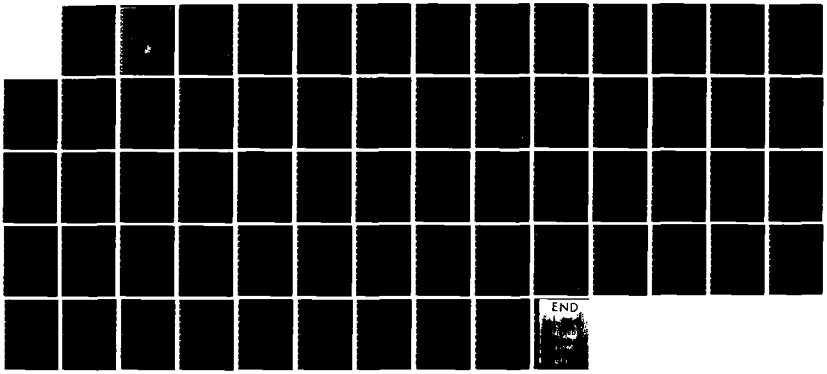
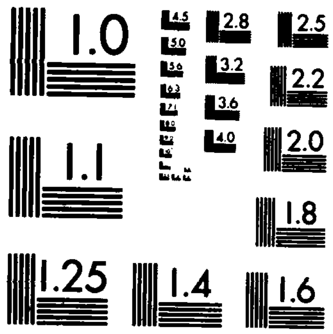
AD-A136 944 THE BEHAVIORAL DEMONSTRATOR: A REQUIREMENTS SPECIFICATION EXECUTOR(U) VIRGINIA POLYTECHNIC INST AND STATE UNIV BLACKSBURG COMPUTER S.. J E CALLAN MAY 83 1/1

UNCLASSIFIED CSIE-83-14 N00014-81-K-0143 F/G 9/2 NL




1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

(1)

THE BEHAVIORAL DEMONSTRATOR: A REQUIREMENTS SPECIFICATION EXECUTOR

JAMES E. CALLAN III

Virginia Polytechnic Institute
and State University
Computer Science
Industrial Engineering and Operations Research
BLACKSBURG, VIRGINIA 24061

CSIE-83-14 MAY 1983 1

THE BEHAVIORAL DEMONSTRATOR: A REQUIREMENTS SPECIFICATION EXECUTOR

JAMES E. CALLAN III

TECHNICAL REPORT

Prepared for
Engineering Psychology Group, Office of Naval Research
ONR Contract Number N00014-81-K-0143
Work Unit Number NR SRO-101

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CSIE-83-14	2. GOVT ACCESSION NO. AD-A136944	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE BEHAVIORAL DEMONSTRATOR: A REQUIREMENTS SPECIFICATION EXECUTOR		5. TYPE OF REPORT & PERIOD COVERED Technical 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) James E. Callan III		8. CONTRACT OR GRANT NUMBER(s) N00014-81-K-0143
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Virginia Polytechnic Institute & State University Blacksburg, Virginia 24061		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61153N42; RR04209; RR0420901; NR SRO-101
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research, Code 442 800 North Quincy Street Arlington, VA 22217		12. REPORT DATE May 1983 13. NUMBER OF PAGES 62
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

SBSDM, supervisory-structure, supervisor-cell, supervised-flow-diagram (SFD) dialogue-function, computational-function, requirements-executor, Behavioral-Demonstrator (BD), Transactional Database (TDB), Dialogue-Management-System (DMS), Behavioral-Translator, SFD-compiler, Human-computer dialogue, Human-factorable, Dialogue-author.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents a design for tools which aid in the requirements verification of computer systems. These tools use a very high level graphical requirements specification language and a system development methodology for human-computer systems. The report moves from an abstract design to actual implementation and uses a sample application system throughout for illustration.

ACKNOWLEDGEMENT

The author wishes to express appreciation to Shuhab Ahmed, Roger Ehrich, Rex Hartson, Pardeep Narang, and Tamer Yunten for their help in preparing this report. This work was done under the supervision of H. Rex Hartson.

ABSTRACT

This report presents a design for tools which aid in the requirements verification of computer systems. These tools use a very high level graphical requirements specification language and a system development methodology for human-computer systems. The report moves from an abstract design to actual implementation and uses a sample application system throughout for illustration.

KEYWORDS

SBSDM, supervisory-structure, supervisor-cell, supervised-flow-diagram (SFD), dialogue-function, computational-function, requirements-executor, Behavioral-Demonstrator (BD), Transactional Database (TDB), Dialogue-Management-System (DMS), Behavioral-Translator, SFD-compiler, Human-computer dialogue, Human-factorable, Dialogue-author.

TABLE OF CONTENTS

INTRODUCTION

A movement has begun in the software engineering industry towards the development of tools which allow system designers to observe the logical flow of their system designs before implementation begins. Designers using such tools may see exactly how a system will behave at the earliest stages of development. Such tools will be used to demonstrate the completeness, or lack thereof, of system requirements specifications. These specifications are stored in a "requirements structure", a formal format completely describing both the data flow and control flow, and the internal functional descriptions of the functional components of systems.

As specifications for a system are developed, they are added to the "requirements structures" for the system. Later, the "requirements specifications" may be executed by these tools in a manner similarly used by interpreters. In effect, the system may be "taken for a spin" to determine how it behaves. If properly designed, these tools can be used for structured walk-throughs of the completed system. In fact, an environment utilizing these tools and a structured design methodology could improve productivity and system reliability by ten-fold.

At Virginia Tech research on such an environment is already underway. The environment is called the Dialogue Management System. This paper presents a design for the tools and environment described above as used in the Dialogue Management System.

The Dialogue Management System (DMS) is being developed to improve human-computer dialogues. It consists of three major components: a dialogue development facility, a computational development facility, and a structured development methodology

[JOHND82a]. The Behavioral Demonstrator (BD) is perhaps the most important tool included in DMS. The BD aids in the functional decomposition and verification of an application system. Users of the BD may execute requirements specifications for an application system even when the system is at the most abstract level of development. In one respect, the BD can be viewed as a development environment supporting the other tools in DMS in a manner consistent with the structured-development methodology presented in [YUNTT82]. This report presents a basic design for implementation of the BD and concludes with a description of a sample application-system.

1.0 THE BEHAVIORAL DEMONSTRATOR

A formal language for describing the "requirements structure" for application systems is basic in the design of any Behavioral Demonstrator (BD). In that formal language three ingredients are needed:

1. A means for describing functional components of systems in terms of modules, sets, or entities.
2. A means for describing the requirements specifications and data flow between the functional components.
3. A group of control structures describing the flow of control between the functional components.

Languages exhibiting these features can be high-level procedural languages, menu-driven interpreters, or even graphical languages such as the Supervised Flow Diagram Language chosen for use in DMS.

The Supervised Flow Diagram Language is presented [YUNTT83] as a high-level requirements specification language supporting the Supervisor-Based System-Development Methodology (SBSDM). The SBSDM was created to improve software design and productivity by human-factoring human-computer dialogues. In this methodology, human-computer dialogues are constructed separately from the system's computational components [JOHND82b]. Application programme-s construct computational components; whereas, "dialogue authors" d ign human-factored human-computer dialogues [JOHND82a]. SBSDM is the -tructured-development methodology for which DMS is being developed.

During requirements specification, Supervised Flow Diagrams (SFDs) are constructed using an SFD editor. Once created, the graphical nodes in the SFDs may be executed by the BD to verify that system requirements are being met. The BD allows a user to walk through a system at its most abstract level of development and determine how a system behaves. In this way, the BD may be viewed as both a requirements "verifier" and a system behavioral demonstrator. As each worker node in the SFDs is expanded (i.e. written by either an application programmer or a dialogue author), the Behavioral Translator, a tool similar to the BD, may be used to execute the finished module. Hence, a user can use the BD as a structured walk-through supervisor during all phases of system development. After all primitive modules have been expanded in an SFD, another BD-like tool compiles the SFD "program", generating the finished system. Viewed this way, the BD may be used as a development framework supporting the SBSDM.

1.1 The SFD Language

At its most abstract form of development, a system will consist of little more than supervisory-cells structured in a hierarchical representation consistent with the SBSDM. Figure 1.1.1 illustrates this hierarchical representation and the corresponding SFD structures.

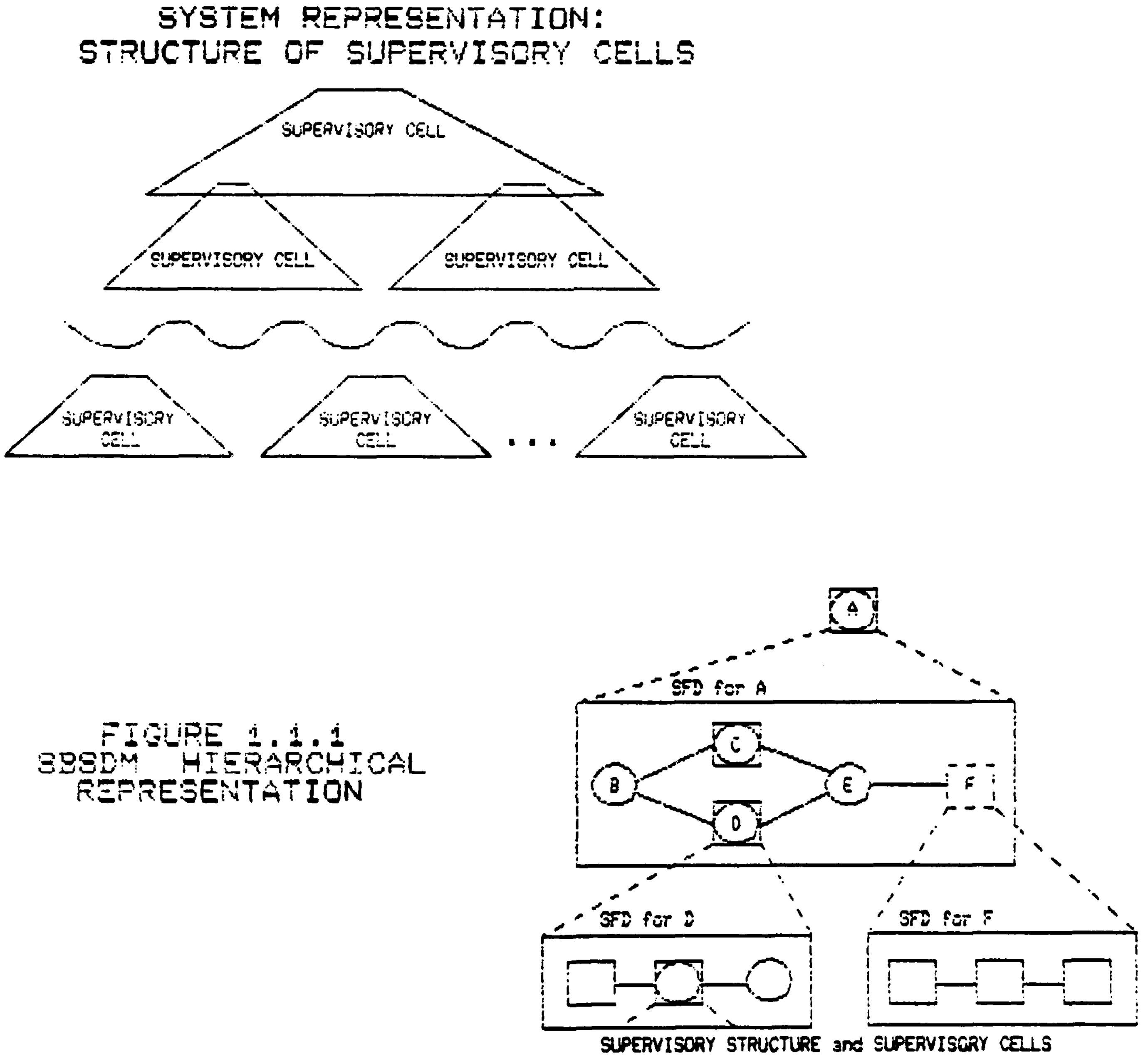


FIGURE 1.1.1
SBSDM HIERARCHICAL
REPRESENTATION

Wherein each supervisory-cell defines "what" is to be done in a given function, the corresponding expansion of a supervisor-cell defines "how" the function is to be performed [YUNTT83]. This expansion is known as the supervised flow diagram corresponding to its supervisory-cell. As depicted in Figure 1.1.1, it is quite possible for nodes in an SFD to be supervisory-cells for other SFD's. Consequently, each supervisory-cell may have two types of subfunctions in its SFD expansion: supervisory-functions and worker-functions. Where the expansion of a supervisory-function is a corresponding SFD, the expansion of a worker-function is the executable source code of a high-level language such as FORTRAN. Like FORTRAN, the SFD language has its own syntax and semantics. Graphic symbols comprise the SFD syntax, and the control constructs dictate its semantics. Figure 1.1.2 shows the graphical symbols included in the SFD language.

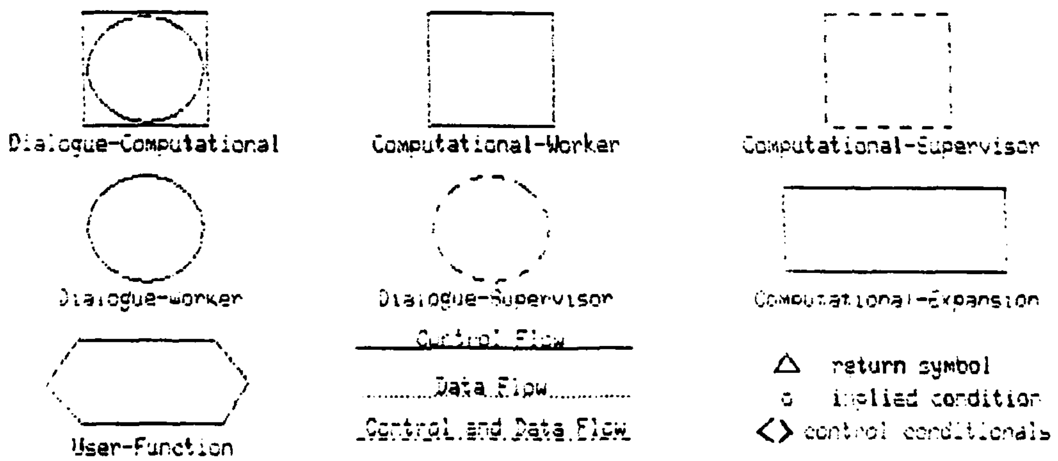


Figure 1.1.2 SFD Language Symbols

The Dialogue-Computational node denotes a supervisory-cell containing both Dialogue and Computational nodes in its SFD expansion. The dashed Dialogue and Computational nodes represent supervisory-cells containing only dialogue or computational functions in their respective SFDs. The rectangular node represents a computational function which the SFDs' author feels is too insignificant to be included in a separate Computational node. The hexagon represents a user function and may be included in an SFD for consistency and better requirements representation. Control Flow and Data Flow are represented by solid and dotted lines, respectively. Control conditionals are contained within < > symbols. The __ symbol represents control return from an SFD expansion to its supervisory-cell. Implied conditionals, to be explained later, are o symbols designating control-line intersections having special semantic actions. As with most high-level languages, four types of control constructs are defined. These are show in Figures 1.1.3-1.1.6. The remainder of this report presents implementation issues concerning BD development.

