BUSINESS - An End-User Oriented Application Development Language

Peter B. Miller
Sergey Tetelbaum
Kincade N. Webb

Ruben Engineering Corporation
Cambridge, Massachusetts Ø2138

ABSTRACT

BUSINESS is a new application development language intended to answer the needs of a rapidly growing user community - the end-user. The underlying computational model is the paper office. BUSINESS instructions manipulate objects within this model in ways analogous to human office workers. The syntax of BUSINESS closely approximates a subset of English. Standard programming functions are embodied in the language in ways that seem natural to first time computer users. For example, to open a file for exclusive read/write access one 'Pulls' a 'Folder' (containing 'Forms' and 'Documents') from a 'Drawer' in a 'File Cabinet'. In the computational model, the Folder is seen to actually move from the Drawer to a 'Desk' belonging to an 'Aide' (the "worker" inside the computer).

To support the construction of reliable software within this model, BUSINESS contains a rich repertoire of structured control constructs: execute (procedures), do (routines), go through (a set of objects), test and select one (from a set of 'if - then' instructions). Base data types include: whole numbers, decimal numbers, money, dates, times and characters. The set of operators and tests within BUSINESS support a complete range of data manipulations and logical decisions. Data structures include Fillins (collections of Fields) and Tables (having Entries at Row/Column intersections). Permanent data is organized by Form (collections of Fillins and Tables).

1.Ø INTRODUCTION

An ever-widening software "gap" has developed that could critically impact the rate of computer utilization - there is a growing shortage of sufficiently trained personnel to write new and maintain old software in traditional progamming languages. This has prompted the development of tools that allow the rapid, cost-effective production of applications software by other than experienced programmers. BUSINESS is intended to be a fully-developed applications development environment that is accessible to end-users who are not, or who do not wish to become, "professional" programmers. In addition, it has also

been designed to address the issue of programming productivity (i.e the reduction in the cost to produce functional software), program modifiability and program maintenance.

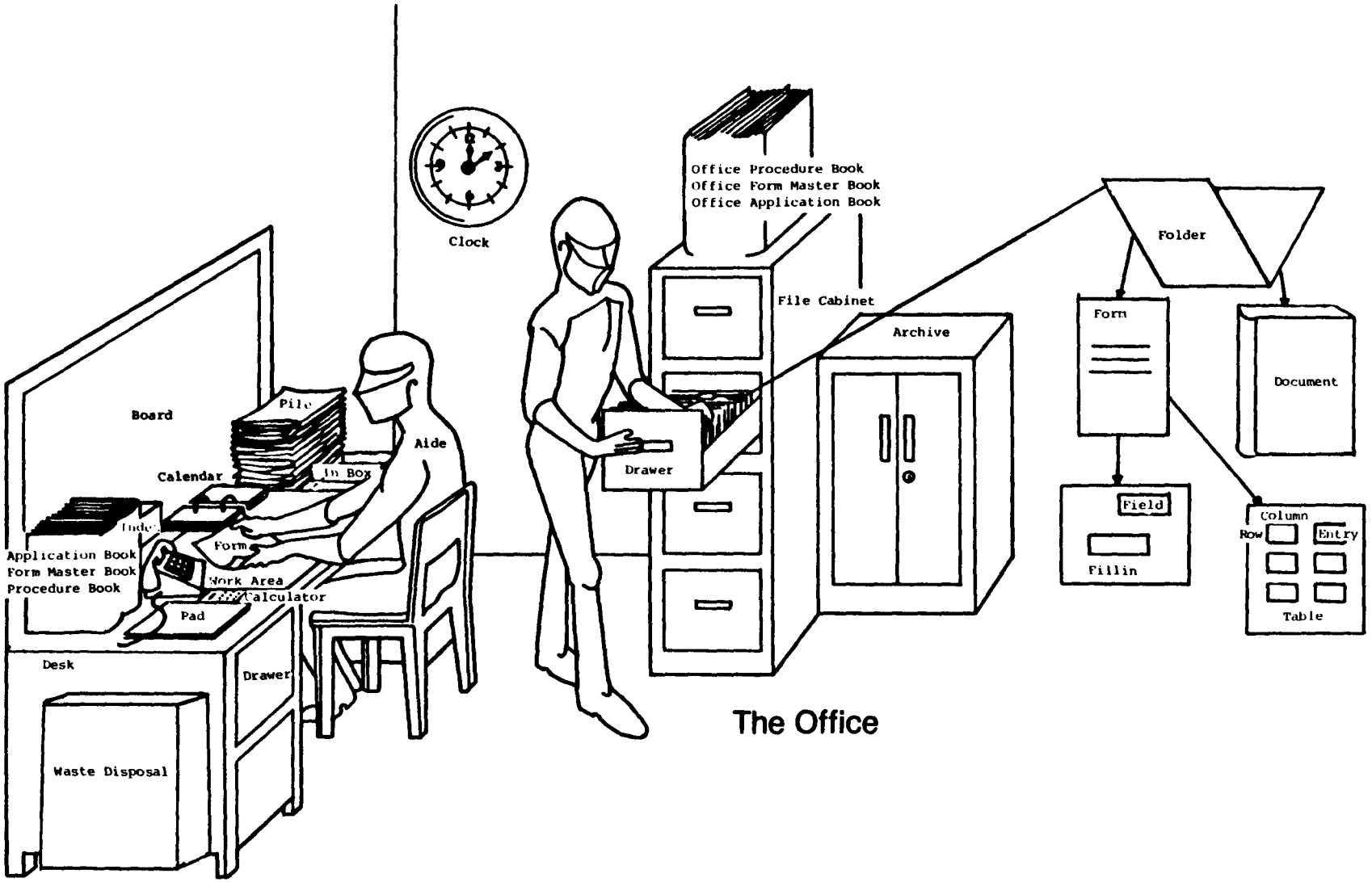
The generation of software can be divided into four distinct stages: requirements analysis, system design, translation to a programming language and verification through proof or exhaustive testing. The most difficult step for an end-user is the translation of his problem domain-oriented conceptualization into a programming language. The advantage of allowing the programming or specification of a solution in terms already familiar to the end-user has been recognized in as diverse areas as simulation and symbolic mathematical analysis. BUSINESS accomplishes this in the area of business data processing and the related area of office automation by modelling the problem domain of the office. This approach has been addressed by various groups such as the End Users Facilities Committee (EUFC) Group of CODASYL [1] and the Office Automation Group at MIT [2]. The office model of BUSINESS closely follows many of the ideas presented in the EUFC report although it was developed independently.

2.0 THE OFFICE MODEL

2.1 Overview

The computational environment of BUSINESS is a reflection of a user's own work environment - an 'office model'. Figure 1 is a representation of the components of the office model. The model consists of 'objects' and a containment relationship - an object that can contain other objects is called a 'location'. Figure 2 is a graph of the containment relationship. At the highest level is an Organization. (Objects that have significance in the office model have an initial capital.) Within an Organization are Offices. One Office within the Organization is designated as the main Office. (An Office corresponds to a node in a distributed processing network.) Within an Office are "workers" called Aides. (An Aide corresponds to an account on a time-sharing system.) Every Office contains a collection of resources including a (public) filing system.

An Office's filing system consists of a collection of File Cabinets one of which is designated as the main File Cabinet. A File Cabinet contains a collection of Drawers. A Drawer contains a collection of Folders. A Folder contains Forms, Documents and Memos. Each Aide's Desk contains Drawers for the Aide's private Folders. On top of an Aide's Desk is a Work Area which can contain Forms, Documents, Memos, Procedures and Form Masters. Also on an Aide's Desk are a (note) Pad, a (black) Board, a calculator, a Calendar, a telephone, an In Box, a collection of Indices containing Index Cards, a collection of Piles containing Folders, Forms, Memos and Documents and the Aide's Form Master, Procedure and Application Books.


