive NoMod))

(VERB (put down) '“(ACTION (put down))
   -f ’(transitive NoMod))

(VERB (put out) '“(ACTION (put out))
   -f ’(transitive L NoMod))

(VERB (open up) '“(ACTION (open up))
   -f ’(transitive LS L S NoMod))

(VERB (close down) '“(ACTION (close down))
   -f ’(transitive LS L S NoMod))

;---------------------------------------------------------------

(S/GOAL (VERB) ‘(.,#>1)
   -if (feat 1 ‘intransitive))
(S/GOAL (VERB CONCEPTUAL-CATEGORY) (if (listp (car #>2))
    '(.#>1 ,#>2)
    '(.#>1 ,#>2))
   -if (feat 1 'intrmsitive))

(S/GOAL (VERB OBJ) '(.#>1 ,#>2)
   -if (AND (feat 1 'transitive) (feat 1 'NoMod)))

(S/GOAL (VERB OBJ CONCEPTUAL-CATEGORY) (if (listp (car #>3))
    '(.#>1 ,#>2 ,#>3)
    '(.#>1 ,#>2 ,#>3))
   -if (AND (feat 1 'transitive) (OR (AND (feat 1 'LDS)
      (feat 3 'L) (feat 3 'D) (feat 3 'S))
      (AND (feat 1 'LS)
      (feat 3 'L) (feat 3 'S) (NOT (feat 3 'D))))
      (AND (feat 1 'DS)
      (feat 3 'D) (feat 3 'S) (NOT (feat 3 'L))))
      (AND (feat 1 'LD)
      (feat 3 'L) (feat 3 'D) (NOT (feat 3 'S))))
      (AND (feat 1 'L)
      (feat 3 'L) (NOR (feat 3 'D) (feat 3 'S))))
      (AND (feat 1 'D)
      (feat 3 'D) (NOR (feat 3 'L) (feat 3 'S))))
      (AND (feat 1 'S)
      (feat 3 'S) (NOR (feat 3 'L) (feat 3 'D)))))
)

;; intransitivity does not want a stative!
(S/GOAL (VERB ADV) '(.#>1 ,#>2)
   -if (AND (feat 1 'intrmsitive) (NOT (feat 2 'S))))

(S/GOAL (VERB CONCEPTUAL-CATEGORY ADV) (if (listp (car #>2))
    '(.#>1 ,#>2 ,#>3)
    '(.#>1 ,#>2 ,#>3))
   -if (AND (feat 1 'intrmsitive) (NOT (feat 3 'S))))

(S/GOAL (VERB ADV CONCEPTUAL-CATEGORY) (if (listp (car #>3))
    '(.#>1 ,#>2 ,#>3)
    '(.#>1 ,#>2 ,#>3))
   -if (AND (feat 1 'intrmsitive) (NOT (feat 3 'S))))

(S/GOAL (VERB OBJ ADV) '(.#>1 ,#>2 ,#>3)
   -if (feat 1 'transitive))
(S/GOAL (VERB OBJ CONCEPTUAL-CATEGORY ADV) (if (listp (car #>3))
  '(.#>1 ,#>2 ,#>3 ,#>4)
  '(.#>1 ,#>2 ,#>3 ,#>4))
-if (AND (feat 1 'transitive)
  (OR (AND (feat 1 'LDS)
    (feat 3 'L) (feat 3 'D) (feat 3 'S))
    (AND (feat 1 'LS)
      (feat 3 'L) (feat 3 'S) (NOT (feat 3 'D)))
    (AND (feat 1 'DS)
      (feat 3 'D) (feat 3 'S) (NOT (feat 3 'L)))
    (AND (feat 1 'LD)
      (feat 3 'L) (feat 3 'D) (NOT (feat 3 'S)))
    (AND (feat 1 'L)
      (feat 3 'L) (NOR (feat 3 'D) (feat 3 'S)))
    (AND (feat 1 'D)
      (feat 3 'D) (NOR (feat 3 'L) (feat 3 'S)))
    (AND (feat 1 'S)
      (feat 3 'S) (NOR (feat 3 'L) (feat 3 'D))))))

(S/GOAL (VERB OBJ ADV CONCEPTUAL-CATEGORY) (if (listp (car #>4))
  '(.#>1 ,#>2 ,#>3 ,#>4)
  '(.#>1 ,#>2 ,#>3 ,#>4))
-if (AND (feat 1 'transitive)
  (OR (AND (feat 1 'LDS)
    (feat 3 'L) (feat 3 'D) (feat 3 'S))
    (AND (feat 1 'LS)
      (feat 3 'L) (feat 3 'S) (NOT (feat 3 'D)))
    (AND (feat 1 'DS)
      (feat 3 'D) (feat 3 'S) (NOT (feat 3 'L)))
    (AND (feat 1 'LD)
      (feat 3 'L) (feat 3 'D) (NOT (feat 3 'S)))
    (AND (feat 1 'L)
      (feat 3 'L) (NOR (feat 3 'D) (feat 3 'S)))
    (AND (feat 1 'D)
      (feat 3 'D) (NOR (feat 3 'L) (feat 3 'S)))
    (AND (feat 1 'S)
      (feat 3 'S) (NOR (feat 3 'L) (feat 3 'D))))))