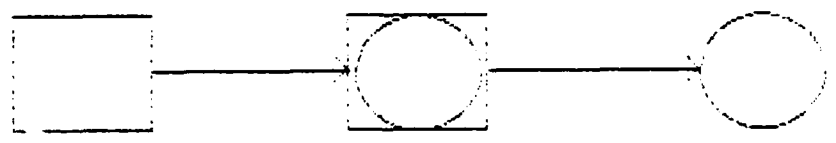


FIGURE 1.1.3 SEQUENTIAL CONTROL FLOW

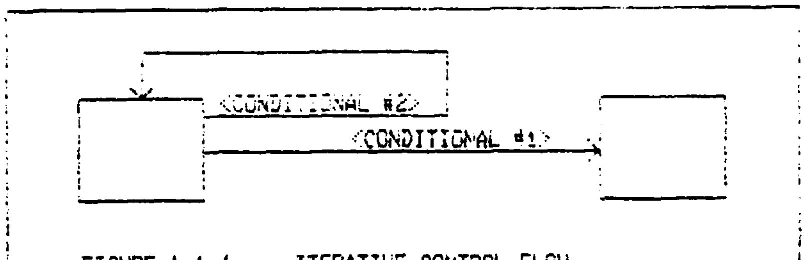


FIGURE 1.1.4 ITERATIVE CONTROL FLOW

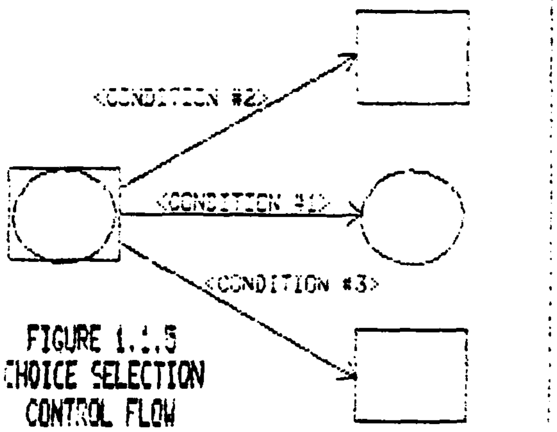


FIGURE 1.1.5
CHOICE SELECTION
CONTROL FLOW

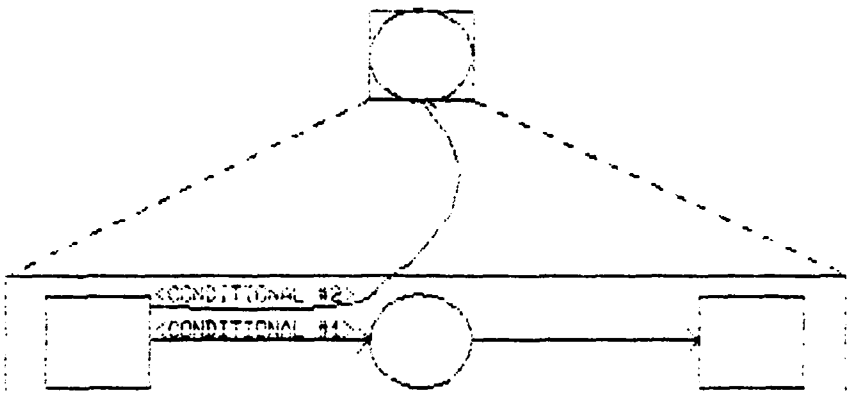


FIGURE 1.1.6 RECURSIVE CONTROL FLOW

1.2 BD Behavior and the User's Console

The user's console for the BD consists of two terminals: a graphical interaction terminal and a control-interaction terminal. The top supervisory-cell in the system and its SFD expansion are the items displayed by the BD when a behavioral demonstration begins. This is displayed on the graphical interaction terminal (GG), shown in Figure 1.2.1, with the cell blinking. The control-interaction terminal (TT) displays the functional description of the cell, data passed to and from the cell, and the control-flow options. Control options are selected via a screen directed, touch-panel keypad as illustrated in Figure 1.2.2.

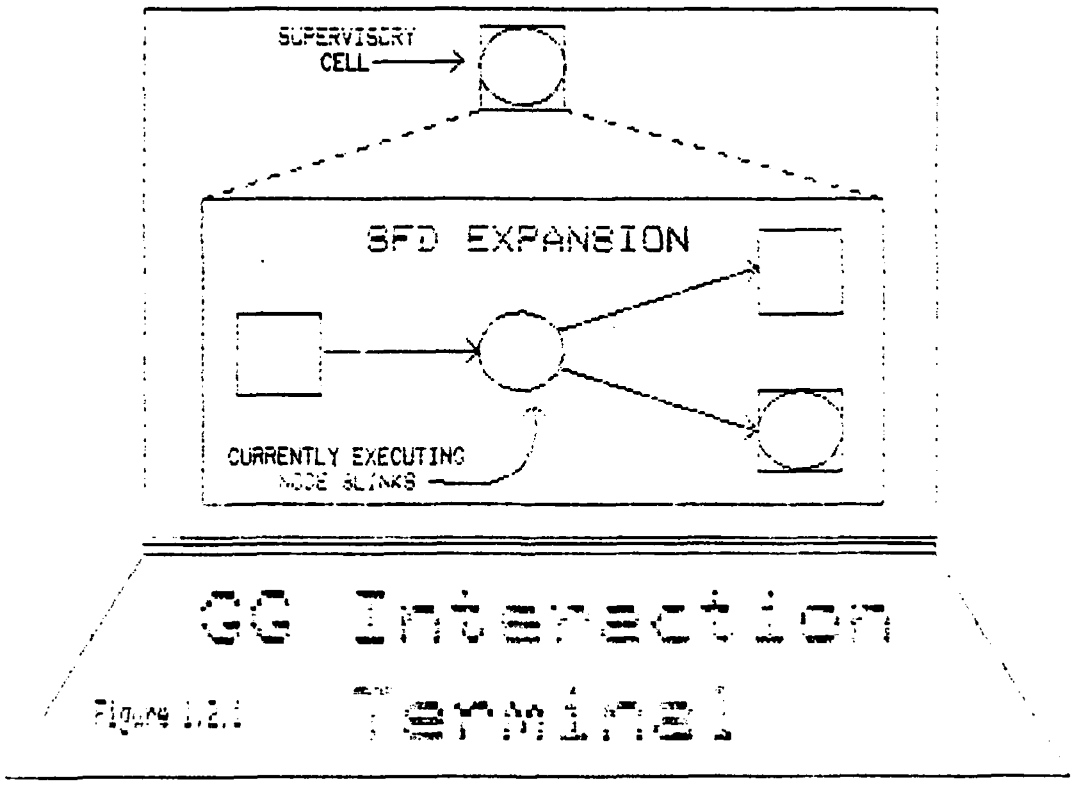


Figure 1.2.1

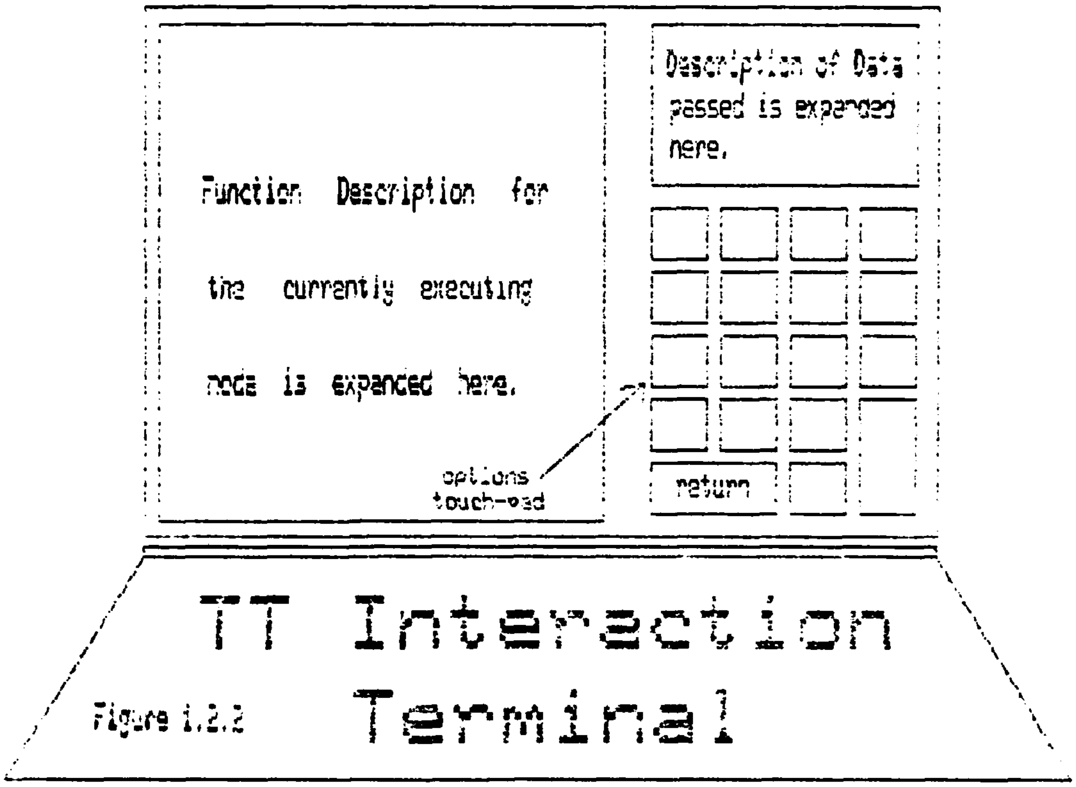

Description of Data passed is expanded here.
Function Description for the currently executing node is expanded here.
options touch-pad
return
TT Interaction Terminal

Figure 1.2.2

Control advances as control keys are pressed. Each node in the current SFD "executes" and control returns to the supervisory-cell.

As worker nodes for a system are expanded (i.e., coded), the Behavioral Translator may be used to execute them during Behavioral Demonstration. Finally, to conclude system development, the SFD Compiler is used to compile and link SFD modules for structured walk-throughs of completed sections of a system.

1.3 An Example BD

In order to illustrate the functions of the BD using the SFD language described in Section 1.1 we will use the following hypothetical, interactive airline-reservation system as an example. This example also appears in [JOHND82a] and [YUNTT83] and is repeated here with permission.

1.3.1 Requirements Specification

PERFORM-AIRLINE-RESERVATION: Manage airline reservation activities to serve the airline users.

REQUIREMENTS:

1. The system must check the agent-id everytime a person logs on and adjust agent-computer communication according to the agent's expertise level (e.g., novice or expert).
2. The system must perform the following functions, each of which can be invoked independently by a terminal function button.
   a. Reservation
   b. Get-payment
   c. Cancel-reservation
   d. Show-user's-reservation-status
3. The system has two major logical record structures. These records are shown in Figure 1.3.1.

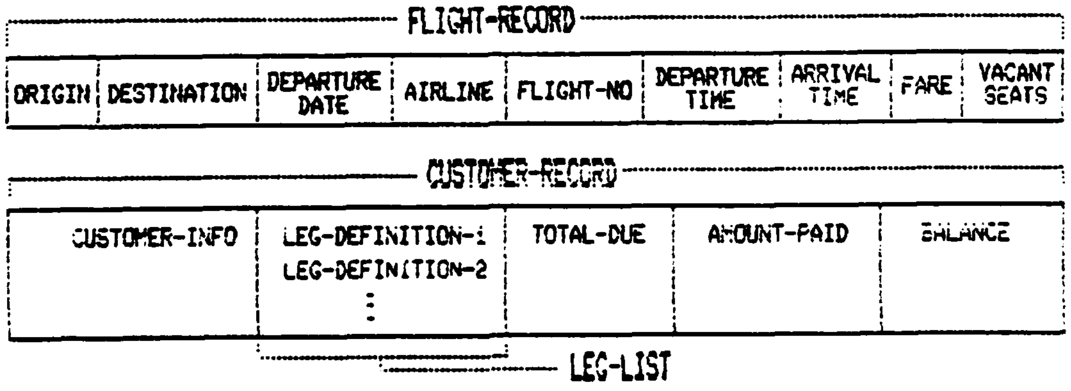


FIGURE 1.3.1 LOGICAL RECORD STRUCTURES

The SFD for PERFORM-AIRLINE-RESERVATION is shown in Figure 1.3.2.

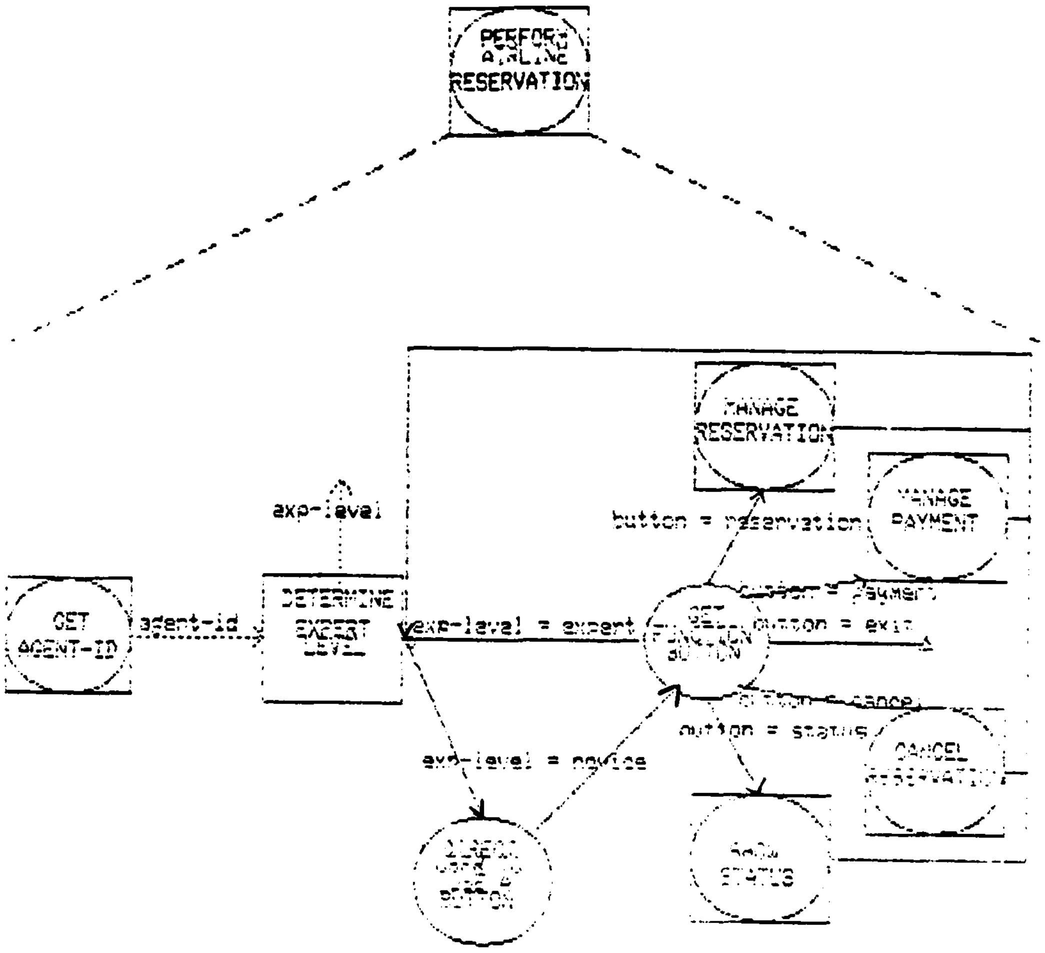


Figure 1.3.2 SFD for PERFORM-AIRLINE-RESERVATION

In the following, the requirements specification for "GET-AGENT-ID" of Figure 1.3.2 will be given.