Clock
Office Procedure Book
Office Form Master Book
Office Application Book
File Cabinet
Archive
Drawer
Folder
Form
Document
Field
Fillin
Column
Row
Entry
Table
Board
Pile
Aide
Calendar
In Box
Index
Application Book
Form Master Book
Procedure Book
Form
Work Area
Calculator
Pad
Desk
Drawer
Waste Disposal

The Office

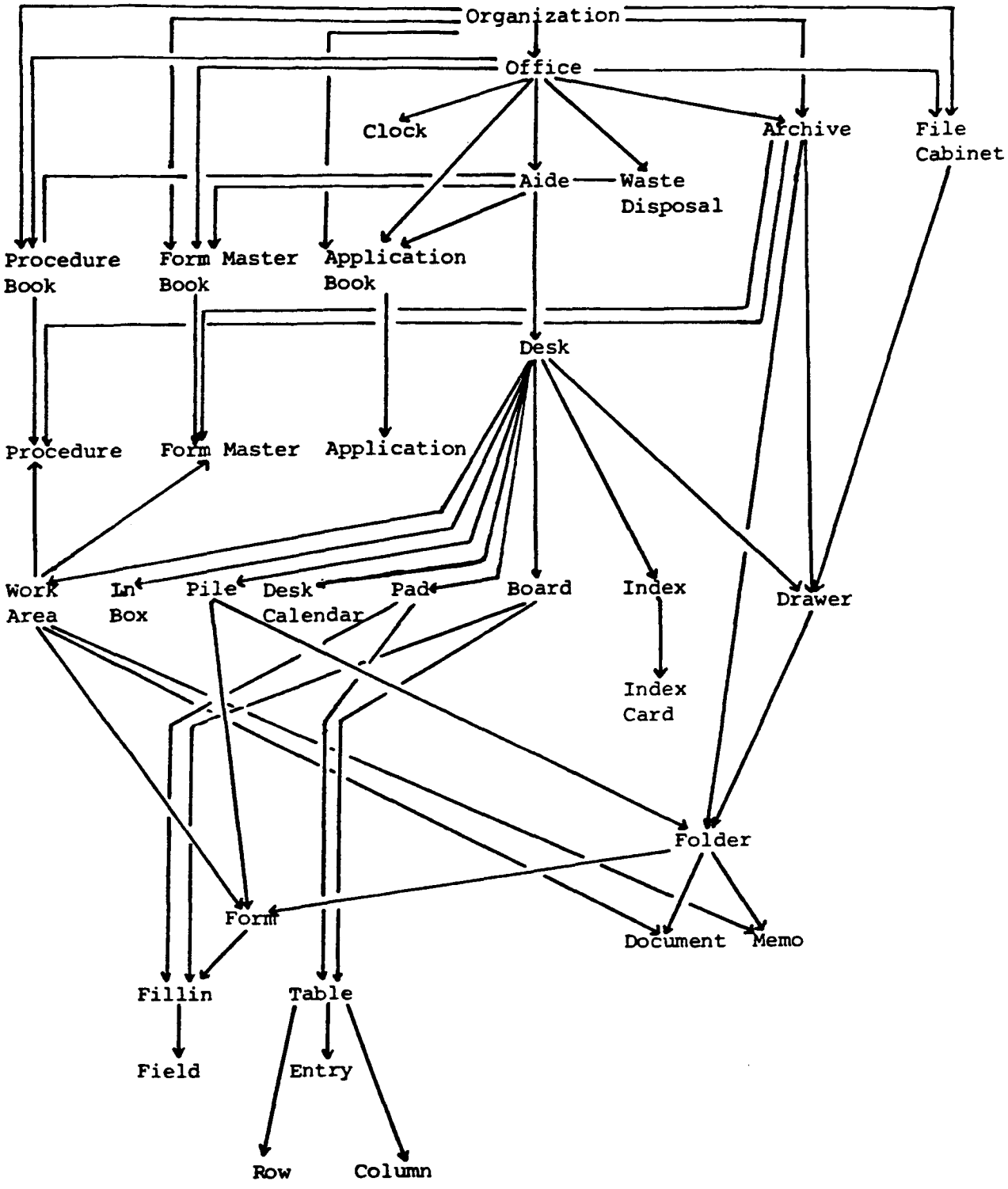


Figure 2: Containment Graph of the Office Model

A Form contains information usually thought of as 'data' while Documents and Memos contain information usually thought of as 'text'. (Memos are used to communicate among users.) A Form has a logical schema composed of combinations of two types of structures, Fillins and Tables. A Fillin is an unstructured collection of Fields. Fields are the atomic element of data for a Fillin. A Table is a data structure that contains Entries at the intersection of every Row and Column. Entries are the atomic element of data representation for a Table. Every Form has a corresponding Form Master which describes the Form's logical and physical (i.e. print/display) structure. These Form Masters are kept in either an Aide's or in the Office's or in the Organization's Form Master Book. Procedures are kept in an Aide's or in the Office's or Organization's Procedure Book. Procedures and Form Masters can be bound together as an Application and kept in the Application Book of an Aide or in the Office or Organization Application Book.

Fields and Entries contain data of specified type and format. The data types are whole number, decimal number, money, date, time and characters. There may be further restrictions on the set of values that a Field or Entry may contain.

A Document or Memo may include, in addition to text, directives on how it is to be displayed or printed.

2.2 Object Names

Every object is named. A name consists of the object's Class, a TYPE and an ID. A reference to an object by name is structured as follows:

TYPE Class ID

The object's type and id are character strings having the following characteristics: they are either space or medial underline delimited groups of capital letters and numbers or single-quote delimited unrestricted character strings (within which single quotes are surrounded by single quotes). If the non-quoted form is used, leading, trailing and multiple embedded spaces are ignored. Object classes are written with an initial capital. Examples of object references are:

CUSTOMER Drawer 1978
ENGINEERING Document 105_30_JAN_81
EMPLOYEE Folder 'Smith, J.M.'

An object whose type is an empty character string is said to be 'untyped'. An object whose id is empty is said to be 'with no id'. An object with both empty type and empty id is said to be 'unnamed'. Certain objects, because they are unique (with respect to a given location), do not require 'given' names; they can remain unnamed (e.g. an Aide's Desk). Objects can also have aliases. Within BUSINESS, some unnamed/untyped objects are almost always referred to by their aliases (e.g. the unnamed

Pile = the work Pile). An object's name and name components (type and id) and class are considered to be part of a collection of attributes of the object and as such are ascertainable for use within a BUSINESS Procedure (e.g. the id of CUSTOMER Drawer 1980 = "1980").