;------------------------------------------------------------------------

(CONCEPTUAL-CATEGORY (TEMPORAL) #>1 -f '(T))
(CONCEPTUAL-CATEGORY (LOCATIVE) #>1 -f '(L))
(CONCEPTUAL-CATEGORY (DIRECTIVE) #>1 -f '(D))
(CONCEPTUAL-CATEGORY (STATIVE) #>1 -f '(S))
(CONCEPTUAL-CATEGORY (CONCEPTUAL-CATEGORY CONCEPTUAL-CATEGORIES)
  (if (listp (car #>1))
    '(.#>1 ,#>2)
    '(.#>1 ,#>2))))

(CONCEPTUAL-CATEGORIES (CONCEPTUAL-CATEGORY) (if (listp (car #>1))
  #>1
  '(#>1)))

;------------------------------------------------------------------------
(DIRECTIVE (to OBJ) '(DIRECTIVE (DESTINATION ,0(cdr #2)))
  -if (OR (feat 2 'D) (feat 2 'L))
(DIRECTIVE (toward OBJ) '(DIRECTIVE (DESTINATION ,0(cdr #2)))
  -if (OR (feat 2 'D) (feat 2 'L))
(DIRECTIVE (from OBJ) '(DIRECTIVE (SOURCE ,0(cdr #2)))
  -if (OR (feat 2 'D) (feat 2 'L))
(LOCATIVE (on OBJ) '(LOCATIVE ,0(cdr #2))
  -if (feat 2 'L))
(LOCATIVE (upon OBJ) '(LOCATIVE ,0(cdr #2))
  -if (feat 2 'L))

(STATIVE (from A-STATE) '(STATIVE (INITIAL ,#2)))
(STATIVE (from state A-STATE) '(STATIVE (INITIAL ,#3)))
(STATIVE (up from A-STATE) '(STATIVE (INITIAL ,#3)))
(STATIVE (up from state A-STATE) '(STATIVE (INITIAL ,#4)))
(STATIVE (down from A-STATE) '(STATIVE (INITIAL ,#3)))
(STATIVE (down from state A-STATE) '(STATIVE (INITIAL ,#4)))

(STATIVE (to A-STATE) '(STATIVE (FINAL ,#2)))
(STATIVE (to state A-STATE) '(STATIVE (FINAL ,#3)))
(STATIVE (up to A-STATE) '(STATIVE (FINAL ,#3)))
(STATIVE (up to state A-STATE) '(STATIVE (FINAL ,#4)))
(STATIVE (down to A-STATE) '(STATIVE (FINAL ,#3)))
(STATIVE (down to state A-STATE) '(STATIVE (FINAL ,#4)))
(STATIVE (toward A-STATE) '(STATIVE (FINAL ,#2)))
(STATIVE (toward state A-STATE) '(STATIVE (FINAL ,#3)))
(STATIVE (up toward A-STATE) '(STATIVE (FINAL ,#3)))
(STATIVE (up toward state A-STATE) '(STATIVE (FINAL ,#4)))
(STATIVE (down toward A-STATE) '(STATIVE (FINAL ,#3)))
(STATIVE (down toward state A-STATE) '(STATIVE (FINAL ,#4)))

(A-STATE (<word>) '(,#1)
  -if (AND *object* (member #1 (struc-value *object* 'states))
  -f '(stative))

(S (S/GOAL) '(((INSTRUMENT ,#1))))
(S (VERB OBJ to S/GOAL) '((,#1)
  ;; Verb
  (AGENT (YOU)) ;; The User
  ,#2 ;; Agent
  ;; The command follows the transformational
  ;; grammar rules, object-controlled equiv.
  ;; The object of the command becomes
  ;; the agent in the instrumental.
  (INSTRUMENT
    ,(car #4)
    (AGENT ,0(cdr #2))
    ,0(cdr #4)))
  -if (AND (feat 1 'command)
    (feat 2 'A)
    (setq *agent* (caadr #2))))

APPENDIX E
Functions Used For The Interface System

Utilities

(DEFUN concat-symbols
   "Changes the list to a symbol."
   (. . .))