GET-AGENT-ID: Gets a correct identification from the agent when the agent logs on.

REQUIREMENTS:

1. The system must have a list of valid agent-id's stored. Figure 1.3.3 expands the SFD for GET-AGENT-ID.

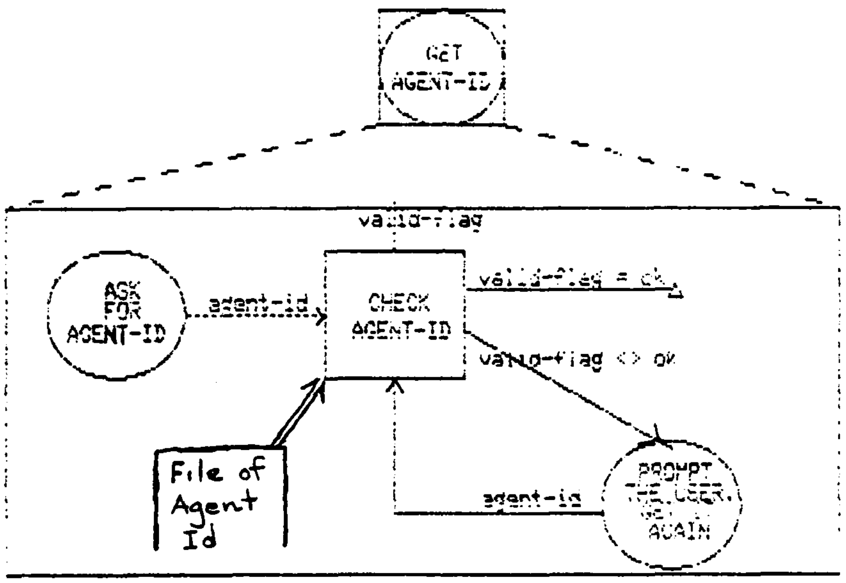


Figure 1.3.3 SFD for GET-AGENT-ID

Normally, all control and data lines on an SFD are administered by the supervisory-function with one exception: "Communication with the data bases is done within the logic of a supervised computational function." A supervisory-function only invokes a supervised-function to do the job. For instance, in Figure 1.3.3, the data flow of GET-AGENT-ID is shown by a double-line. The double-line indicates that this flow will actually happen within the logic of CHECK-AGENT-ID. It is cited on the SFD to show the existence of the file.

In a real application-system requirements specification, each of the other supervisory-functions would be further refined, providing that this information was available. For brevity in our example, we will assume that the remaining specifications are still quite abstract.

1.3.2 BD Operation

Once a user invokes the BD, the GG displays a window into an SFD structure while the TT expands the functional description, data descriptors, and control alternatives for the blinking node. Control selections are accepted from the TT video touch-panel, and control advances to the selected node. Figures 1.3.4-1.3.9 illustrate these advances in our airline example. Figure 1.3.8 shows how selection control is represented, and Figure 1.3.9 depicts iterative control-flow.

Although our example only illustrates sequence, choice, and iteration, the BD can "execute" complex conditionals and recursion with only minor modifications to the SFD "program." In Chapter 3 we shall detail the representation for each of the control constructs.

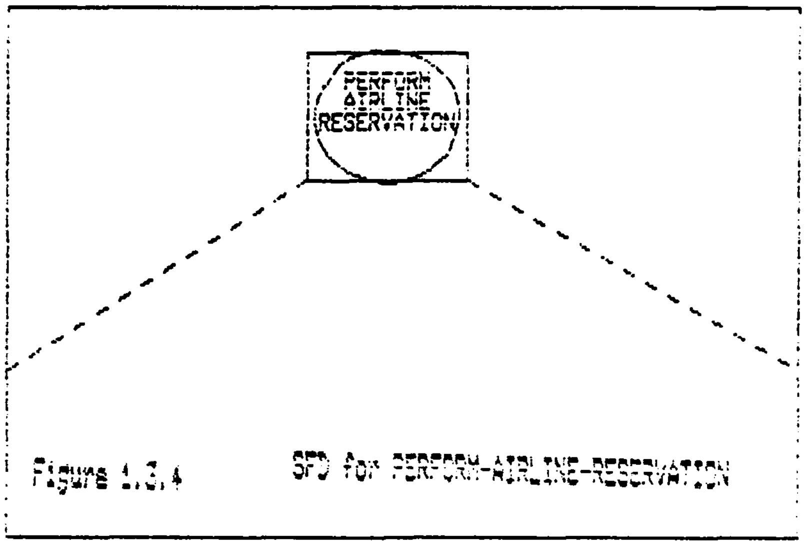
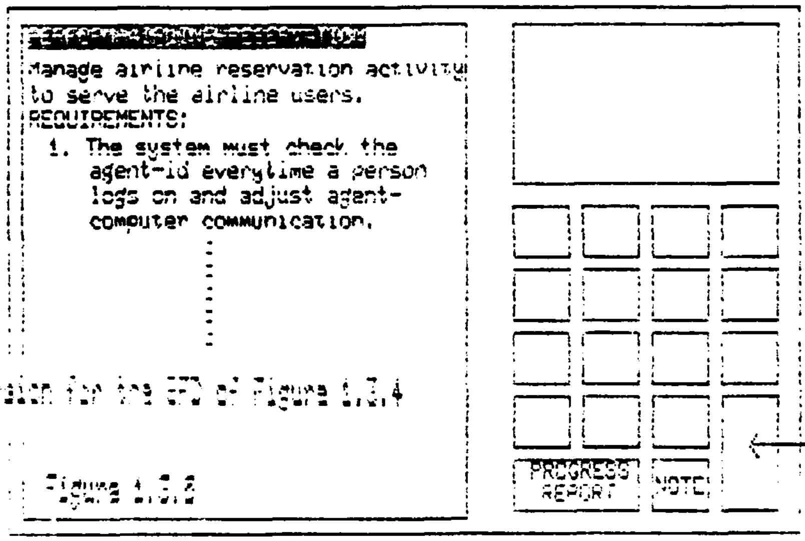


Figure 1.3.1 SFD for PERFORM-AIRLINE-RESERVATION



[illegible]

Manage airline reservation activity
to serve the airline users.
REQUIREMENTS:

1. The system must check the
   agent-id everytime a person
   logs on and adjust agent-
   computer communication.

Textual Expansion for the SFD of Figure 1.3.1

Figure 1.3.2

SFD Expansion for PERFORM-AIRLINE-RESERVATION

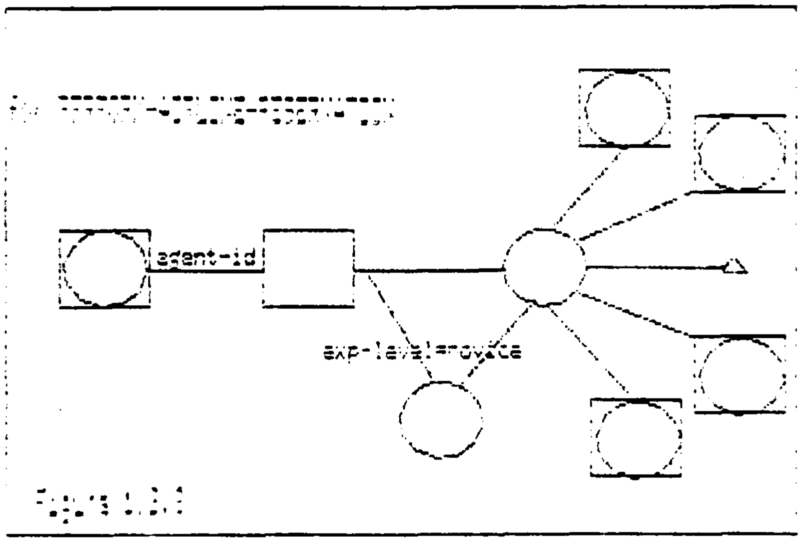
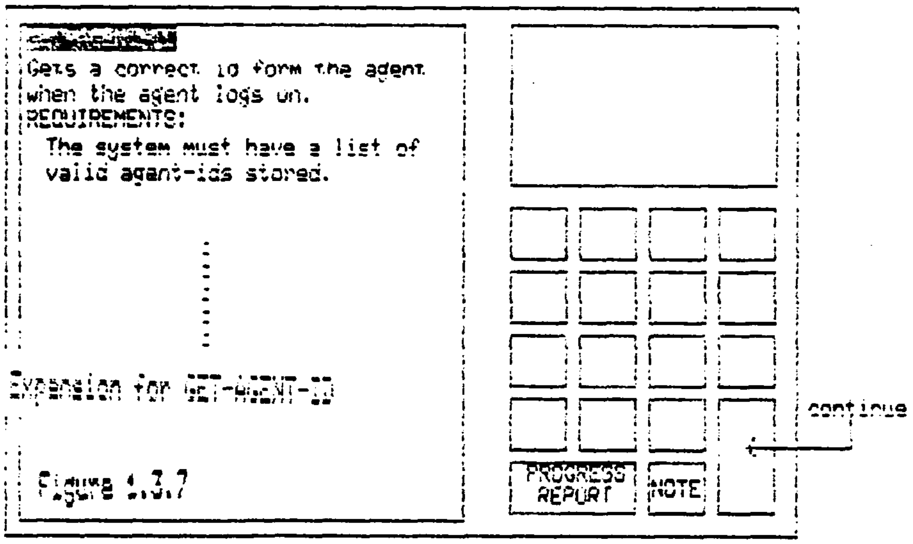


Figure 1.3.3


[illegible]

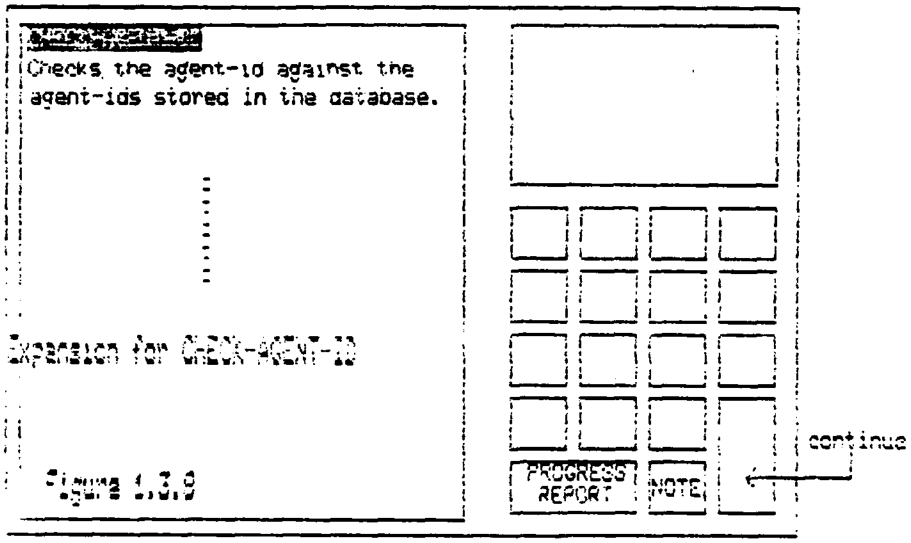
Gets a correct id form the agent when the agent logs on.

REQUIREMENTS:

The system must have a list of valid agent-ids stored.

Figure 1.3.7

Textual Expansion for GET-AGENT-ID


[illegible]

Checks the agent-id against the agent-ids stored in the database.

Figure 1.3.8

Textual Expansion for CHECK-AGENT-ID

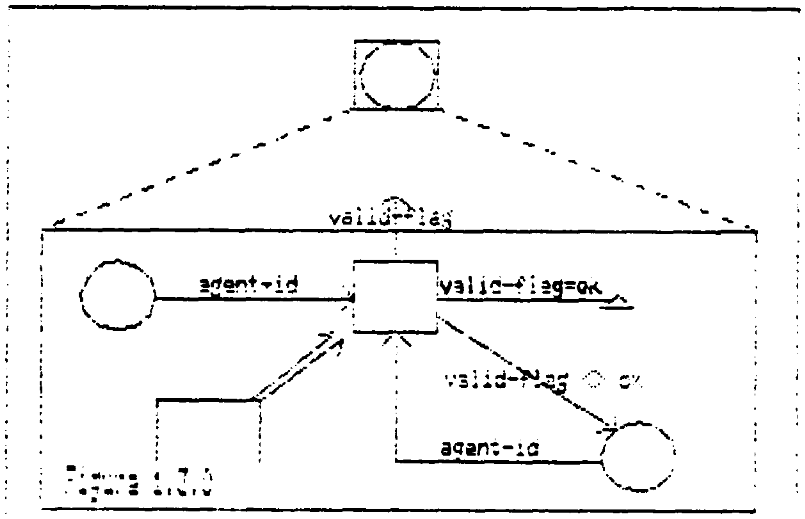


Figure 1.3.9

DFD Expansion for GET-AGENT-ID

2.0 DESIGN ISSUES

In Section 1.3 we gave an example illustrating how the BD operates. Here we present in detail all of the functions that the BD will include and refine the SFD language and BD database design.

2.1 BD Formal Requirements Specification

The BD will:

1. Demonstrate the flow of data and control in systems;
2. Provide a communication forum for design teams during the development process;
3. Serve to manage progress on each component of a system; and
4. Enforce the SBSDM.

To demonstrate the flow of data and control in systems, we will use the SFD language with more precise syntactic specifications. The SFD language syntax should visually reflect the function and "state" of each component in a system.