2.3 Referencing Objects

A reference to an object is either an explicit reference by name or a relative reference. A relative reference specifies an object's position with respect to the ordering at a given location. Objects are maintained in different orders at different locations. For example, Folders in a Drawer are ordered alphabetically by the name of the object while Forms within a Folder are ordered by any desired attribute including insertion order. If objects are ordered by name, they are first ordered by class then by type within class and then by id within type. Fillins, Tables, Rows, Columns, Fields and Entries are not ordered by type. Within each type, however, they are identified and ordered by ordinal number (e.g. AMOUNT Field 3, LINE Row 29). Examples of relative reference are:

the: the unique or only object, object of a given class or object of a given class and type at a location: the CUSTOMER Drawer

the last: the last object, object of a given type or object of a given class and type in an ordering: the last COMPANY Folder

the current: the current object, object of a given class or object of a given class and type of a 'go through' instruction: the current ORDER Form

the selected: the unique object, object of a given class or object of a given class and type at a location that has been singled out by a 'select' instruction: the selected CLIENT Folder.

all: a set of objects of the same type and/or object class. If 'all' is used, the object class is pluralized: all CLIENT Folders

O1 through O2: an (inclusive) range of objects (i.e. all objects including O1 upto and including O2 in the ordering): EMPLOYEE Folder JONES through EMPLOYEE Folder SMITH

A reference to an object must also contain as much information as is necessary to unambiguously specify a unique object or set of objects at a location. Certain locations are assumed if none are specified explicitly in a reference or implicitly through an 'assume' instruction. Some of these assumed locations are:

```
Work Area, Pile, Pad,: the Desk
Form, Memo, Document: the Work Area
Fillin,Table: the Pad
Field: the unnamed Fillin
Entry,Row,Column: the unnamed Table
Folder: the work Pile
Drawer: the main File Cabinet
```

If the class is omitted in a reference, Field is assumed. The class 'Thing' can refer to any other object class. References can be indirect via a computed type, id, class, complete name or relative position number within an ordering:

```
the PO Form (whose id equals the ORDER NUMBER)
the Drawer (whose type equals the CLIENT,
 whose id equals the YEAR)
the Document (whose name equals the name of the
 current Document in the Work Area)
the PO Form (whose number equals the TOTAL minus 5)
```

or derived from Indices:

```
the CLIENT Form (listed on the current Index Card
 in the PROSPECTS Index)
```

or (if the object is a Form) specified in terms of a condition that must be satisfied by its data elements, some subset of which can be defined as 'keys' in the Form Master:

```
all PO Forms (where the ORDER CLASS equals "S")
```

Examples of complete references are:

```
the ORDER Form
all CLIENT Folders in the first CUSTOMER Drawer
 in the Desk
PO Form 373 in ORDER Folder ABC in CLIENT Drawer 1979
 in the main File Cabinet in the CHICAGO Office
```

2.4 Characteristics Of Objects And Locations

The Work Area.

The Work Area is the location where Forms, Documents, Memos, Form Masters and Procedures are examined and changed. Within the Work Area, objects are ordered by their 'arrival.' The most recent object (or object of a given type) to arrive in the Work Area is

referenced as the 'first' one.

Piles.

Piles combine attributes of heterogeneous, indexed-sequential files (i.e. files containing multiple record types), stacks, double-ended queues and random access files. An object can be placed on or removed from a Pile either on top, at the bottom or relative to any known object already on the Pile. Piles are are a location where arbitrary arranging (i.e. sorting) of objects such as Folders is performed. Objects are ordered on Piles from top to bottom. Piles do not need to be explicitly created (as do all other objects); they can come into existence when an object is placed on one. Piles are the target location for Folders when they are 'pulled' from a Drawer in a File Cabinet or Desk.

Indices.

Indices contain Index Cards. An Index Card contains references to other objects (i.e. their names and locations). Index Cards themselves have no individual identity except for their numerically relative position within an Index (e.g. Index Card number 5). Indices are used very much as they would be in a real office - to improve the performance of Procedures that require searches for information.

The Pad and the Board.

The Pad and Board are the locations used to hold temporary information needed or generated by Procedures. There is a stack-frame discipline for Procedure execution. Parameters are passed (by value) to the next page of the Pad which becomes the 'local' variable space of the executed Procedure. Upon finishing, values are returned to the executing Procedure by 'noting' them on the previous page of the Pad - the page associated with the executing Procedure. The page of the Pad assciated with the Procedure that finished is erased. The Board is intended for information used by a collection of related Procedures. It can be seen as the repository for 'global' variables.

The In Box.

Aides are permitted to transmit Folders, Forms, Documents and Memos among themselves. The In Box is the location used to receive objects transmitted by another Aide. Objects can not be directly referenced when they are in the In Box; the In Box must first be emptied to a Pile.

The Calculator, Calendar, Telephone and Clock.

An Aide uses the calculator to perform all the calculations

specified in a Procedure. It is not directly referenced as an object in the language; its use is implicit. An Aide has a Calendar on his Desk. The Calendar has two functions. An Aide uses the Calendar to obtain the date (as a datum) and it is used to make appointments. (An appointment is a "reminder" to execute a Procedure or run an Application at a specified time during a given day.) The Aide's Desk has a telephone with which the Aide can call or reply to another Aide. An Office has a wall Clock from which an Aide can obtain the time of day (as a datum).

Archives and the Waste Disposal.

Archives are objects that contain Drawers, Folders, Form Masters and Procedures. To be used they must be 'installed' in an Office. The Waste Disposal is the location where discarded objects are moved. It has the characteristics of a traversible stack. Any object in the Waste Disposal can be retrieved up until the time a dispose instruction is issued.

3.0 PROGRAMMING IN BUSINESS

The programming model inherent in BUSINESS is based on a dialogue between a user and one of the Aides within an Office. A user instructs an Aide within the context of the office model. To the user this means specifying and writing Procedures that will be executed by an Aide. In addition, "working groups" of Aides can be established by ordering them (through the Aide with which a user is dialoging) to execute Procedures.

Communication with Aides is established by summoning them to their Desks. When work is completed, the summoned Aide is dismissed (i.e. they leave their Desks). Only one user can dialogue with an Aide at any one time.

3.1 Language Characteristics

Statements in BUSINESS are composed of one or more instructions. Every instruction in the language represents a plausible physical action within the office model. An instruction contains a single verb that specifies the intended action of the instruction. For clarity, an instruction always begins with its verb. Secondary or qualifying actions of an instruction are specified by clauses usually headed by a present participle. With a few exceptions, every instruction can be written so as to read like a sentence in English. An instruction starts with an initial capital and ends with a period. Parenthesized clauses and phrases are used to set off required, possibly detailed specifications in as natural a way as possible.

Instructions are organized into Procedures and Routines within Procedures. (Procedures and Routines are named objects within the office model.) The written representation of a

Procedure is structured as follows: If a statement is subordinate to an instruction it is written below and indented 'sufficiently' (at least five spaces) from the instruction to which it belongs or (less frequently used) it starts with 'Begin' and terminates with 'End'. A Procedure consists of a name declaration, (optional) parameter declaration and Procedure attributes, a body consisting of a single statement and an (optional) collection of Routines. For readability, a line is skipped between an instruction and its subordinate statement and at the end of its subordinate statement. (However, multiple skipped lines can be written as a single line.) In addition, a long instruction can be continued on succeeding lines. By convention, the second and succeeding lines of an instruction are indented one space. A Routine consists of a Routine name and a body consisting of a single statement. All Routines follow the body of the Procedure. Any instruction can be labelled. If an instruction is labelled, by convention, the label appears on its own line directly above the instruction associated with the label. Also by convention, a line is skipped before an instruction with a label. Procedure and Routine names and labels are always followed by a colon.

There is an implicit binding in a Procedure of the Field, Entry, Row, Column, Fillin and Table names of a given type of Form and their associated attributes through the use of Form Masters. Form Masters can have sub-schemas called Views which can be used to restrict information access or specify alternate print/display structures. If Form Masters other than the Aide's are to be used, they are specified in the Procedure attributes by specifying their location (i.e. which Form Master Book they are containted in). If a View of a Form Master is used it to must be specified either in the Procedure attributes or through a 'use' instruction. An equivalent declaration of the logical schema and display/print structure of a Form can also be specified in the Procedure attributes. Any "global" data on the Board that is used is also specified in the Procedure attributes. The Procedure attributes also (optionally) specify if the Procedure is an initial (i.e. 'main') Procedure who its author is and when it was created and last modified.

BUSINESS is a structured language; it is organized by Procedures and by powerful control constructs within Procedures. Unrestricted use of the goto-equivalent instruction ('continue at') is not permitted.