(DEFMACRO struc-value
   "Returns a value from a defstruct, using variables."
   (. . .))

(DEFMACRO struc-valuep
   "Determines if the value is a member of the object's slot value."
   (. . .))

(DEFMACRO movement-type
   "Determines the type of movement of the control."
   (. . .))

(DEFUN flatten
   "Reduces the list to a simple list of atoms."
   (. . .))

(DEFUN bell
   "Produces the sound of the bell"
   (. . .))

(DEFMACRO the-last
   "The last item in a list."
   (. . .))

(DEFMACRO and/or-list
   "Reformats a list into an and/or string"
   (. . .))

(DEFUN print-condensed
   "Prints the list in a condensed form, i.e. only one space."
   (. . .))

(DEFMACRO p
   "Prints a sequence of args."
   (. . .))
(DEFMACRO error-msg
   "A copy of P with a throw function to the talk loop."
   ( . . . ))

(DEFMACRO nor
   "(Not (or...))"
   ( . . . ))

(DEFMACRO abbrev-form
   "Searches a list for the full name of an abbreviated word."
   ( . . . ))

(DEFMACRO remove-one
   "Removes the first item that matches a member of the sequence."
   ( . . . ))

(DEFMACRO change-input-to-list
   "Takes a simple string as input and changes it to a list."
   ( . . . ))

(DEFMACRO no-input
   "A predicate determining the equivalent of a NIL output"
   ( . . . ))

(DEFMACRO defined-object?
   "A predicate determining if the object is defined."
   ( . . . ))

(DEFMACRO fixed-object?
   "A predicate determining if the object can be physically moved."
   ( . . . ))

(DEFMACRO endless-list?
   "A predicate determining if the movement of the control is endless."
   ( . . . ))

The Interface Loop

(DEFUN talk
   "Interface with the user."
   ( . . . ))

(DEFUN talk1
   "The interface loop."
   ( . . . ))

Interpretation of messages

(DEFUN interpret-messages
   "Interprets the messages in the object structures"
   ( . . . ))
Send messages to the mock simulator

(DEFUN uc-print
  "Prints the list in a condensed upper-case form, i.e. only one space."
  (. . .))

From Winston and Horn [26, Chapter 12]
- (DEFMACRO rest-of-phrase
  (. . .))
- (DEFMACRO pattern-ind
  (. . .))
- (DEFMACRO pattern-var
  (. . .))
- (DEFMACRO shove-gr
  (. . .))
- (DEFMACRO pull-value
  (. . .))
- (DEFUN shove-pl
  (. . .))
- (DEFMACRO restriction-ind
  (. . .))
- (DEFMACRO restriction-pred
  (. . .))
- (DEFUN restrictp
  "Determines if the restriction predicate is correct."
  (. . .))
- (DEFUN message-match
  "Matches datum to a pattern."
  (. . .))

(DEFUN apply-message
  "Changes the current value of the object."
  (. . .))

(DEFUN acknowledge-message
  "Written acknowledgement that the message was accepted."
  (. . .))

(DEFUN send
  "Sends a message if it matches one of the patterns."
  (. . .))
Determine specific object information

(DEFUN determine-moving-object-mapping
  "Determine the 3 space mapping of the object."
  ( . . ))

(DEFUN determine-texture-directional
  "Determines the object’s 3 space mapping."
  ( . . ))

(DEFUN determine-subset
  "Determines a specific subset of the object’s states."
  ( . . ))

(DEFUN general-directional?
  "Determines if the directive word is valid for the object."
  ( . . ))

(DEFUN determine-state-error!
  "Determines the state error
    and returns the user to the 'talk' level"
  ( . . ))

(DEFUN determine-general-directive-state
  "Determines the state from the general word."
  ( . . ))

(DEFUN determine-raised-state
  "Determines the state above the current state."
  ( . . ))

(DEFUN determine-lowered-state
  "Determines the state below the current state."
  ( . . ))

(DEFUN determine-state
  "Determine the final state of the object."
  ( . . ))

(DEFUN determine-state-using-stative
  "Determine the final state of the object using the stative information."
  ( . . ))

(DEFUN determine-state-using-directive
  "Determine the final state of the object using the directive information."
  ( . . ))

(DEFUN determine-state-using-limited-states
  "Determine the final state of the object using limited states."
  ( . . ))
(DEFUN query-user
   "Queries the user as to which member of a list is meant."
   (...) )

---

Resolve any questions received from the parsed input

(DEFUN resolve-agent
   "Determines the exact agent, whether implied or stated."
   (...) )

(DEFUN immediately-obvious-locative
   "Only returns a superficially obvious locative."
   (...) )

(DEFUN resolve-isa-object
   "Queries the user for a more specific object."
   (...) )

