Using box structures for definition of requirements specifications

by J. E. Odom

Box structures provide a stepwise refinement and verification methodology for information systems analysis and design. They are especially useful for recording and decomposing requirements specifications. The benefits of using the structures center around making the requirements clear to readers, helping to make the requirements complete, and providing an artifact that will enhance the traceability of the requirements. This paper describes the methodology of applying box structures and presents an example of their use in the definition of requirements specifications.

This paper addresses the problem of writing good requirements specifications that can be understood by potential customers of an information system as well as by the designer who will provide a solution to the problems that the requirements specifications define (see Requirement Specification Process in Figure 1). The characteristics of these requirements specifications will determine the success of the solution and how much rework will have to be done before it will satisfy customer needs.

The portion of the development process in Figure 1 shows the different processes and artifacts that are on both sides of the line separating the problem and solution domains. The artifacts that come out of the box structure methodology are the box definition graphics (BDG) and the box definition language (BDL). The BDG and BDL are used to describe the same model and can be used interchangeably. The BDG, along with supporting text, are very useful in communicating high-level concepts and overall system structure. The BDL is more useful when the design becomes more detailed. This paper will focus primarily on BDG but will use some BDL when describing an example problem.

Box structures are limited to three major structures: black box, state box, and clear box.[1] The clear box can contain the four basic procedural structures: sequence, alternation, iteration, and concurrent,[1] which are sufficient from a structured programming viewpoint.[2] These structures make up the entire set of basic structures necessary for the definition of any requirement. The structures can be described in graphical terms with BDG or in narrative terms with BDL.[1]

Figure 2 summarizes the three equivalent box structure views of an information system in a stepwise