BUSINESS makes assumptions about the nature of communication with a user (assumed to be via VDT and keyboard). The VDT is assumed to contain three 'windows':

display: Forms, Documents, Memos, Form Masters and Procedures are shown to a user via this window.

dialogue: Any information to be obtained from or supplied to a user extrinsic to Forms, Documents, Memos, Form Masters and Procedures (excluding system status information)

```
Procedure name:

(parameter specification & Procedure attributes)

     instruction-1.
     instruction-2.

          instruction-2.1.
          instruction-2.2.
               .
               .
               .
     instruction-3.
      continuation of instruction-3.
          .
          .
          .
label of instruction-k:
     instruction-k.
          .
          .
          .
     last instruction of Procedure body.

Routine-1 name:

     instruction-1.
     instruction-2.
          .
          .
          .
     last instruction of Routine-1 body.
          .
          .
          .
Routine-k name:
          .
          .
          .
     last instruction of Routine-k body.
```

Figure 3 - Written Representation of a Procedure

is communicated via the dialogue window.

choice: Any information to be supplied by a user which can be formulated as a set of discrete choices can be obtained via the choice window.

It is convenient to think of BUSINESS instructions as falling into a hierarchical arrangement based on the principal verb. Figure 4 illustrates one such arrangement. In BUSINESS instructions can only exist within named Procedures. In practice, an implementation can allow a single, unnamed Procedure to be created and executed "directly." Direct execution can be thought of as the use of an extended operating system command language. BUSINESS uses a rich set of defaults to overcome some of its inherent verbosity. Among these are the assumed locations for objects previously mentioned.

3.2 The Instructions

Only the instructions in the programming group will be fully described in this section.

3.2.1 The Filing Group -

The instructions in the filing group implement traditional concepts of multi-user resource allocation applied to the filing system within the office model.

Pull, File.

The contents of a Folder cannot be modified unless the Folder is 'pulled' from a Drawer. In the office model, the result of pulling a Folder is that it moves from its original location to a Pile on the Desk of the Aide who has been instructed to pull it. Filing a Folder results in the Folder moving from a Pile to a Drawer where it now becomes available to other Aides in the Organization.

Examples: Pull the EMPLOYEE Folder in the COMPANY Drawer.
File the current CLIENT Folder on the work Pile in the OLD CLIENT Drawer in the Desk.

Hold, Release.

If a Folder is needed only for the examination of its contents, but it must be guaranteed that the Folder be present and unchanged during the examination, it can be held. Releasing is the complementary operation. More than one Aide can hold a Folder at the same time.

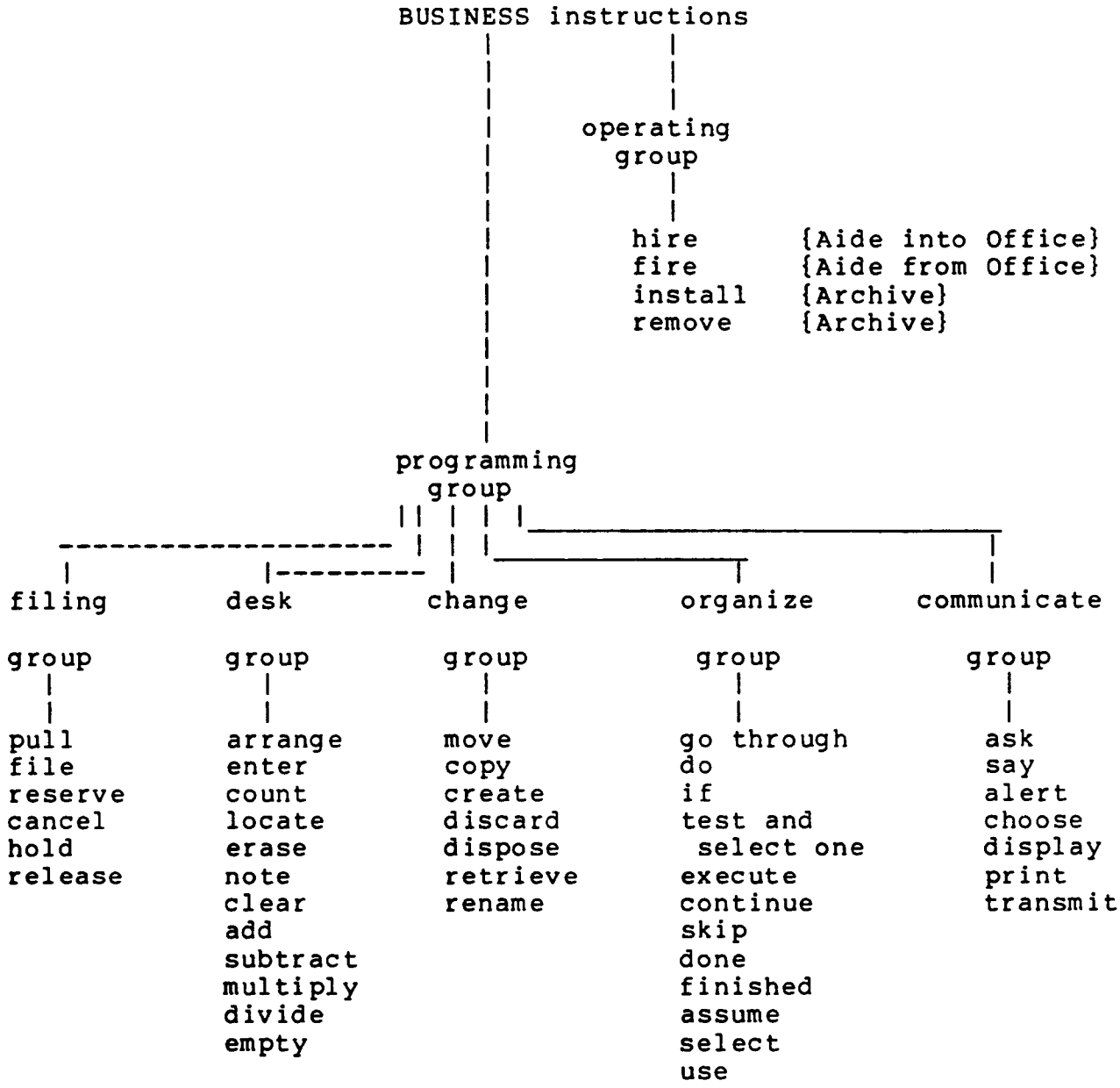


Figure 4 - BUSINESS Instruction Hierarchy

Examples: Hold ORDER Folder 1980 in the RECENT ORDER Drawer
in the 1980 File Cabinet in the NEW ORLEANS Office.
Release all PRODUCT Folders in the PRODUCT Drawer.

Reserve, Cancel.

An integral data set may be required to be present for the duration of some activity. An example of this is a payroll. The reserve instruction guarantees that a File Cabinet, set of File Cabinets, Drawer or set of Drawers will contain all the Drawers and/or Folders belonging to them when the reservation is granted and that none of the reserved resources can be modified by another Aide for the duration of the reservation. A reservation of a Drawer will not be granted unless all pulled Folders are filed and all held Folders are released. Cancel is the complementary instruction.

Examples: Reserve the EMPLOYEE Drawer in the main File Cabinet.
Cancel the reservation of the EMPLOYEE Drawer.

3.2.2 The Desk Group -

The instructions in the desk group refer to actions that an Aide performs at (or in the near vicinity of) his Desk.

Arrange.

This instruction is used to sort and/or merge objects on a Pile or set of Piles, the Rows or Columns of a Table and the contents of Fields.

Examples: Arrange all ORDER Forms on the work Pile
by the AMOUNT Field.
Arrange all CREDIT Forms on all CREDIT Piles
by the CREDIT TOTAL times the CREDIT RATIO
ordering smallest to largest
merging to the FINAL CREDIT Pile.
Arrange all LINE Rows of the ITEM Table
by the QUANTITY Column.