As users of the BD discover requirement inconsistencies and needs for requirement modifications, the BD provides an opportunity for those users to document their observations.

During the development of an application system, programmers and dialogue authors are working in parallel expanding worker-functions. Since this is a temporal process, some facility must be provided to keep track of the progress on each worker-function. The BD can serve not only to verify and maintain requirements, but also to continually inform users of system progress during behavioral demonstration.

The designs of the SFD language and BD enforce the methodology in which they are embedded.

2.2 Graphical Support Environment

The BD requires a supporting graphical editor for the creation and modification of SFD "programs." This editor provides all the features in an SFD structure and stores these structures in a transaction database. SFD structures will include the following:

- Graphical representation of supervisor-cells and their corresponding SFD expansions;
- Functional descriptions for each functional component;
- Progress reports for all functional components;
- Notes and comments for each functional component; and
- Control structure linkage descriptors and data descriptors.

2.2.1 SFD Graphical Standards

Since SFDs pictorially represent function as well as structure, SFD nodes will have color, size, and shape standards. Those standards are grouped by function in the following paragraphs.

CONTROL STRUCTURES:

In the SFD language, lines represent flow of control. There are three types of control flow lines:

-Control lines not labelled by data or conditionals are represented by solid,white lines.

-Control lines containing only data are represented by dash-dot, white lines.

-Control lines labelled with conditionals are presented as dotted, white lines.

This specification implies that no control lines exist which contain both data and a conditional. Note also that control lines may be arcs as well as straight lines.

The two remaining control constructs: the implied conditionals and the return branches are white o symbols and red △ symbols, respectively.

DATA FLOW STRUCTURES:

Data Flow is represented by white, dotted lines or arcs. All lines contain arrow heads that indicate the direction of both data flow and control flow.

D-C NODE STANDARDS:

D-C nodes are unshaded in the inner circle and unshaded on the outer square.

D-Cs are blue if the have a corresponding SFD expansion and red if they have not been expanded.

D-Cs should not exceed 5 cm in diameter.

COMPUTATIONAL WORKERS:

Computational-Worker functions are cyan, unshaded squares not exceeding 2.5 cm across.

COMPUTATIONAL SUPERVISORS:

Computational Supervisors are unshaded, dashed-line, cyan squares not exceeding 3.8 cm across. If a Supervisor's SFD has not been expanded, then it's node will be red.

DIALOGUE WORKER:

Dialogue-Worker nodes are yellow, unshaded, circles not exceeding 2.5 cm in diameter.

DIALOGUE SUPERVISORS:

Dialogue Supervisors are yellow, unshaded, dashed-line circles not exceeding 3.8 cm in diameter. If the Supervisor has no SFD expansion, then the node will be red.

USER FUNCTIONS:

User Functions are represented by green, shaded, solid-line hexagons not exceeding 5 cm across.

COMPUTATIONAL EXPANSIONS:

Computational-Expansion nodes are white, shaded, rectangles without size limitations. The explicit code expansion will be written in black text.

CONDITIONALS:

Conditionals label control lines and are green text contained within red < > symbols. The < > symbols are optional.

DATA DESCRIPTORS:

Data Descriptors are magenta text that label Data Flow or Control Flow lines.

2.2.2 Textual Standards

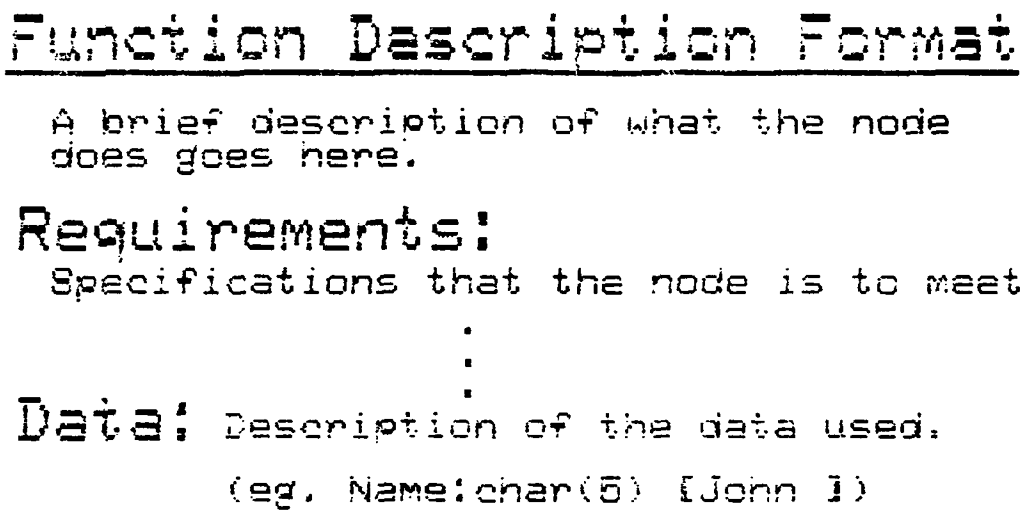
Standards for the text contained in the functional descriptions, progress reports, and notes made by the users are provided in the following paragraphs.

FUNCTIONAL DESCRIPTIONS:

Functional Descriptions of system components contain the following information:

- The description of the function this component will carry out;
- The data passed to and from this component; and
- A procedural specification containing implementation details, as they become available.

Figure 2.2.1 The format for Functional Descriptions.



Function Description Format

A brief description of what the node does goes here.

Requirements:

Specifications that the node is to meet.

.
.
.

Data: Description of the data used.

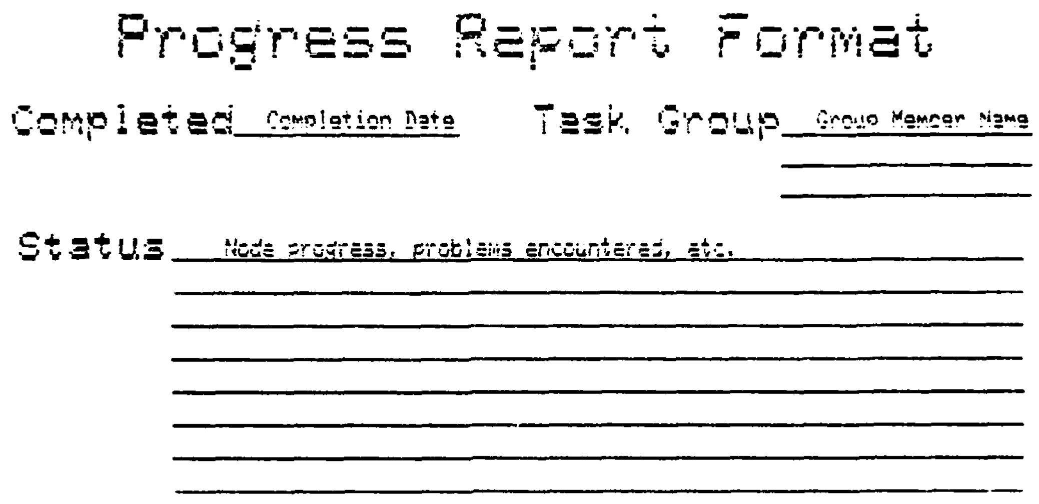
(eg. Name:char(5) [John])

PROGRESS REPORTS:

Progress Reports contain the following information:

- The date the expansion for this node was complete;
- The names of the persons responsible for this node's expansion (i.e., the Task Group for this node); and
- A status report expounding on the current progress and/or problems which are impeding progress.

Figure 2.2.2 The format for Progress Reports.



Progress Report Format

Completed Completion Date Task Group Group Member Name

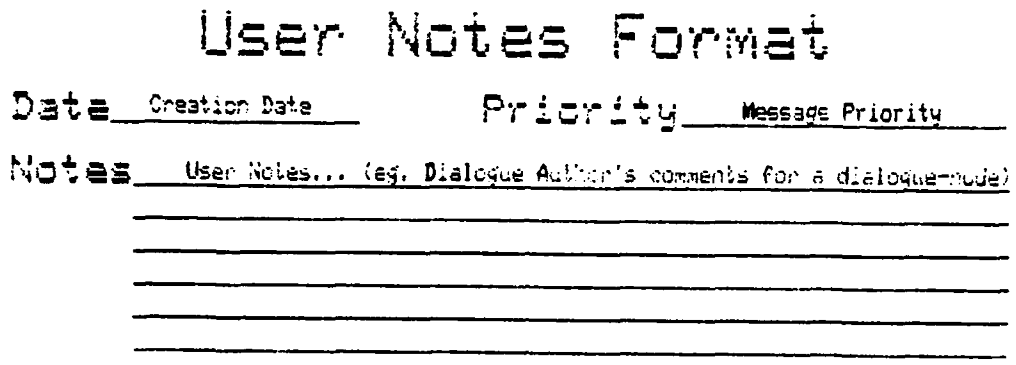
Status Node progress, problems encountered, etc.

USER NOTES:

User Notes contain the following information:

- The date the notes were made or become effective;
- The priority of the message (Will it critically affect progress?); and
- A section for notes which will be distributed to the "mailboxes" of the Task Group members for this node based cn the date and priority fields.

Figure 2.2.3 The format for User Notes.



User Notes Format

Date Creation Date
Priority Message Priority
Notes User Notes... (eg. Dialogue Author's comments for a dialogue-node)

2.3 Conceptual Data Model

A key issue in the design of the BD is the design of a database representation for both the graphical and textual content of SFD nodes. Listed below are the entities and attributes with which we are concerned.

2.3.1 Node Attributes

Node_Name:

Each node in an SFD expansion will have a uniquely identifying name. This name will be an abstract of the functional description of the node and will serve to label the node. It also is the relation's primary key.

Node_Type:

All nodes will be a member of the following set of node types:

Dialogue-Computational
Computational-Worker
Computational-Worker
Computational-Supervisor
Dialogue-Worker
Dialogue-Supervisor
Computational-Expansion
User-Function-Module.

Node_Composition:

Each node has a graphical representation which may include related graphical entities such as $\triangle$ and o. A node's composition descriptor is a pointer to a linked chain of object relations describing the graphical composition of a node before the node is expanded.

Node_Expansion:

Each node has a pointer to a linked chain of object relations describing its SFD's graphical representation, control flow, and data flow.

Node_Control_Entry:

Each node has a pointer to the first object relation in the Node_Expansion chain which represents the first function to be executed when the supervisory-function is called.

Node_Function:

Each node has a corresponding functional description written by an SFD author (i.e., Project manager or Systems Analyst).

Node_Progress:

Each node has a corresponding progress report to keep the system designers informed of system progress.

Node_Notes:

During behavioral demonstration, the user has the opportunity to make notes which would later be read by the node's author(s).

2.3.2 Object Attributes

Object_Name:

Objects of the executable type (e.g., Computational-Worker) have unique object names which serve as primary keys in the object relations. These names correspond to the Node_names in the Node relations. Text for objects of type COND, TEXT, and DATA will appear in this tuple and can be thought of as an attribute of the object that functions the same as the object's name.

Object_Type:

Object_Type describes the graphical entity that an object represents.

Object_Color:

Each object has a color descriptor.

Object_Shading:

Objects may be shaded, unshaded, or system-defined composites.

Object_location:

All objects can be positioned by knowing two absolute coordinates relative to the absolute origin on the screen.

Object_Link

Each object may be composed of other objects and/or SFD nodes. This attribute links objects in the same object chain.

Object_Successor:

Objects of type EDGE represent control lines connecting two nodes. These objects have a successor attributes whose tuple values are pointers to the node to be executed once this EDGE has been traversed.

2.4 Definition of Formal Relations

The conceptual data model is formalized after the CODASYL relational database, and presented in Figure 2.4.1 as that model is implemented under DMS. Clearly, during implementation some new attributes were defined. We introduced these new relations in order to satisfy certain hardware characteristics of the GG device.

NODE RELATION

Name	Type	Composition	Expansion	Control	Function	Progress	Notes	Clear

OBJECT RELATION

Name	Type	Successor	Link	X_1	Y_1	X_2	Y_2	Color	Shading	Line_mode	Write_mode	Blinking

FIGURE 2.4.1 FORMAL DATABASE RELATIONS

Figure 2.4.2 shows how the formal relations appear in the transaction database.

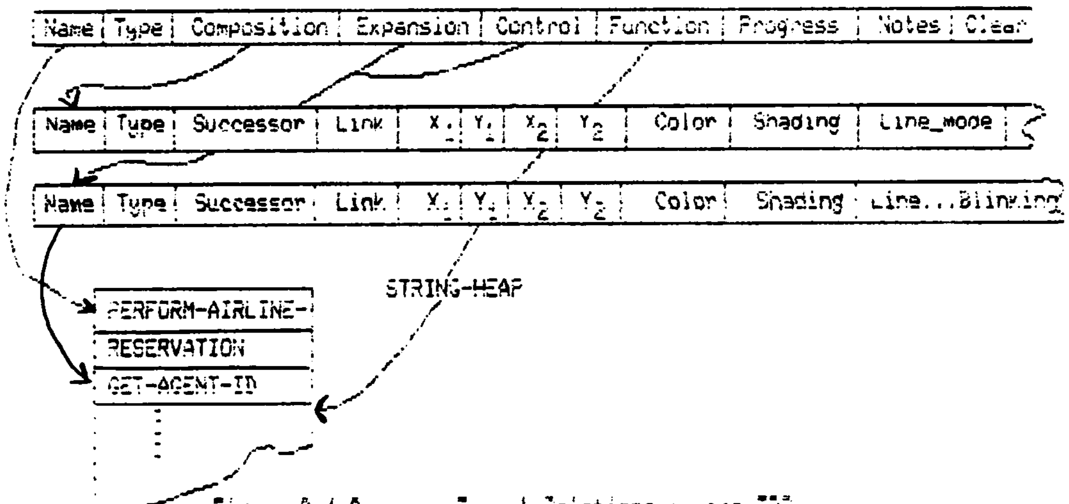


Figure 2.4.2 Formal Relations in the TDB

3.0 BD EXECUTION ENVIRONMENT

The BD execution environment contains three support subsystems: the SFD language editor, the BD database manager, and the BD proper. The SFD language editor and the BD database manager were discussed earlier; in this Section, the BD proper and its relationship to the BD database are examined.

3.1 BD: A Finite-State Automata

The BD is a push-down automata operating on relations contained in the transaction database. As each SFD node is retrieved from the BD database, the BD moves from state to state and node expansion to node expansion. A stack becomes useful when the BD retrieves a supervisory-cell that has an SFD expansion. The BD, in this case, must stack the current window of the supervisory-structure shown on the GG and the current node position within the SFD now executing. In this way, the BD will be able to "return" from a lower-level SFD expansion to its supervisory-function and continue behavioral demonstration. In at least one situation, the handling of implied conditionals, semantic actions are also placed on the stack; hence, the BD could be viewed as an augmented transition network.

Figure 3.1.1 shows the stacking action of the BD.