Figure 1 Problem and solution domains

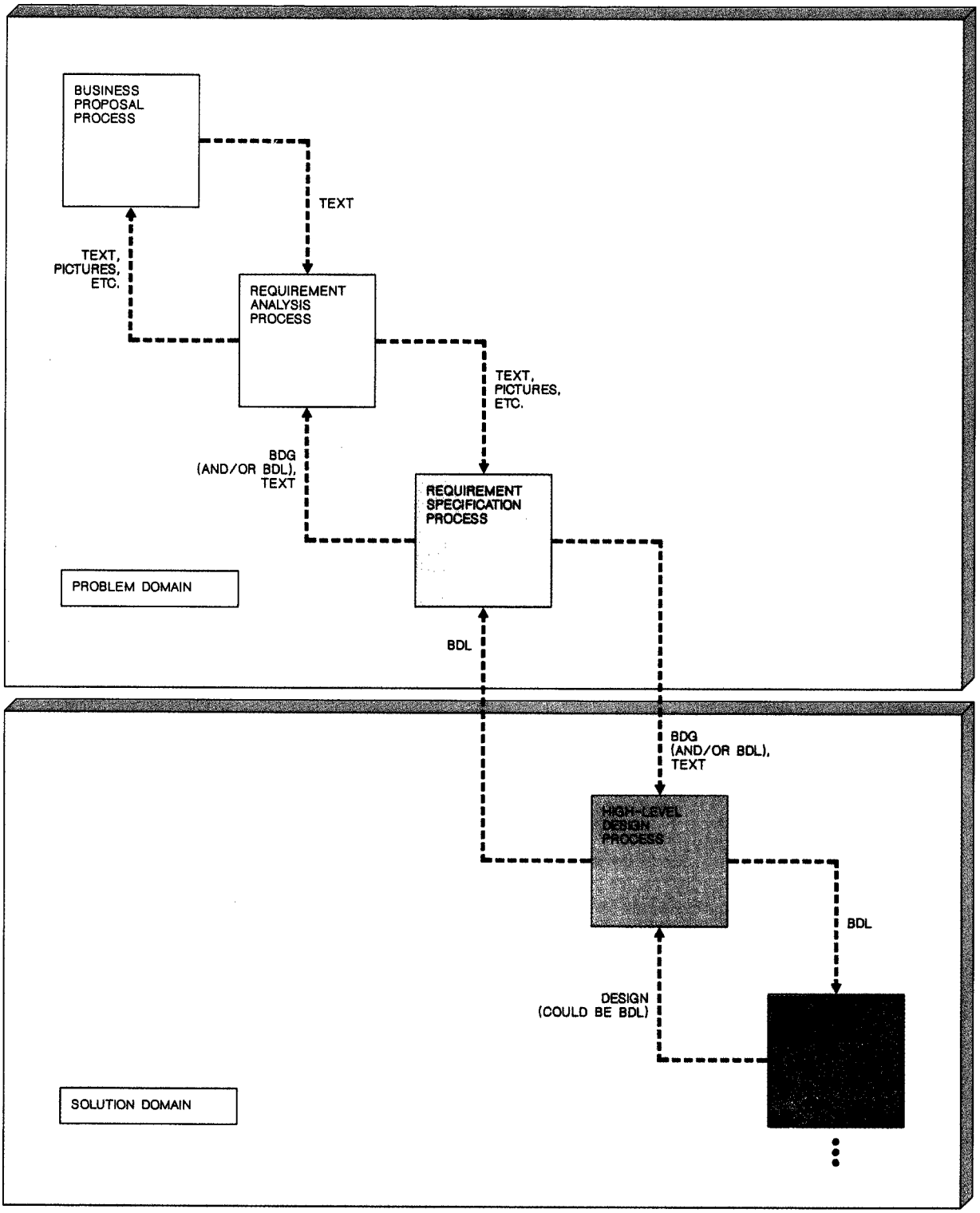
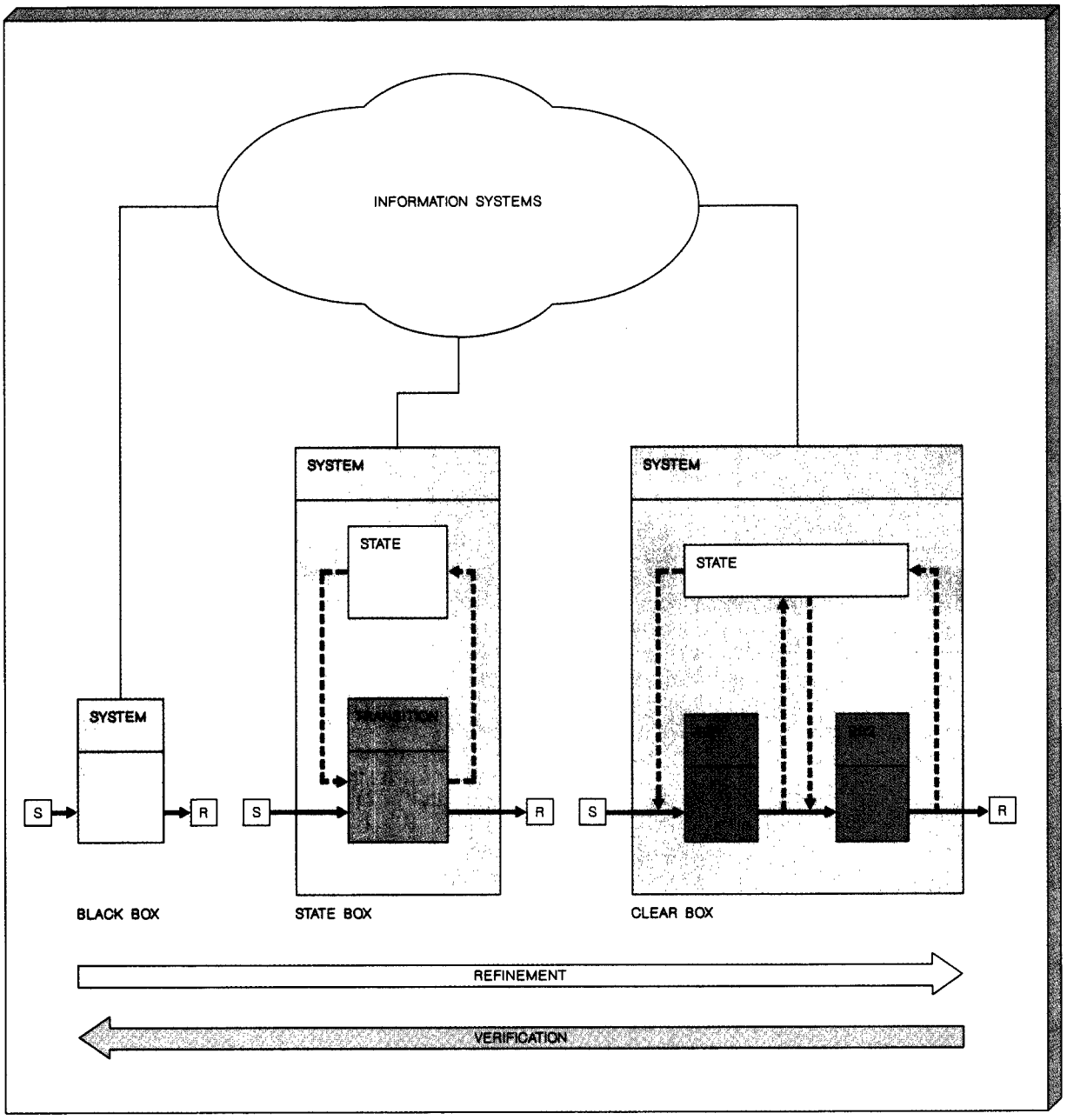