Enter, Note.

The enter instruction is a generalized "assignment" instruction. It two forms:

Enter <object>.

and

Enter <object> as <information>.

In the first form, since no information is actually supplied to

enter into the object, the Aide assumes that the user will supply it. This form of the enter instruction causes the activation of an (interactive) editor for that object class to mediate the entering. In the second form, either an expression or commensurate object is supplied. Data is matched on a name basis through the use of a common data dictionary. Objects must be located in the Work Area to be entered. The note instruction is equivalent to the enter instruction but applies only to objects on the Pad or the Board.

Examples: Enter the PURCHASE ORDER Form.
Enter the PRICE of the PRODUCT Form as the COST marked up by 10 percent.
Enter the current DETAIL Row of the SUMMARY Table of ESTIMATE Form FINAL as the PART Fillin of the PART DESCRIPTION Form.
Enter the Entry (on the current LINE Row, under the PRICE Column) of the ITEM Table of the current ORDER Form as $500.00.
Note the MAXIMUM PRICE as $999.99.

Locate, Count.

The locate instruction is used to ascertain and note the status attribute of some object within the office model. The count instruction is used to count objects at a given location. If not otherwise specified, the result of a locate instruction is noted on the Pad in the STATUS Field, the result of a count instruction in the COUNT Field. The status of an object depends on the class of the object. For example, a Folder can be "present", "unknown", "held" or "pulled". A Drawer can be "present", "unknown", "reserved", or "incomplete" (i.e. missing some pulled Folders).

Examples: Locate the PURCHASE ORDER Form (whose id equals the ORDER NUMBER) in the ORDER Folder on ORDER Pile 10 noting the status as the ORDER STATUS on the Board.
Count all DRUG REQUEST Forms in the PATIENT Folder (whose id equals the PATIENT NUMBER of the PATIENT INFORMATION Form).

Erase.

Erase removes the information 'inside' an object. After an object has been erased it is considered to be "empty".

Examples: Erase the PURCHASE ORDER Form.
Erase all SUBTOTAL Fields on the Board.

Empty.

The empty instruction is used to move the contents of an Aide's In Box to a Pile. (Objects cannot be referred to in the In Box). The order of the objects in the In Box is maintained on the Pile.

Examples: Empty the In Box onto the work Pile.

Clear, Add, Subtract, Multiply, Divide.

These instructions are abreviated forms of the enter and note instructions. If no Field or Entry is indicated, the RESULT Field on the Pad is assumed. The clear instruction sets a whole number Field or Entry to Ø, a decimal number Field or Entry to Ø.Ø and a money Field to $Ø.ØØ.

Examples: Add 1 to the RUNNING TOTAL.
Multiply the GROSS MARGIN by 1.5.
Clear the CREDIT SUBTOTAL.
Subtract 1 from the last TO DO Field of the SUBMIT Form.
Divide the AMOUNT by .ØØØ3.

3.2.3 The Change Group -

The instructions in the change group manipulate objects within the office model.

Move, Copy, Rename.

The move instruction moves an object (or set of objects) from one location to another. The copy instruction makes an identical copy of an object (or set of objects) and moves the copy to the specified location. The copy instruction can rename the copy of an object, the move instruction can not. The rename instruction allows a user to change the name of an object.

Examples: Move the PO Form (whose id equals the ORDER NUMBER) in the ORDER Folder on the REQUEST Pile to the Work Area.
Move ORDER Folder AlØØ through ORDER Folder D999 in CLIENT Drawer 1979 to ORDER Archive 1979.
Copy the CLIENT Form (whose id equals the id of the CLIENT Form listed on the selected Index Card in the PROSPECTS Index) in the CLIENT Folder on the work Pile to the Work Area.
Copy ENGINEERING Drawer ED_Ø7 in the ENGINEERING File Cabinet in the NEW YORK Office to the ENGINEERING File Cabinet in the SAN FRANCISCO Office.
Rename ORDER Form 3344H to ORDER Form 3344HO.

Create, Discard, Dispose, Retrieve.

New objects in the office model are created with the create instruction. Objects are discarded via the discard instruction. When an object is discarded it is moved either to the Office's or to the Aide's Waste Disposal. The dispose instruction "shreds" everything in a Waste Disposal. The retrieve instruction allows

objects that have been discarded to be retrieved from a Waste Disposal up until they are disposed.

```
Examples: Create CLIENT Drawer 1980.
          Create the ORDER Form (whose id equals
           the NEXT ORDER NUMBER).
          Create the COST Table on the Pad with 52 WEEK Rows,
           the DEPARTMENT Column (characters,
                                  format is "3A",
                                  value is in
                                   ("SLS", "G&A", "ENG")),
           the ITEM Column (whole number),
           the COST Column (money),
           the OVERHEAD Column (decimal number),
           and the TOTAL Column (money).
          Create Index Card listing the current CLIENT Form
           in the PROSPECTS Index.
          Discard all ORDER Forms on the PROCESSED Pile.
          Dispose the contents of the Office Waste Disposal.
          Retrieve EMPLOYEE Folder SMITH in the
           Office Waste Disposal to the work Pile.
```

Replace, Exchange.

The replace instruction replaces an object (or set of objects) at a location with an object (or set of objects) at another location. It can also replace with a copy of an object (or set of objects). The exchange instruction swaps the locations of objects.

```
Examples: Replace the current PO Form on the work Pile
           with a copy of the PO Form in the Work Area.
          Exchange the CLIENT Drawer in the main File Cabinet
           with the CLIENT Drawer in the WORKING File Cabinet.
```

Combine.

The combine instruction allows merging the contents of a set of objects.

```
Examples: Combine all ORDER Folders on the PROCESSED
           Pile with the COMPLETED ORDER Folder
           on the work Pile.
```

3.2.4 The Organize Group -

The organize group of instructions specify the control structure of Procedures. The 'go through', 'do', 'if', 'select', 'assume' 'test and select one' and 'use' instructions allow a subordinate statement.

Go through.

The go through instruction is the most important control construct in BUSINESS. Through it, an Aide is instructed to sequence through a set of objects at a specified location in the object order at that location (or optionally, in reverse order, by name or explicit list). The object under consideration is referred to as 'the current' one. In addition, the sequencing can be conditionalized (i.e. initiation and termination conditions can be specified) and certain object manipulations useful in programming can be specified. In going through a set of Forms in a Folder (for example), if the clause "examining each one" is part of the instruction, the Aide will make a copy of the current Form at its source location and move the copy to the Work Area. Upon termination of one iteration, the copy in the Work Area is discarded. The copy of the Form in the Work Area can also be referred to as 'the current' one at its location. If the clause "changing each one" is part of the instruction, the copy in the Work Area replaces the Form at the source location upon the completetion of one iteration. In this case, the user is relieved of handling the boundary conditions (initialization, termination) of a typical 'record update' procedure - the go through instruction handles it. Similiar actions can be associated with Folders ("pulling and filing each one", "holding and releasing each one") and Drawers and File Cabinets ("reserving and cancelling each one").

Examples: Go through all ORDER Forms (where the ORDER CODE equals "S") in the ORDER Folder on the PROCESS Pile examining each one.
Go through all WEEK Rows of the SUMMARY Table on the Board.
Go through all Index Cards in the PROSPECTS Index.

Do.

The do instruction specifies a conditional iteration control structure. The general form of the instruction is as follows ('[]' enclose an optional construct):

```
Do [the following]
   [if <condition>]
   [while/until <condition>]
   [repeating from <expression>
       [to <expression>] [by <expression>]
   [noting the repetition as <Field/Entry>]]
```

In addition, Routines are executed using the do instruction:

```
Do <Routine name> [if.....]
```

Examples: Do CHECK OUT Routine AMOUNT if the BALANCE of the selected CREDIT MEMO Form is greater than $5000.00.

```
Do the following repeating from 1 to 100 by 2
 noting the repetition as I.
```

If.