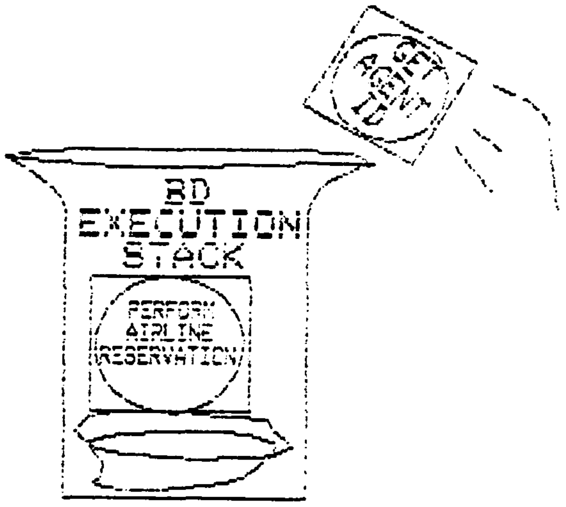
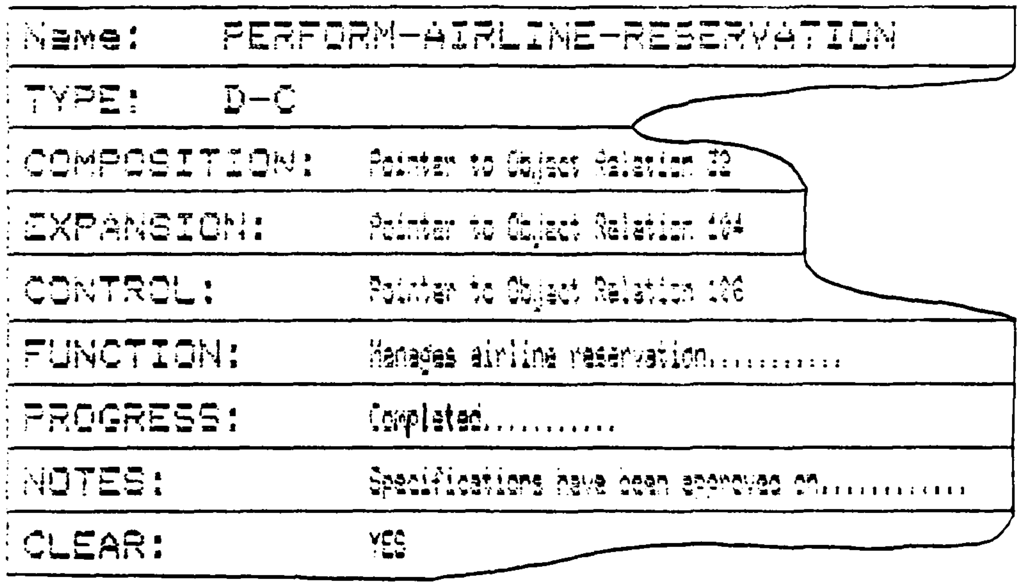


FIGURE 3.1.1 STACKING ACTION OF THE BD

During behavioral demonstration, the BD dynamically binds with each pair of edge-connected SFD nodes a control and data-flow structure. This structure then governs how the BD displays on the TT the control-path alternatives, the functional descriptions, the data descriptors, and the command-level alternatives. The BD treats each SFD entry as a token of the graphical-requirements language. The refined SFD syntax governs how each SFD node may be interpreted and later expanded.

For illustration, consider our airline example discussed in Section 1.3. The BD first retrieves the node relation with Node_Name = 'PERFORM-AIRLINE-RESERVATION.' A typical relation retrieval might return the tuples shown in Figure 3.1.2. The BD then expands the node's graphical composition and expansion by following Node_Composition and Node_Expansion to their respective Object chains and converting the tuple values retrieved from these Object relations into subroutine calls for graphic display. The BD displays the functional description for the current cell by simply displaying the value contained in the Node_Function tuple. It then follows Node_Control_Entry to the first Object relation on the control path. Based on the tuple values obtained from this object relation the BD redraws the first node blinking.



Name:	PERFORM-AIRLINE-RESERVATION
TYPE:	D-C
COMPOSITION:	Pointer to Object Relation 32
EXPANSION:	Pointer to Object Relation 104
CONTROL:	Pointer to Object Relation 106
FUNCTION:	Manages airline reservation...........
PROGRESS:	Completed..........
NOTES:	Specifications have been approved on............
CLEAR:	YES

Figure 3.1.2 TYPICAL RELATION RETRIEVAL

The current state-of-affairs is placed on the stack and expansion for the blinking node continues similarly. After control returns to the supervisory-cell (GET-AGENT-ID is popped off the stack, the object relations comprising the cell are searched for objects of type DATA, COND, and EDGE. DATA objects are used for displaying parameters and COND objects are bound with EDGE object

Object_Successors. The BD generates keys on the TT to correspond with the COND labels (discussed in Section 3.3.4) and binds with the input of each key the object pointer contained in the Object_Successor of the bound EDGE object.

After continuing in this process, the BD will expand the last node in the SFD for PERFORM-AIRLINE-RESERVATION and return to the PERFORM-AIRLINE-RESERVATION node, concluding behavioral demonstration.

3.2 Graphical Expansion

The following algorithim describes how the BD windows SFDs:

```
INPUT : Object relation to execute.
OUTPUT: Object's SFD expansion window.
     Save Object_Location coordinates
     Extract Node relation where Node_Name = Object Name
     Extract Object relation pointed to by Node_Composition
     Take the difference between the saved Object_Location
        coordinates and the Object_Location coordinates
        specified by Node_Composition
     While the coordinates are different do
        Translate the current Node_Expansion object by
           object by an offset
        Add offset to the saved Object_Location coordinates
           and resave
     End While
     Expand Object chain specified by Node_Expansion
     Blink object specified by Node_Control_Entry
     Go to Control Mode
```

The above algorithim illustrates the control flow for SFD windowing but fails to explain how the objects are actually drawn. We will not concern ourselves with this issue since it is hardware dependent. In a nutshell, the BD converts object tuple values into parameters for simple graphic routines which draw basic objects.

3.3 Textual Expansion

The following algorithm describes how the BD dynamically binds conditionals to control paths and generates control-option labels for labelling touch-keys on the TT:

```
INPUT : Object relation to execute.
OUTPUT: New path for control [user's choice].
    If Object_Type = IMPL then
        Next_Object = Object_on_Stack
    Else
        Index1 = 1
        Index2 = 1
        Control_key(1) = 'NEXT NODE'
        While Object_Type is not executable do
            Extract next Object relation on Object chain
            If Object_Type = COND then
                Control_key(Index1) = Object_Name
                Index1 = Index1 + 1
            End If
            If Object_Type = EDGE then
                Control_path(Index2) = Object_Successor
                Index2 = Index2 + 1
            End If
        End While
        Display Control_keys and receive input in KEY
        Index = Input_key(KEY)
        Next_Object = Control_path(Index)
    End If
```

Since the other textual-expansion functions the BD performs are fairly straight-forward their algorithms are not included.

3.3.1 TDB Description

[HARTH83] describes the Transaction Data Base (TDB) as a database resident in primary memory that manages screens for application programs. In a later version of DMS there will be a secondary storage database, the Dialogue Data Base (DDB), which will contain all screens for applications. The TDB contains the following predefined relations [HARTH83]: Screen, Group, and Object. Each Screen is composed of Groups of Objects, where Objects in the same group are stored in a dequeue that links Object relations. These three relations can easily represent the SFD Node and Object relations thus maintaining consistency within the DMS environment. For more information concerning the TDB or SFD relations as implemented on the TDB we refer our readers to [HARTH83].

3.3.2 BD I/O Support

DMS currently supports two device drivers as described in [EHRR82a]. Following the SBSDM tradition, these drivers are intelligent device-drivers providing high-level interactive dialogue services for both the TT and GG devices. PAD, a service that draws a touch-panel on the screen, is an example common to both drivers; whereas, CIRCLE, a service that draws a circle on the screen, is an example included only in the GG device-driver. The BD uses the TT and GG drivers to generate the dialogue for textual and graphical expansion, respectively.

During execution, the BD retrieves tuple-relations from the TDB, translates the tuple values into service calls to the TT and GG drivers, and makes the calls to the drivers to provide the user interactions. The user experiences "requirements execution." The next section illustrates by example this tuple-value translation process.

3.3.3 An Example Tuple Translation

Consider the airline example examined in Section 3.1. Figure 3.3.1 shows the tuples retrieved by the BD when it retrieves the supervisory-cell PERFORM-AIRLINE-RESERVATION.

Tuple	Value
Name:	PERFORM-AIRLINE-RESERVATION
TYPE:	D-C
COMPOSITION:	Pointer to Object Relation 22
EXPANSION:	Pointer to Object Relation 104
CONTROL:	Pointer to Object Relation 106
FUNCTION:	Manages airline reservation...........
PROGRESS:	Completed________ Task Group____________
NOTES:	Nil
CLEAR:	Yes

Figure 3.3.1 Tuple Retrieval for PERFORM-AIRLINE-RESERVATION

The BD first uses the value of Node_Composition to extract the Object chain describing how the node is to be drawn on the GG. Figure 3.3.2 illustrates these relations.

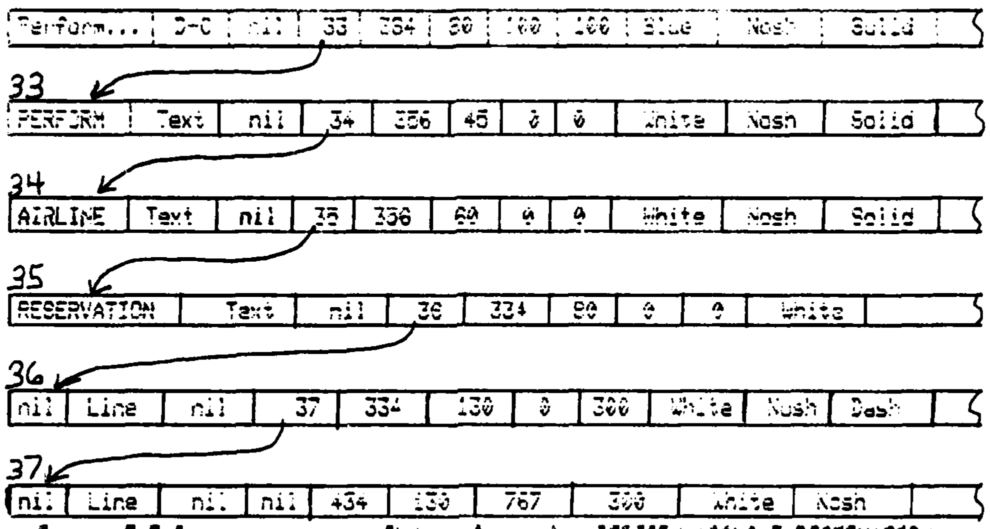

Perform... | D-C | nil | 33 | 334 | 80 | 100 | 100 | Blue | Nosh | Solid
33
PERFORM | Text | nil | 34 | 336 | 45 | 0 | 0 | White | Nosh | Solid
34
AIRLINE | Text | nil | 35 | 336 | 60 | 0 | 0 | White | Nosh | Solid
35
RESERVATION | Text | nil | 36 | 334 | 90 | 0 | 0 | White
36
nil | Line | nil | 37 | 334 | 130 | 0 | 300 | White | Nosh | Dash
37
nil | Line | nil | nil | 434 | 130 | 767 | 300 | White | Nosh

Figure 3.3.2 Object Chain for PERFORM-AIRLINE-RESERVATION

The first Object relation translates into the following FORTRAN calls to the GG device-driver:

```
Call Box ('gg',X2,Y2,X1,Y1,'keep'//line_mode//write_mode//color//
          shading//blinking)
Call Circle
          ('gg',X2/2,X1,Y1,'keep'//line_mode//write_mode//color//
          shading//blinking)
```

For more information on the device-drivers see [EHRR82b].

The BD next expands the function description for PERFORM-AIRLINE-RESERVATION on the TT by simply printing the value of the Node_Function tuple. It next clears the GG screen based on the value of Node_Clear and expands the SFD expansion by translating the Object chain pointed to by the value of Node_Expansion into GG device-driver calls. Once the expansion has been drawn, the BD stacks the current node and object values and passes control to the object pointed to by Node_Control_Entry.

Tuple translation continues similarly until the tuple value for Node_Control_Entry = nil or control passes to an object of type RETN. At this point the BD "pops" the stack values and continues execution.

3.3.4 BD Command Alternatives

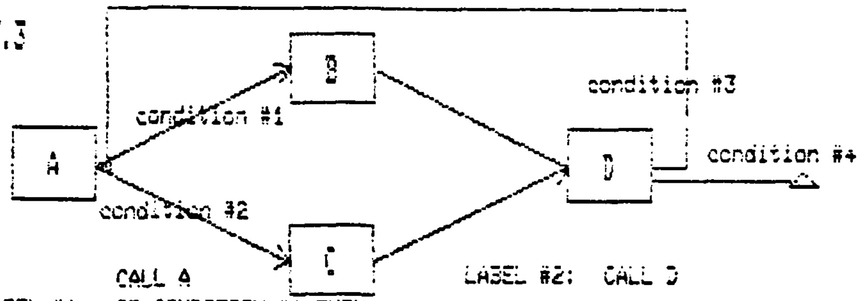
Previous Sections have alluded to the Progress Reports and User Notes but have not discussed how BD users access and use these entities. Figures 1.3.4 - 1.3.9 show various command alternatives as they appear on the TT touch-panel for our airline example. These figures illustrate the three command alternatives which are detailed here:

CONTINUE	- continues sequential execution. This command is used for stepping sequentially through TT displays.
PROGRESS	- displays the node's Progress Report in the functional-description text box.
NOTES	- places the BD into edit-mode so that the user may enter notes. After these notes have been entered, they will be routed to the mail files of the persons responsible for the node's expansion.

In earlier Sections we mentioned the implied conditional. This is an implied state and command alternative included in the SFD language syntax. Consider the SFD and its corresponding code shown in Figure 3.3.3.

IMPLIED CONDITIONALS

Figure 3.3.3



```
           CALL A                                    LABEL #2:  CALL D
LABEL #1:  IF CONDITION #1 THEN                                 IF CONDITION #3 THEN
              CALL B                                               GOTO LABEL #1
              GOTO LABEL #2                                     END IF
           END IF                                               IF CONDITION #4 THEN
           IF CONDITION #2 THEN                                    RETURN
              CALL C                                            END IF
              GOTO LABEL #2
           END IF
```

When control has passed through node A and the user has chosen to go to node C, then after leaving node D control passes to node C. This causes node A to be expanded only once.