(DEFUN resolve-object
   "Determines the exact object, whether implied or stated."
   (...) )

(DEFUN resolve-locative
   "Determines the exact locative of the object"
   (...) )

(DEFUN resolve-s-initial
   "Resolves any problems if the entered initial state is incorrect."
   (...) )

(DEFUN resolve-direction
   "Resolves a directive list into a single atom directive."
   (...) )

(DEFMACRO assoc-item
   "Pulls out the object from the key pair."
   (...) )

(DEFMACRO assoc-item-d
   "Pulls out the object and deletes the key pair from the list."
   (...) )

(DEFUN apply-constituents
   "Evaluates the constituents of partial case frame as per the verb"
   (...) )

---

Apply the proper primitive to the data

(DEFUN apply-primitive
   "Sends each list to the primitive function."
   (...) )
Primitive analysis

(DEFUN determine-pmove-direction
   "Choose the proper direction for the movement."
   (. . .))

(DEFUN add-pmove-measurement
   "Add up the number of measurements followed in the path."
   (. . .))

(DEFUN limit-pmove-path
   "Reduce the path list s.t. it starts at an endpoint of the chosen path."
   (. . .))

(DEFUN determine-pmove-movement
   "Determine the movement for the best path."
   (. . .))

(DEFUN best-pmove-path
   "The best path to be used in a PMove action."
   (. . .))

(DEFUN PMove
   "The primitive verb PMove."
   (. . .))

(DEFUN PLook
   "The primitive verb PLook."
   (. . .))

A description of the objects

(DEFUN describe-object
   "Describe's the attributes of an object."
   (. . .))

(DEFUN describe-control
   "Describes a control."
   (. . .))

(DEFUN describe-indicator
   "Describes an indicator."
   (. . .))

(DEFUN describe-isa
   "Describes an isa parent."
   (. . .))
(DEFUN describe-agent
  "Describes an agent."
  (...))

---

To move

(DEFUN move-verb
  "The action verb, move."
  (...))

---

To turn

(DEFUN turn-verb
  "The action verb, turn."
  (...))

(DEFUN turn-up-verb
  "The action verb, turn up."
  (...))

(DEFUN turn-down-verb
  "The action verb, turn down."
  (...))

(DEFUN turn-on-verb
  "The action verb, turn on."
  (...))

(DEFUN turn-off-verb
  "The action verb, turn off."
  (...))

(DEFUN turn-out-verb
  "The action verb, turn out."
  (...))

(DEFUN turn-over-verb
  "The action verb, turn over."
  (...))

---

To rotate

(DEFUN rotate-verb
  "The action verb, rotate."
  (...))
To put

(DEFUN put-verb
   "The action verb, put."
   (. . .))

(DEFUN put-ca-verb
   "The action verb, put on."
   (. . .))

(DEFUN put-aside-verb
   "The action verb, put aside."
   (. . .))

(DEFUN put-down-verb
   "The action verb, put down."
   (. . .))

(DEFUN put-out-verb
   "The action verb, put out."
   (. . .))

To open

(DEFUN open-verb
   "The action verb, open."
   (. . .))

(DEFUN open-up-verb
   "The action verb, open up."
   (. . .))

To close

(DEFUN close-verb
   "The action verb, close."
   (. . .))

(DEFUN close-down-verb
   "The action verb, close down."
   (. . .))

Replaces an indicator with its connected control
(DEFUN specific-indicator-connections
    "Returns the specific connections within the indicator."
    ( . . )
)

(DEFUN get-indicator-connections
    "Returns a formatted list of all specified connections."
    ( . . )
)

Utility functions for syntax rules

(DEFUN official-name
    "Finds the agent's official name, i.e., its structure name."
    ( . . )
)

(DEFUN official-name1
    "Checks the list for a match."
    ( . . )
)

(DEFUN search-for-gender
    "Creates a list of all the agent members which match the gender."
    ( . . )
)

(DEFUN find-the-errors
    "Finds all words that are not defined."
    ( . . )
)

The object structures

(DEFSTRUCT agent
    "An agent object."
    ( . . )
)

(DEFSTRUCT isa-parent
    "A list of the members (children) of the ISA-PARENT object."
    ( . . )
)

(DEFSTRUCT object
    "The global type for all objects."
    ( . . )
)

(DEFSTRUCT (indicator (:include object))
    "An indicator type of object."
    ( . . )
)

(DEFSTRUCT (control (:include object))
    "A control type of object"
    ( . . )
)
Create agent structures

(DEFUN delete-agent-copies
   "Deletes a previous agent property list, if any.
    for a specific agent."
   (. . .))

(DEFMACRO create-agent
   "Creates an agent defstruct and adds its names to the agent list."
   (. . .))