The if instruction specifies a condition and a subordinate statement to be executed if the condition is true. Conditions are composed of logical combinations of tests. The logical operators 'and', 'or' and 'it is not true that' or 'not' are allowed. An if instruction also allows an optional collection of 'else if' and 'else' parts.

```
<if instruction> ::= if <condition>
                     then <statement>
                [else if <condition>
                     then <statement>]
                        :
                        :
                [else if <condition>
                     then <statement>]
                [else <statement>]
```

Both an 'else if' and 'else' part allow a subordinate statement. If the condition of the if instruction is false and a collection of else if parts is included, each condition is tested. If one is true, its subordinate instruction is executed. If an else part is included and the condition of no other part is true, its subordinate statement is executed. In the written representation of an if instruction, the word 'then' substitutes for the terminating period and is placed on its own line at the same indentation level as the subordinate statement.

Examples:
```
If the GROSS MARGIN of the SUMMARY Form
 is less than $100000.00
     then <statement>
If the TITLE Field of the CLIENT Form
 contains "Doctor" and the INCOME on the Board
 is between $25000.00 and $50000.00
     then <statement>
Else if the TITLE Field of the CLIENT Form
 contains "Mr"
     then <statement>
Else <statement>
```

Test and select one.

The test and select one instruction allows a subordinate statement consisting of 'if' instructions which are selectively executed. Each subordinate 'if' instruction is executed in sequence. If a condition of one of the 'if' instructions is true, its subordinate statement is executed. Any remaining 'if' instructions within the statement are ignored. 'Otherwise.' is used as an abbreviation for an 'if' instruction whose condition is always true. A data expression to be used as a test value can optionally be specified.

Examples: Test and select one.

```
If the VALUE is equal to "NEW"
     then

     Execute A/P Procedure ORDER.

If the VALUE is equal to "OLD"
     then

     Execute A/P Procedure CANCEL.

Otherwise.

     Execute A/P Procedure MENU.
```

Execute.

The execute instruction allows the execution of Procedures. When an execute instruction is executed and no location is supplied for the Procedure, an Aide first searches his Work Area, then his Procedure Book, then the Office Procedure Book and finally the Organization Procedure Book for the Procedure. (This allows for the development of 'local' experimental versions of Procedures while not affecting the behavior of production versions.) Parameters are specified positionally or by name. If the data type of a parameter is not supplied, characters is assumed. The execute instruction and a Procedure header have the following forms (*> and <* enclose a list of items):

```
Execute <Procedure name>

  [using *>[*><expression> [for <Field>]<*]
           [*><Fillin>
             [for <Fillin>]
               [*>with <Field> for <Field><*]<*]
           [*><Table>
             [for <Table>]
                [*>with <Row/Column>
                    for <Row/Column><*]<*]]<*]
  [getting back *>[*><Field/Entry>
                       [from <Field/Entry><*]]
                  [*><Fillin>
                    [from <Fillin>
                       [*>with <Field>
                           from <Field><*]]<*]
                  [*><Table>
                    [from <Table>
                       [*>with <Row/Column>
                          from <Row/Column><*]]<*]<*]

<Procedure name> :

        [(
         [needs nothing
                *>[*><Field>
```

```
                    [(<data type>)]*<]
                 [*><Fillin>
                     [with *><Field>
                        [(<data type>)]<*]*<]
                  [*><Table>
                     [with *><Row/Coluumn>
                        [(<data type>)]<*]<*]]<*]
           [returns nothing
                    *>[*><Field>
                          [<(data type)>]*<]
                      [*><Fillin>
                          [with *><Field>
                             [(<data type>)]<*]<*]
                      [*><Table>
                          [with *><Row/Column>
                             [(<data type>)]<*]<*]<*]
           [uses nothing
                 *>[<Form Master/View reference>]
                   [<Form Master specification>
                   [<Global data specification>]<*]
           [assumes nothing
                    *>[<object-location assumptions>]
           [author is <literal>]
           [created on <date>]
           [last modified on <date> [at <time>]]
           [initial] {'main' Program}
           [safe] {non-interruptable}
         )]
```

The following sequence of events occurs:

1. If a 'using' clause is part of the execute instruction, for each item in the list, any <expression> is evaluated, and the computed value is written on the next page of the Pad as the value of the Field (of the unnamed Fillin) specified, any Fillin (Table) with the specified subset of Fields (Rows/Columns) is created and the values of the corresponding Fields (Entries)are copied.

2. The Pad is turned to the next page. All information noted on the Pad will occur only on that page. Previous pages of the Pad are not accessible within the executed Procedure.

3. The Procedure header is checked for a 'needs' clause. If one is found, the Aide verifies that any information needed is actually on the Pad (as a result of the 'using' clause) and matches the data type specification. Any assumed locations for objects or Views of Form Masters are in effect for the duration of Procedure execution unless overridden.

5. When the Procedure is finished, if a 'getting back' clause is specified, for each item on the list, the information indicated is noted on the previous page of the Pad (the one associated with the Procedure issuing the execute instruction) using the correspondences

specified in the 'returns' clause of the Procedure header.

6. The page of the Pad associated with the executed Procedure is erased.

7. The Aide flips back to the previous page of the Pad.

Examples: Execute the NEW VENDOR Procedure using the RESPONSE for the VENDOR NUMBER getting back the VENDOR STATUS.

Run.

The run insruction specifies the activation of an Application. An Application is a 'compiled' collection of Procedures and Form Masters. One Procedure among them is the 'initial' or main Procedure. Execution starts with this Procedure. There is no passing of values into or out of an Application via parameters.

Examples: Run the ACCOUNTS RECEIVABLE Application in the Office Application Book.

Continue, Skip, Done, Finished.

The continue instruction allows direct transfer of control within a Procedure or Routine to an instruction directly above or below or to an instruction going outwards along the control tree. The instruction is specified by its label. If no instruction is indicated execution continues normally. The skip instruction allows the termination of one iteration of any enclosing go through or do instruction. The done instruction terminates a superior instruction (go through, do, test and select one, select, assume or use) or Routine. The instructions are referred to by label if they are other than the immediately enclosing instruction. The finished instruction a pending execute instruction.

Examples: Skip.
Done with CREDIT Routine VERIFY.
Done with go through at FIND CUSTOMER.
Continue at NEW CUSTOMER.
Finished with this Procedure.
Finished with the VALIDATE Procedure.

Order, Summon, Dismiss.

The order instruction allows a user to issue instructions to an Aide other than the one with which he is dialoging. Use of this instruction permits the formation of a "working group" of Aides each handling some part of a total task. The interlocutor Aide can either be instructed to wait for the ordered Aide to finish or immediately continue after the order has been issued. The ordered Aide is summoned to his Desk (if possible) and dismissed

upon completion of the order unless this has been done explicitly via the summon and dismiss instructions. Summoning an Aide gives the summoner exclusive access to that Aide. Dismissing an Aide releases him.

Examples: Summon the ACCOUNTS PAYABLE Aide.
Order the POST PROCESSING Aide to execute the CHECK ORDER Procedure in the Office Procedure Book.
Dismiss the ORDERING Aide.

Wait.

The wait instruction allows an Aide to be instructed to wait for some event to take place to invoke an associated action following the event(s) and to continue execution after that event (and action) has occured. Among the events that an Aide can wait for and actions that can be taken are:

1. Another Aide or group of Aides to be dismissed (i.e. to have their status change from "summoned" to "dismissed"). When the Aide or Aides become available, they can be summoned. An Aide can also wait for another Aide or group of Aides that has been given an order to finish.