3.3.5 Problems and Solutions

Currently, the BD places a restriction on SFD structures: only fifteen edges are allowed on nodes. The TT touch-panel having only eighteen keys makes this restriction. Clearly, this could become a problem for an application-system design for which there are large "Case" structures. A solution to this problem, to be included in a future version of the BD, would be to add a service to both the TT and GG drivers which can maintain a touch-keyboard similar to the one depicted in Figure 3.3.4. During execution, the BD would adapt keyboards to the application required.

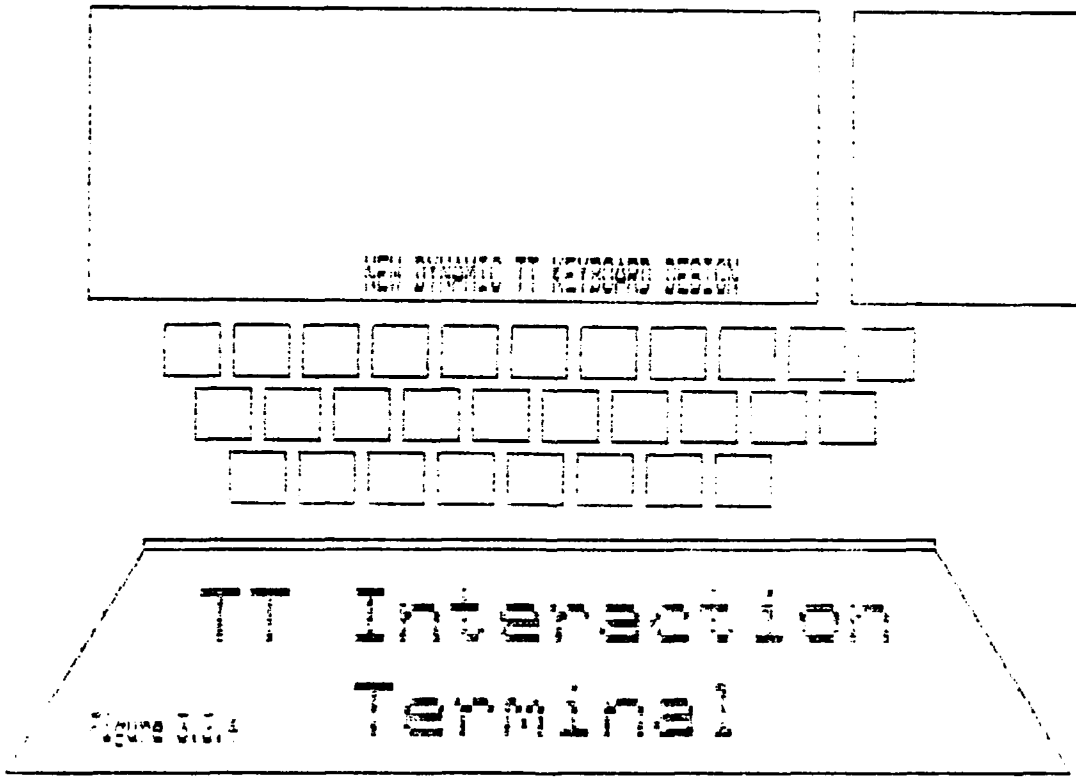
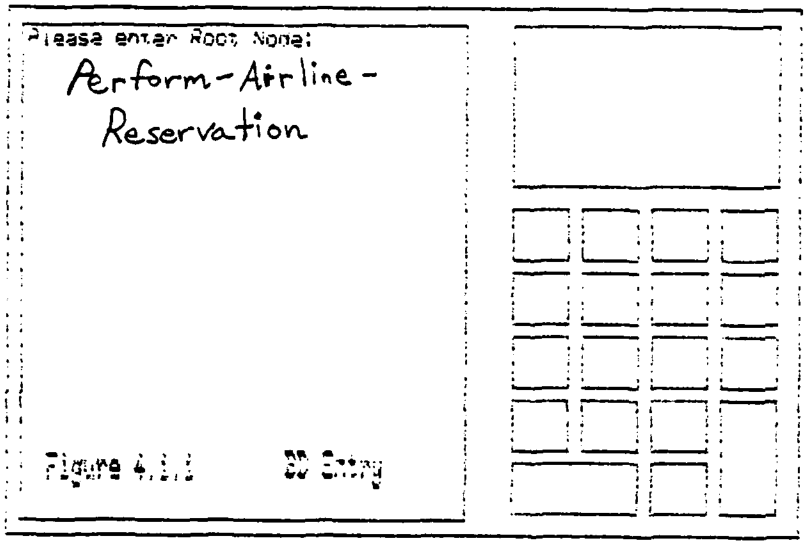


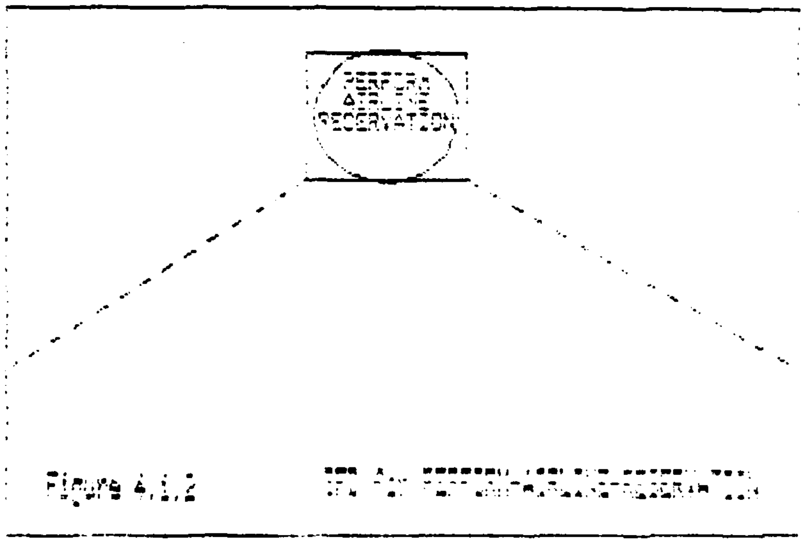
Figure 3.3.4

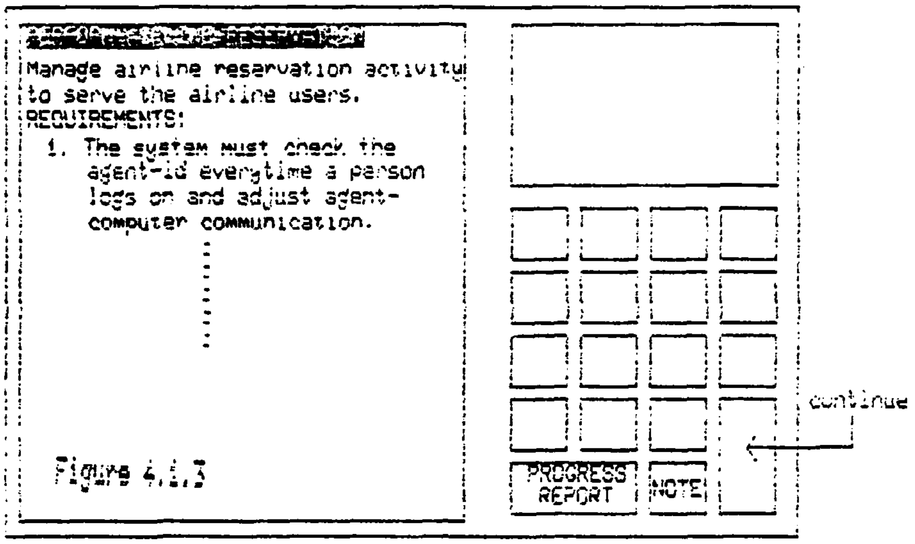
4.0 BEHAVIORAL DEMONSTRATION OF THE AIRLINE EXAMPLE

Figures 4.1.1 - 4.3.3 depict a terminal session with the BD. The airline example is used again. Data expansion is excluded for simplicity, but for completeness it is illustrated in Figure 4.2.5.



After the user enters the root node, in this case PERFORM-AIRLINE-RESERVATION, the database is searched and the SFD for the root node retrieved.


PERFORM AIRLINE RESERVATION
Figure 4.1.2
SFD for PERFORM-AIRLINE-RESERVATION



PERFORM-AIRLINE-RESERVATION

Manage airline reservation activity
to serve the airline users.
REQUIREMENTS:

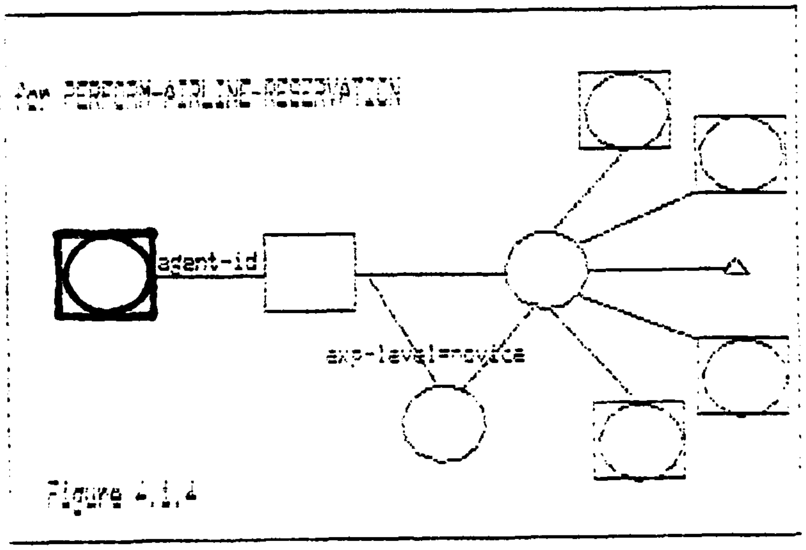
1. The system must check the
   agent-id everytime a person
   logs on and adjust agent-
   computer communication.

Figure 4.1.3

Textual Expansion for PERFORM-AIRLINE-RESERVATION

The requirements specifications for the root node are displayed, and the user selects a command option. In this case CONTINUE is selected.

SFD Expansion for PERFORM-AIRLINE-RESERVATION



The SFD for the current node is expanded and demonstration continues with the entry node in the SFD expansion.

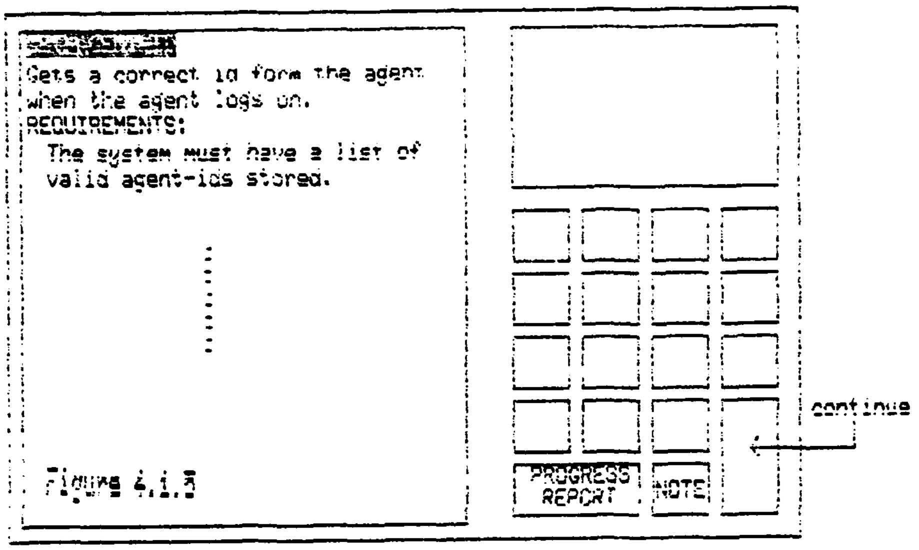

[illegible]
Gets a correct id form the agent
when the agent logs on.
REQUIREMENTS:
 The system must have a list of
 valid agent-ids stored.

Figure 4.1.5

Textual Expansion for GET-AGENT-ID

User selects CONTINUE.

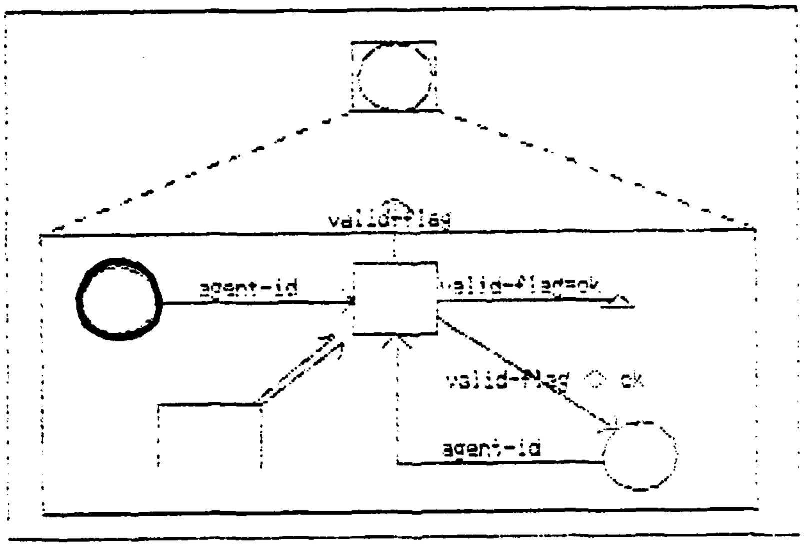


Figure 4.1.6 SFD Expansion for GET-AGENT-ID

The SFD for GET-AGENT-ID is expanded and demonstration continues with ASK-FOR-AGENT-ID.

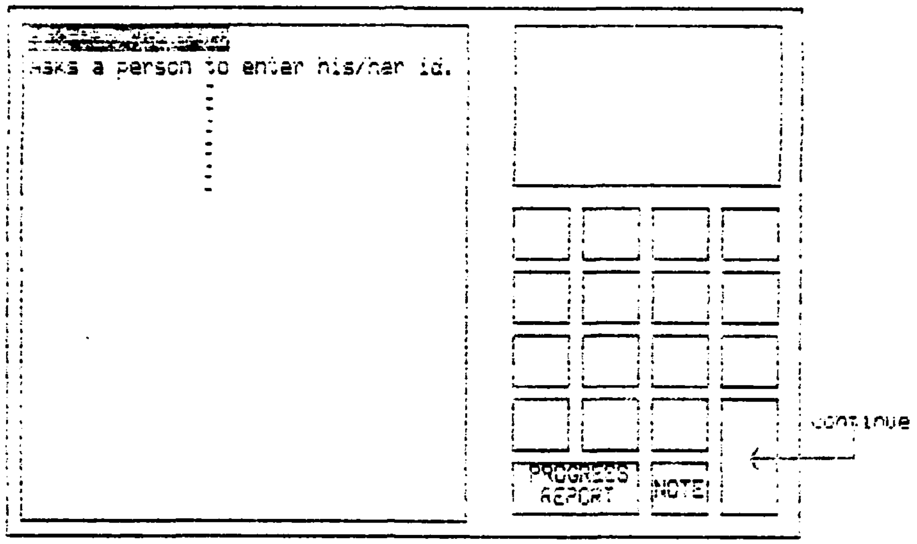


Figure 4.1.7 Textual Expansion for ASK-FOR-AGENT-ID

User presses CONTINUE.