2. A Drawer, set of Drawers, File Cabinet or group of File Cabinets to become available (i.e. to have their status change from "reserved" or "incomplete" to "present"). When the requested resources become available they can be reserved.

3. A Folder or set of Folders to be returned (i.e. to have their status change from "pulled" or "held" to "present"). When the requested Folder or Folders become available they can be pulled or held.

4. The In Box to have something in it (i.e. to have its status change from "empty" to "full"). When the In Box does contain something it can be emptied to a Pile.

5. A given amount of time to pass.

6. A specific time of day and/or date.

Examples: Wait for the POST PROCESSING Aide to finish.
Wait to summon the CHECK ORDER Aide.
Wait to pull the selected EMPLOYEE Folder.
Wait for 20 seconds.
Wait for 16:00.
Wait to empty the In Box.

Make, Break.

The make appointment instruction allows a user to initiate the

execution of a Procedure or running of an Application at a given time and day. If an Aide is busy, the appointment is queued on a FIFO basis. The user can also instruct an Aide to break an appointment that has been made. Appointments are maintained in the Aide's Calendar.

Examples: Make appointment at 17:00 on 2/14/81
to run the VALENTINES DAY Application.
Break appointment at 9:ØØ.

Select, Assume, Use.

The select instruction allows a user to instruct an Aide to "single out" an object at a location and refer to it as the selected object within a subordinate statement. The selection can also be conditionalized. The assume instruction instructs the Aide to assume a location for an object or set of objects within a subordinate statement. The use instruction allows the specification of a View of a Form Master to be used within a subordinate statement

Examples: Select the PURCHASE ORDER Form (whose id equals
the ORDER NUMBER).
Select the LINE Row (whose id equals the TARGET)
of the LINE ITEM Table of the ORDER Form
Assume the location of the CONVERSION Table is
the current MONEY Form in the Work Area.
Use the SHIPPING View of the ORDER Form Master.

3.2.5 The Communicate Group -

The instructions in the communicate group specify different forms of communication between a user and an Aide, between Aides and between an Aide and a receiver/transmitter of information "outside" the Organization.

Ask, Say, Alert.

The ask instruction allows an Aide to obtain a response (to a question) from a user or other Aide via the dialogue window. If not otherwise specified, the user's response is noted in the RESPONSE Field on the Pad. The say instruction allows an Aide to communicate information directly to a user via the dialogue window. The alert instruction allows an Aide to get the users attention (through some audible and/or visual signal).

Examples: Ask "What is the client's name?" noting the response as
the CLIENT NAME of the CLIENT Form.
Say "The client's name is " followed by
the CLIENT NAME of the CLIENT Form.
Alert the user.

Call, Reply.

The call instruction allows a user to have his Aide initiate a dialogue with another Aide. In the office model, it is equivalent to making a telephone call to another Aide. The dialogue window is used to contain the 'conversation'. The reply instruction allows a user to "pick up the phone."

Examples: Call the PURCHASING Aide in the DENVER Office.
Reply.

Choose.

The choose instruction allows the Aide to present a user with a set of choices via the choice window from which the user is required to choose one. Unless otherwise specified, the choice (number) is noted in the CHOICE NUMBER Field and the choice text is noted in the CHOICE Field.

Examples: Choose 1 for "Yes", 2 for "No", 3 for "Maybe".
Choose among all OPTION Fields of the SALES MENU Form saying "Choose a sales option, please.".

Display, Print.

The display instruction uses the display window to show a user the contents of objects. The print instruction performs the same function but using a device that yields "hard copy". In general, 'contents' means the names of the set of objects and/or values "at the next level" (e.g. Displaying a Drawer will display the names of the Folders in the Drawer.) The display format of a Form is specified via the Form Master for the given type of Form. In addition, when necessary, direct device control and picture-oriented formatted display/print are also available.

Examples: Display the current ORDER Form {in the Work Area}.
Display the contents of the Pad.
Print the names of all ORDER Forms on the work Pile.
Print the contents of the main File Cabinet.
Print new page, on line 10 at position 30 show "The largest value found was" followed by the value of the MAXIMUM FOUND Field on the Board as "NNNNN".
Print the CLIENT Form using the main printer.

Transmit.

The transmit instruction allows a user to instruct an Aide to transmit information to other Aides. When information is transmitted to an Aide it goes into the (target) Aide's In Box.

Examples: Transmit all ORDER Forms on the COMPLETED Pile to the POST PROCESSING Aide.

3.3 Programming Example

In this example, an account summary for a customer must be produced. This account summary combines the total debits and credits for the customer to produce a current account balance. Four types of Forms are used:

1. CUSTOMER INFORMATION: (where the id equals the {customer's} ACCOUNT NUMBER)
   which contains (among other information):
   the {customer's} NAME Field (characters)
   the {customer's} BALANCE Field (money)

2. CREDIT: (where the id equals an arbitrary sequence number)
   which contains (among other information):
   the CREDIT AMOUNT Field (money)

3. DEBIT: (where the id equals an arbitrary sequence number)
   which contains (among other information):
   the DEBIT AMOUNT Field (money)

4. ACCOUNT SUMMARY: (where the id equals the {customer's} ACCOUNT NUMBER)
   which contains:
   the CREDIT TOTAL Field (money)
   the DEBIT TOTAL Field (money)
   the CURRENT BALANCE Field (money)
   the ACCOUNT NUMBER Field (whole number)
   the CUSTOMER NAME Field (characters)

The underlying information structure:

```
the main File Cabinet
  which contains:
        the CUSTOMER Drawer
          which contains:
             all CUSTOMER Folders (where the id equals
                   the {customer's}
                   ACCOUNT NUMBER)
               Each CUSTOMER Folder contains:
         the CUSTOMER INFORMATION Form
         all CREDIT Forms
         all DEBIT Forms
                {for that customer}
```

The ACCOUNT SUMMARY Procedure peforms the following steps:

1. The Aide asks the user for a (customer's) ACCOUNT NUMBER. If the ACCOUNT NUMBER is empty, the Aide finishes.

2. He verifies that a CUSTOMER Folder with that ACCOUNT NUMBER is present in the CUSTOMER Drawer.

3. If the Folder is unknown, he informs the user

and continues with step 1.

4. Otherwise the Aide waits to hold the Folder.

5. Once the Folder has been held, the Aide creates an empty ACCOUNT SUMMARY Form in the Work Area and goes through all the credits and totals them up then goes through all the debits and totals them up.

6. The Aide computes the CURRENT BALANCE by subtracting the DEBIT TOTAL from the CREDIT TOTAL and adding it to the BALANCE of the CUSTOMER INFORMATION Form.

7. The Aide transfers the customer's NAME to the ACCOUNT SUMMARY Form from the CUSTOMER INFORMATION Form he copied to the Work Area and also transfers the ACCOUNT NUMBER.

8. The Aide releases the customer's Folder.

9. The Aide displays the ACCOUNT SUMMARY Form, moves it to a Pile, discards the (copy of) the CUSTOMER INFORMATION Form in his Work Area and continues with step 1.