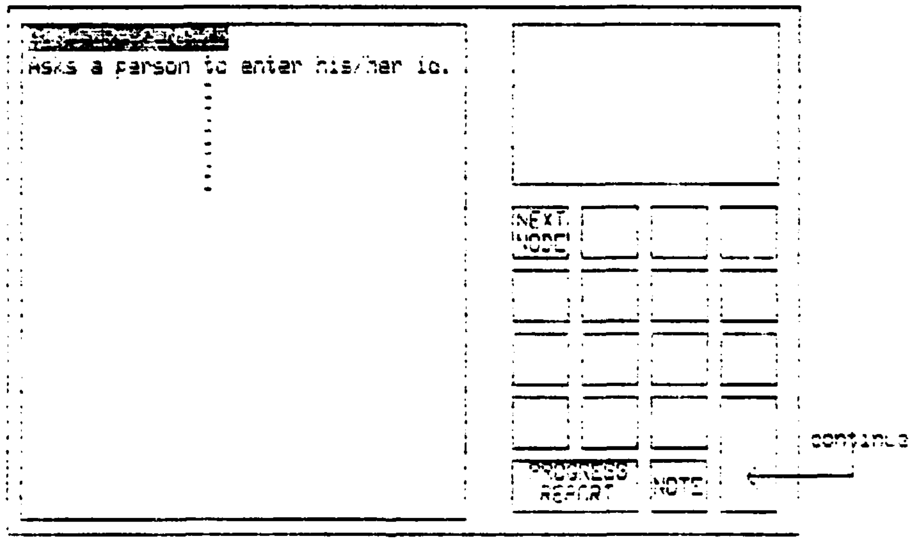


Figure 4.1.8 [illegible] Devices for ASK-FOR-AGENT-ID

Since ASK-FOR-AGENT-ID is a worker node (Dialogue) the user is given the option to continue sequentially through the SFD to the next node. User selects NEXT NODE.

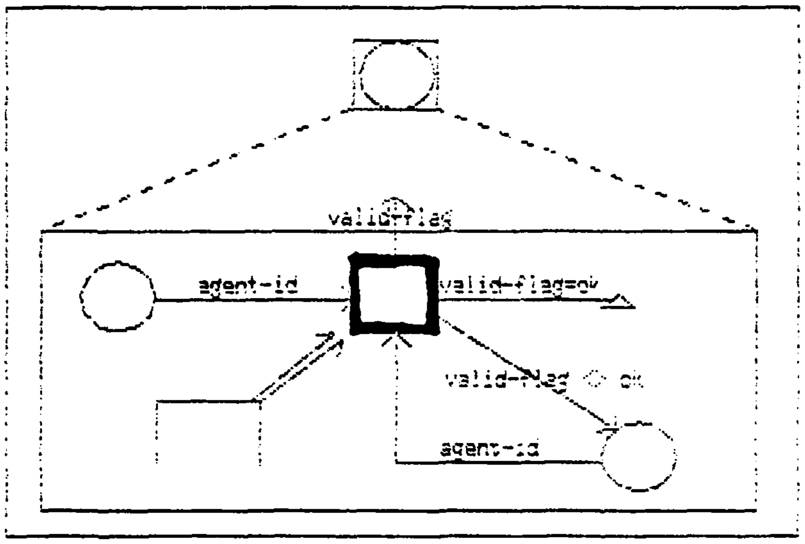


Figure 4.1.3 SFD Expansion for GET-AGENT-ID

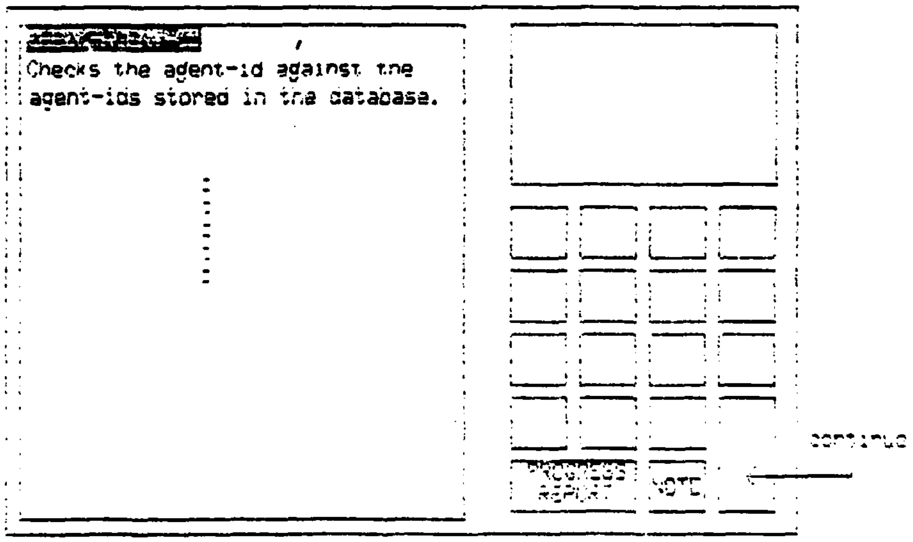
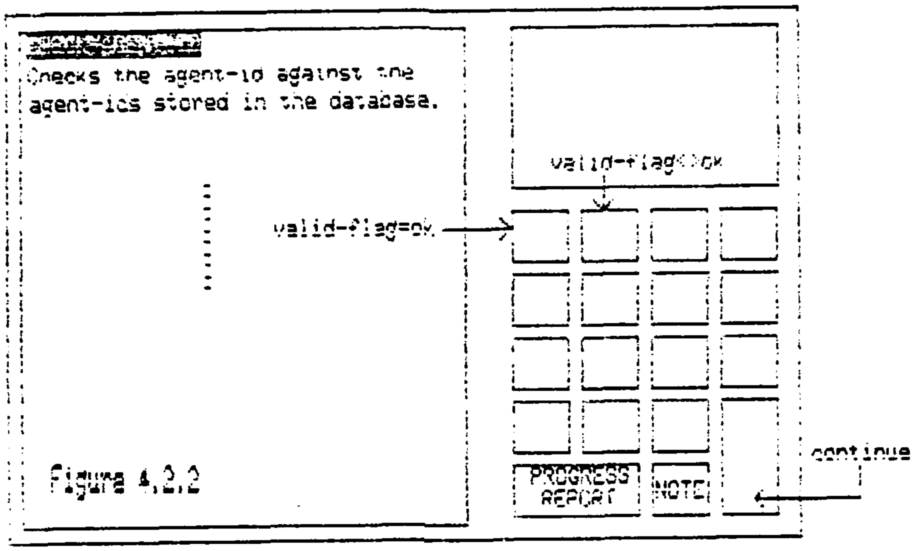

Checks the agent-id against the
agent-ids stored in the database.

Figure 4.2.1 Textual Expansion for CHECK-AGENT-ID


Checks the agent-id against the agent-ids stored in the database.

Control Options for CHECK-AGENT-ID

Demonstration continues until control reaches the decision point where valid-flag requires a value. In Figure 4.2.2 the user is given the choice of two control paths. User selects Valid-Flag<>OK.

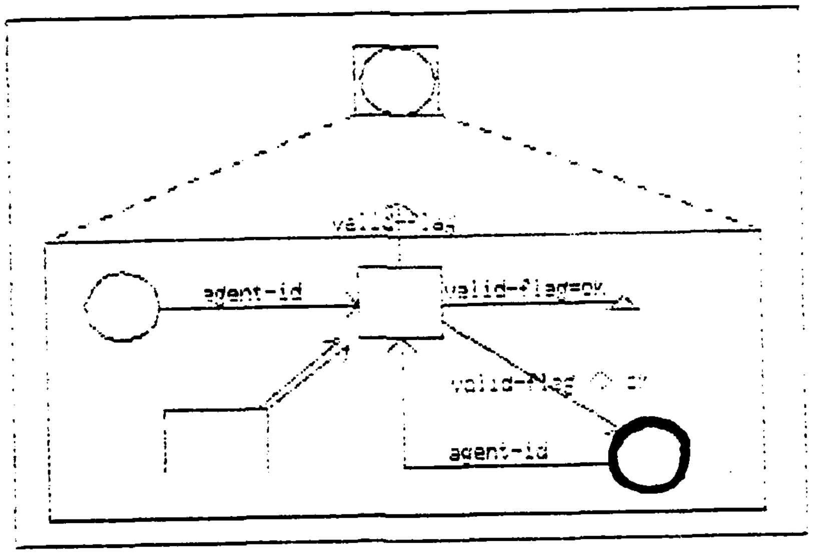


Figure 4.2.3 SFD Expansion for GET-AGENT-ID

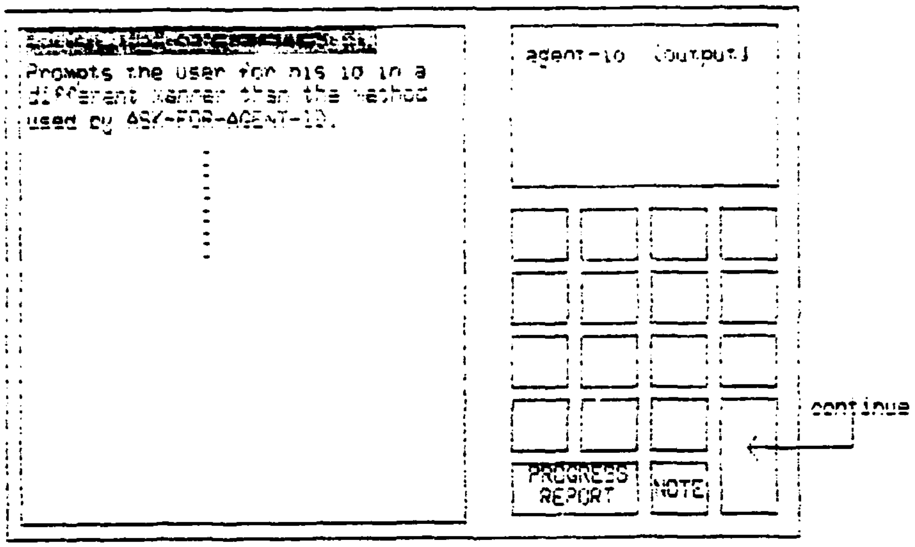

Prompts the user for his id in a different manner than the method used by ASK-FOR-AGENT-ID.

Figure 4.2.4 Textual Expansion for PROMPT-THE-USER-GET-IT-AGAIN

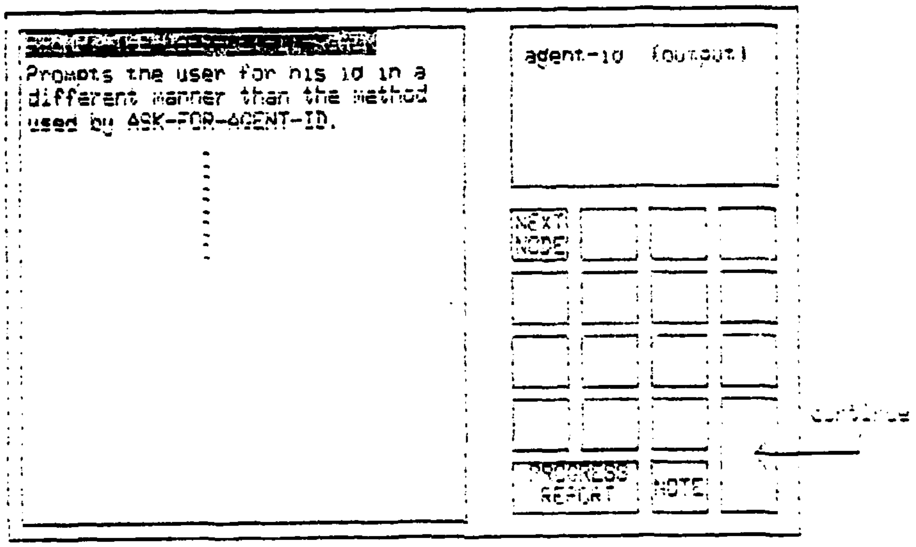

Prompts the user for his id in a different manner than the method used by ASK-FOR-AGENT-ID.

Figure 4.2.5 Control Options for PROMPT-THE-USER-GET-IT-AGAIN

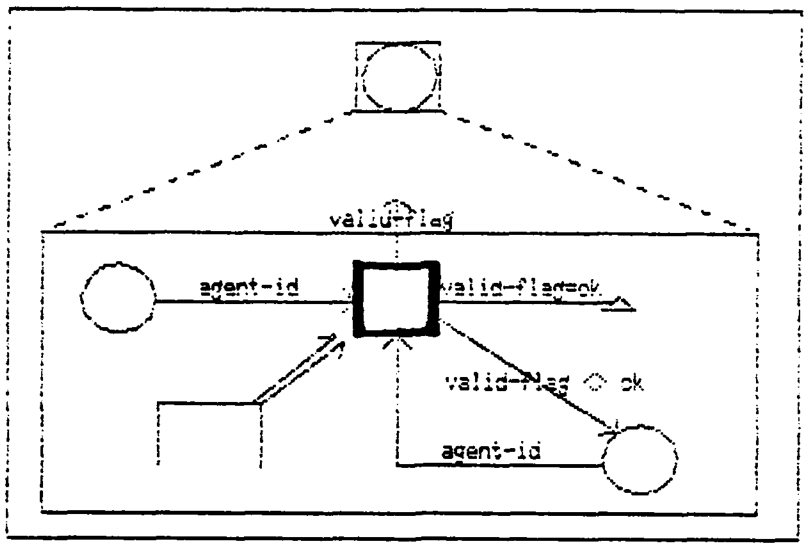


Figure 4.2.6 SFD Expansion for GET-AGENT-ID

Control continues in an iterative loop until the user selects Valid-Flag=OK in Figure 4.2.6, then demonstration resumes with the supervisory-structure that supervises GET-AGENT-ID, shown in Figure 4.2.7.