ACCOUNT SUMMARY Procedure:

```
(initial
 needs nothing
 returns nothing
 assumes nothing
 uses the CUSTOMER INFORMATION Form Master
       in the Office Form Master Book,
      the CREDIT Form Master in the Office Form Master Book,
      the DEBIT Form Master in the Office Form Master Book,
      the ACCOUNT SUMMARY Form Master
         (which contains the CREDIT TOTAL Field
                           (data type is money
                            format is "$NNNNNN.NN"),
                          the DEBIT TOTAL Field
                           (data type is money
                            format is "$NNNNNN.NN"),
                          the CURRENT BALANCE Field
                           (data type is money
                            format is "$NNNNNNN.NN"),
                          the CUSTOMER NAME Field
                           (data type is characters
                            format is "30C")
          displayed as follows
                          on line 1 at position 30
                           show "Account Summary",
                          on line 2 at position 30
                           show "---------------",
                          on line 5 at position 3
                           show "Customer Name: "
                           followed by the CUSTOMER NAME Field,
                          on line 6 at position 3
                           show "Credit Total: "
                           followed by the CREDIT TOTAL Field,
                          on line 7 at position 3
                           show "Debit Total:  "
                           followed by the DEBIT TOTAL Field,
                          on line 8 at position 3
                           show "Current Balance: "
                           followed by the CURRENT BALANCE Field)
 author is "P.Miller")

      Do the OBTAIN ACCOUNT NUMBER Routine.
      Locate the CUSTOMER Folder
       (whose id equals the ACCOUNT NUMBER)
       in the CUSTOMER Drawer.
      Test and select one.

           If the STATUS is "unknown"
                then

                Say "No such customer Folder is present.",
                 "Check account number and reenter."
                Continue at the beginning of this Procedure.

           Otherwise.
```

```
                Wait to hold the CUSTOMER Folder (whose id equals
                 the ACCOUNT NUMBER) in the CUSTOMER Drawer.

           Select the CUSTOMER Folder (whose id equals the ACCOUNT
            NUMBER) in the CUSTOMER Drawer.

                Create the ACCOUNT SUMMARY Form (whose id equals
                 the ACCOUNT NUMBER).
                Go through all CREDIT Forms in the selected CUSTOMER
                 Folder examining each one.

                     Add the CREDIT AMOUNT of the CREDIT Form
                      to the CREDIT TOTAL of the ACCOUNT SUMMARY Form.

                Go through all DEBIT Forms in the selected CUSTOMER
                 Folder examining each one.

                     Add the DEBIT AMOUNT of the DEBIT Form to
                      the DEBIT TOTAL of the ACCOUNT SUMMARY Form.

                Copy the CUSTOMER INFORMATION Form in the selected
                 CUSTOMER Folder to the Work Area.
                Assume the location of all Fields is the ACCOUNT
                 SUMMARY Form.

                     Enter the CURRENT BALANCE as the BALANCE
                      of the CUSTOMER INFORMATION Form plus the
                      CREDIT TOTAL minus the DEBIT TOTAL.
                     Enter the ACCOUNT NUMBER as the ACCOUNT NUMBER
                      on the Pad.
                     Enter the CUSTOMER NAME as the NAME Field of
                      the CUSTOMER INFORMATION Form.

                Release the selected CUSTOMER Folder.

           Display the ACCOUNT SUMMARY Form.
           Move the ACCOUNT SUMMARY Form to the ACCOUNT SUMMARY Pile.
           Discard the CUSTOMER INFORMATION Form in the Work Area.
           Continue at the beginning of this Procedure.

OBTAIN ACCOUNT NUMBER Routine:

           Ask "Enter the customer's account number, please: " noting
            the response as the ACCOUNT NUMBER.
           If the ACCOUNT NUMBER is empty
                then

                Finished with this Procedure.
```

3.4 Results And Conclusions

In this paper we have presented an applications language based on a model of a paper office. We believe that languages that are problem-domain oriented aid (end-)users in solving problems in that they facilitate the translation of a solution formulated within the conceptual world already familiar to them into executable code. Initial results using BUSINESS to code standard financial application packages indicate between an 8 and 10 times reduction in the amount of required code with a commensurate improvement in total developement time. We also believe in the DIALOGUE model of programming. Program development should be viewed as a structured communication between ACTIVE entities. In this way, programming comes to be seen as a form of instruction or tutoring rather than mechanical manipulation.

We believe that the office environment is a very rich one that has evolved over many decades. We recognize that, given the possibilities of new technology, it itself will change and evolve in people's minds. (In future office models, instead of a calculator, an Aide may have their own Work Station, for example.) "Electronic paper" having "magical" qualities may become so familiar to real office workers that office models such as the one presented here may safely be able to use these characteristics.

Finally, we believe that as these newer model-based programming concepts begin to be tested in the marketplace computer scientists will begin to better understand the requirements of end-users for structured and/or other disciplined programming methodologies in producing reliable software quickly.

4.0 APPENDIX 1 - OPERATORS AND TESTS

BUSINESS contains a full range of operators with which to form expressions and tests with which to form conditions. The detailed semantics of the operators and tests will not be described here (most are obvious). In general, there are restrictions that apply to the <term>s of each operator/test, however, coercion between values of different data types is implicit in the language. Among the current complement of operators are:

```
<term> (plus, +, minus, -, times, *,
       divided by, /) <term>
the remainder of <term> divided by <term>
       (<term> modulo <term>)
<term> percent of <term>
<term> discounted by <term>
<term> marked up by <term>
<term> rounded up by <term>
<term> rounded down by <term>
the position of <term> in <term>
```

```
        [following character <term>]
the number of (days, months, years)
        between <term> and <term>
the number of (minutes, hours)
        between <term> and <term>
the (day, month, year) of <term>
the (minute, hour) of <term>
the number of characters in <term>
the number of words in <term>
        [using <term> {as word delimiter}]
the number of lines in <term>
<term> in uppercase
<term> in lowercase
the largest of *> term <*
the smallest of *> term <*
(the negative of, -) <term>
character(s) <term> (through <term>) of <term>
word(s) <term> (through <term>)
        of <term> [using <term>]
line(s) <term> (through <term>) of <term>
<term> padded (left, right) with <
        character> to <term> places
<term> stripped of all (leading, trailing)
        <character> characters
the ordering equivalent of <term>
the (whole, dollar) part of <term>
the (decimal, cents) part of <term>
<term> followed by <term> {concatenation}
<term> preceded by <term> {reverse concatenation}
<term> (days, months, years) (before, after) <term>
<term> (minutes, hours) (before, after) <term>
the (whole number, decimal number, money,
     time, date, character)
     equivalent of <term>
<term> ([raised] to the power, ^) <term>
the natural logarithm of <term>
the logarithm of <term> [to the base <term>]
```

The current complement of tests are:

```
<term> (is, is equal to, equals, =,
        is not, is not equal to) <term>
<object> (is, is not) empty
<term> (is, is not) equivalent to a
        (whole number, decimal number,
         date, time, money)
<term> (is valid for, is not valid for)
        <Field/Entry>
<term> (has the format,
        does not have the format) <format>
<term> (has the format of,
        does not have the format of) <Field/Entry>
<term> (is a value in,
        is not a value in) *> term <*
<term> (is a value of, is not a value of) <Field/Entry>
<term> is (greater than, >,  less than, <,
           not greater than,
```

```
                not less than) <term>
<term> is (between, not between) <term>
<term> (follows, preceeds, does not follow,
        does not preceed) <term>
<term> (falls between,
        does not fall between) <term> {dates and times}
<term> (contains, does not contain) <term>
<term> (exactly matches, does not exactly match) <term>

<term> is (odd, even, negative)
```

5.0 ACKNOWLEDGEMENTS

The authors wish to acknowledge Christine Aquilino for her help in preparing the figures and Murray Ruben whose interest in developing an applications development environment for the non-programming entrepreneur prompted the entire BUSINESS language development.

6.0 REFERENCES

1. Lefkovits, et. al., A Status Report on the Activities of the CODASYL End User Facilities Commitee (EUFC), SIGMOD Record, Volume 2 and 3, ACM, August 1979.

2. Hammer, M. and Berkowitz, B., DIAL: A Programming Language for Data Intensive Applcations., Laboratory for Computer Science, Massachusetts Institute of Technology, January 1980.