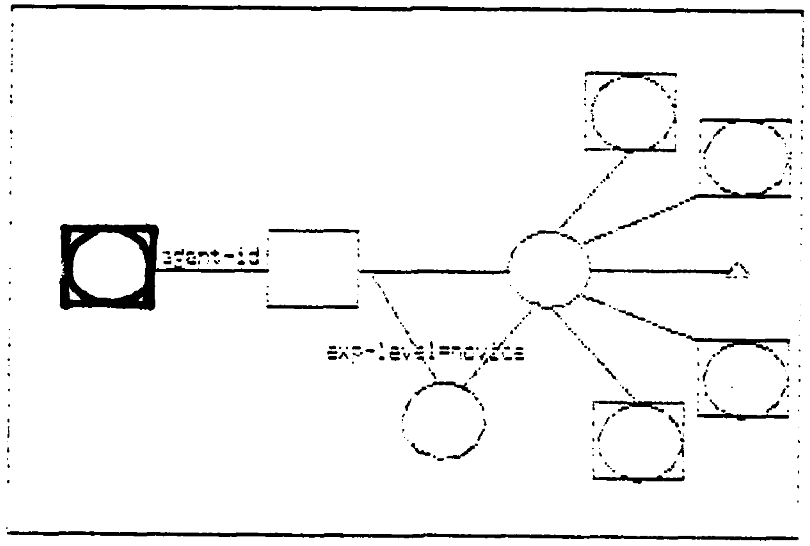
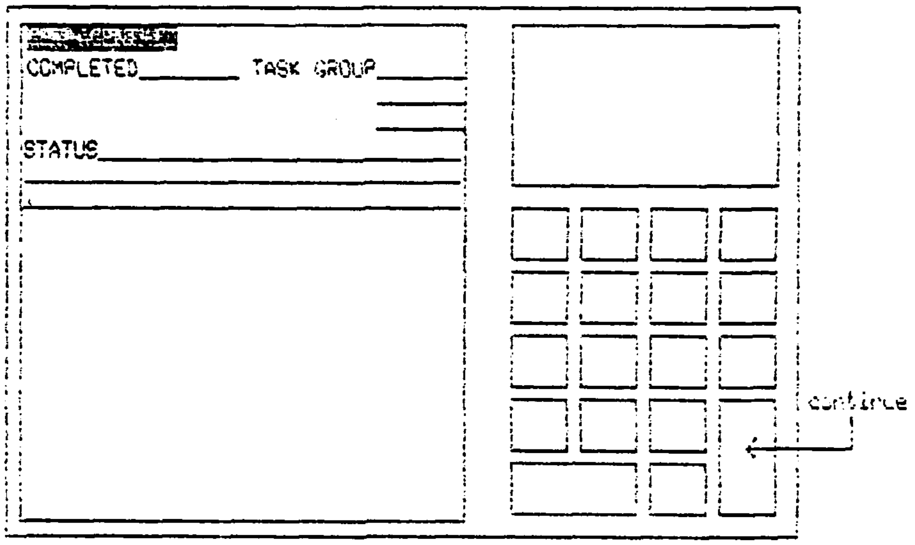


Figure 4.2.7 SFD Expansion for PERFORM-AIRLINE-RESERVATION


COMPLETED________ TASK GROUP________

STATUS________

Figure 4.2.8 Progress Report for GET-AGENT-ID

Demonstration continues, and in Figure 4.2.8 the user has selected PROGRESS REPORT.

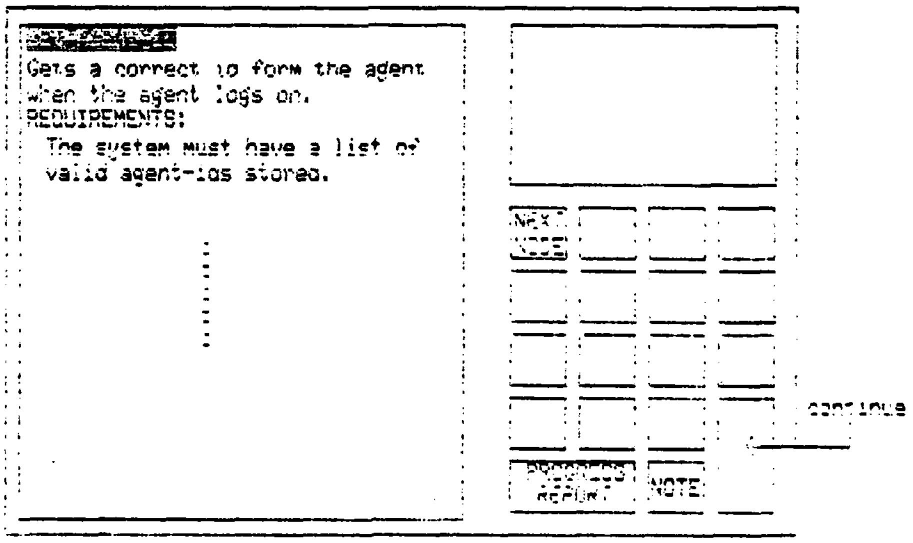

Gets a correct id form the agent when the agent logs on.
REQUIREMENTS:
The system must have a list of valid agent-ids stored.

Figure 4.2.9 Control Options for GET-AGENT-ID

DFD Expansion for PERFORM-AIRLINE-RESERVATION

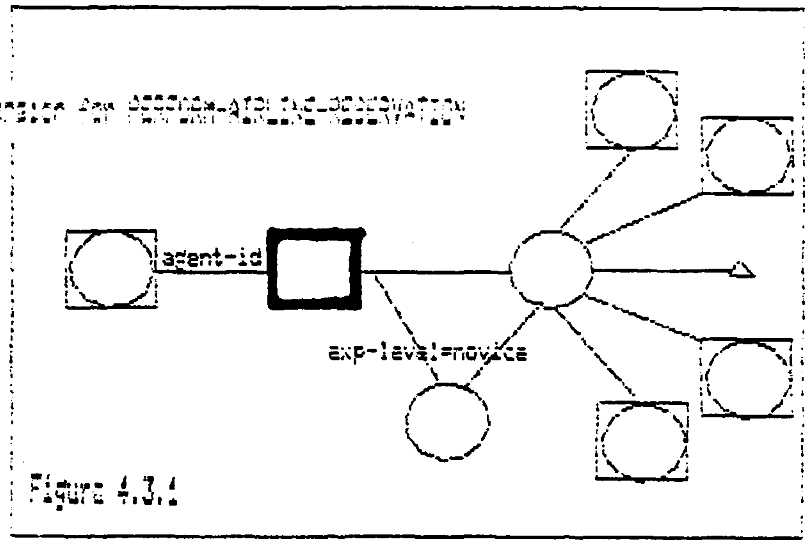
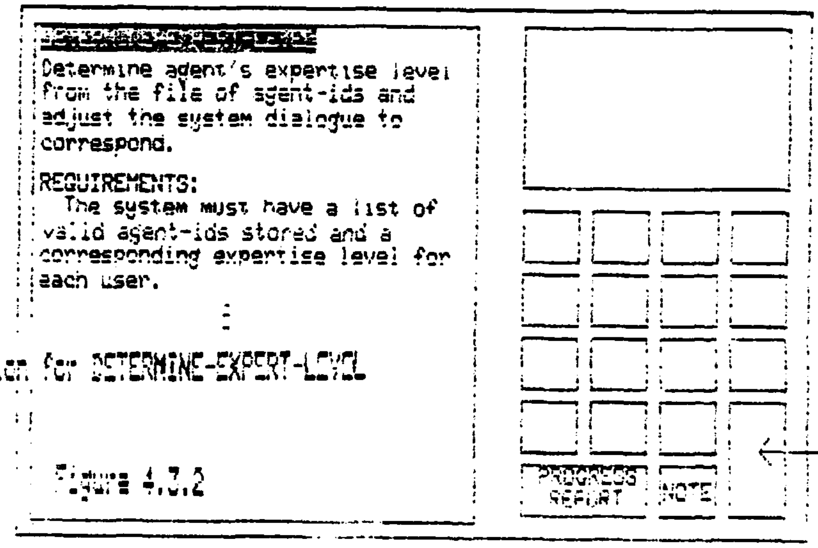


Figure 4.3.1

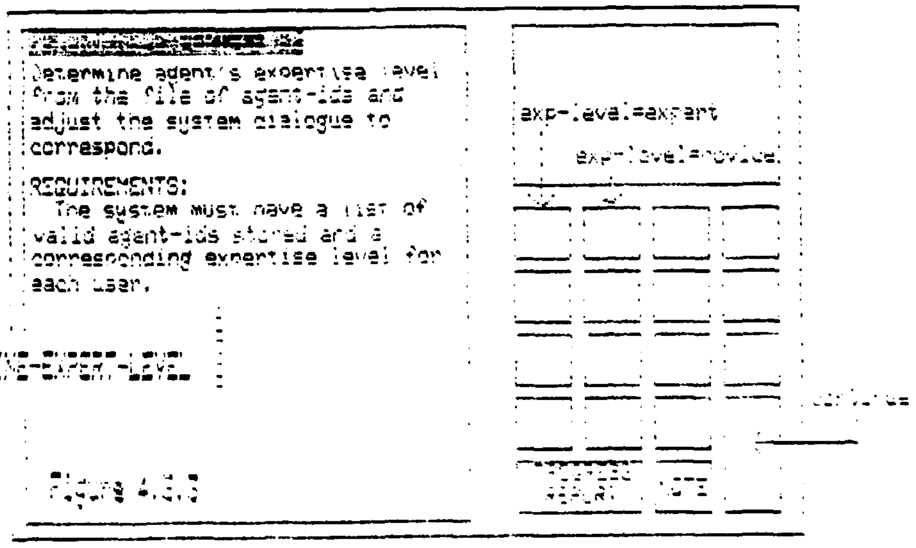

Determine agent's expertise level from the file of agent-ids and adjust the system dialogue to correspond.

REQUIREMENTS:
The system must have a list of valid agent-ids stored and a corresponding expertise level for each user.

Textual Expansion for DETERMINE-EXPERT-LEVEL

Figure 4.3.2

Demonstration continues


Determine agent's expertise level from the file of agent-ids and adjust the system dialogue to correspond.

REQUIREMENTS:
The system must have a list of valid agent-ids stored and a corresponding expertise level for each user.

Control Options for DETERMINE-EXPERT-LEVEL

Figure 4.3.3

5.0 FUTURE WORK

In Section 1.0 the BD was presented as one of three similar and related design-tools to be integrated into DMS. Neither the Behavioral Translator, nor the SFD Compiler have as yet been discussed. Those design tools represent areas for future work.

The SFD Compiler will produce the code for a system, or subsets of a system, after the supervisory-structure for the system, or subset, has been completed. The code produced will be written in Fortran, Pascal, or some other high-level programming language. Clearly, since most of the complicated control constructs will be coded automatically by the SFD compiler, productivity on a system design using the SBSDM will be greatly increased.

The SFD Translator allows designers of systems to execute any completed portion of a system during any phase. By checking the progress report on each node the SFD Translator determines if the node has been completed. Completed computational nodes are linked and the data necessary for execution binded. A separate process is allocated to "run" the completed nodes and interprocess "windows" are used to retrieve output data. For completed dialogue nodes the SFD Translator calls the "dialogue executer" to perform a dialogue-transaction. See [JOHND82a] for a description of dialogue-transactions and the "dialogue executer."

REFERENCES

EHRR82a Ehrich, R. W. "The DMS Input-Output Services." Departments of Computer Science and Industrial Engineering Technical Report, VPI&SU (1982).

EHRR82b Ehrich, R. W. "The DMS Multiprocess Execution Environment." Department of Computer Science Technical Report, VPI&SU (1982).

HARTH83 Hartson, H.R., Goodwin J., and Ahmed S. "A Manual for The Transaction Database." Department of Computer Science Technical Report, VPI&SU (1983).

JOHND82a Johnson, D. H. and Hartson, H. R. "The Role and Tools of a Dialogue Author in Creating Human-Computer Interfaces." Department of Computer Science Technical Report CSIE-82-8, VPI&SU (1983).

JOHND82b Johnson, D. H. and Hartson, H. R. "Dialogue Management: New Concepts in Human-Computer Interface Development." Department of Computer Science Technical Report, VPI&SU (1982).

YUNTT82 Yunten, T. and Hartson, H. R. "Human-Computer System Development Methodology for the Dialogue Management System." Department of Computer Science Technical Report CSIE-82-7, VPI&SU (1982).

YUNTT83 Yunten, T. and Hartson, H. R. "Supervisor-Based System Development Methodology (SBSDM)." Department of Computer Science Technical Report CSIE-83-10, VPI&SU (1983).

OFFICE OF NAVAL RESEARCH

Engineering Psychology Group

TECHNICAL REPORTS DISTRIBUTION LIST

OSD

CAPT Paul R. Chatelier
Office of the Deputy Under Secretary
of Defense
OUSDRE (E&LS)
Pentagon, Room 3D129
Washington, D.C. 20301

Department of the Navy

Engineering Psychology Group
Office of Naval Research
Code 442 EP
Arlington, VA 22217 (2 cys.)

Communication & Computer Technology
Programs
Code 240
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Information Sciences Division
Code 433
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Special Assistant for Marine Corps
Matters
Code 100M
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Director
Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375

Dr. Michael Melich
Communications Sciences Division
Code 7500
Naval Research Laboratory
Washington, D.C. 20375

Department of the Navy

Dr. J.S. Lawson
Naval Electronic Systems Command
NELEX-06T
Washington, D.C. 20306

Dr. Robert E. Conley
Office of Chief of Naval Operations
Command and Control
OP-094H
Washington, D.C. 20350

Dr. Robert G. Smith
Office of the Chief of Naval
Operations, OP987H
Personnel Logistics Plans
Washington, D.C. 20350

Combat Control Systems Department
Code 35
Naval Underwater Systems Center
Newport, RI 02840

Dean of Research Administration
Naval Postgraduate School
Monterey, CA 93940

J. Impagliazzo
Code 101
Naval Underwater Systems Center
Newport, RI 02840

Dr. A.L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, D.C. 20380

Dr. L. Chmura
Naval Research Laboratory
Code 7592
Computer Sciences & Systems
Washington, D.C. 20375

Department of the Navy

Chief, C^3 Division
Development Center
MCDEC
Quantico, VA 22134

Human Factors Technology Administrator
Office of Naval Technology
Code MAT 0722
800 N. Quincy Street
Arlington, VA 22217

Mr. Philip Andrews
Naval Sea Systems Command
NAVSEA 03416
Washington, D.C. 20362

Commander
Naval Electronics Systems Command
Human Factors Engineering Branch
Code 81323
Washington, D.C. 20360

Larry Olmstead
Naval Surface Weapons Center
NSWC/DL
Code N-32
Dahlgren, VA 22448

Dean of the Academic Departments
U.S. Naval Academy
Annapolis, MD 21402

Office of the Chief of Naval
Operations (OP-115)
Washington, D.C. 20350

Department of the Army

Mr. J. Barber
HQS, Department of the Army
DAPE-MBR
Washington, D.C. 20310

Other Government Agencies

Defense Technical Information Center
Cameron Station, Bldg. 5
Alexandria, VA 22314 (12 copies)

Other Organizations

Mr. Edward M. Connelly
Performance Measurement
Associates, Inc.
410 Pine Street, S.E.
Suite 300
Vienna, VA 22180