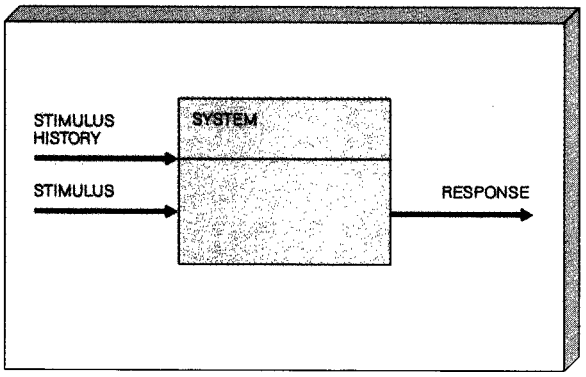
Figure 2 The three equivalent box structure views



refinement and verification methodology. Each box exhibits identical stimulus (or stimuli) and response behavior at increasing levels of detail.[1]

The use of box structures builds upon the idea that the definition of requirements in the form of requirements specifications is part of the problem domain as opposed to the solution domain. The definition of the requirements in this form is not intended to presuppose a design direction for the final product. Rather, it is intended to provide a method for establishing a precise definition of the problem that can

Figure 3 Black box view



be used in communicating the final form of the requirement. Box structures can be used to design the solution, but that is not within the scope of this paper.

Different types of problems will need different techniques for collecting and analyzing the requirements, but a single technique for documenting requirements would significantly improve human communication. A requirements methodology exists that provides an approach to collecting and defining underlying problems and to developing a prose functional specification.[3] In order that a design, programming, and testing organization can develop software that is accurate, easy to maintain, and is what the customer wants,[4] without too much rework, a standard technique for recording the requirements based on a model should be used. Box structures provide an approach to recording the requirements with enough detail and precision to accomplish this.

Developing the black box view of a requirement

The functional definition of black box behavior is:

(Stimulus History, Stimulus) → (Response)

When used with box structures, this definition of a black box is slightly different from that used in testing theory. A description of this definition is as follows:

"The black box is a mechanism that accepts stimuli, and for each stimulus produces a response before accepting another stimulus; furthermore, each response is uniquely determined by the history of stimuli accepted by the black box."[1]

This definition includes the consideration of a stimulus history, as we see in Figure 3. The stimulus history consists of all of the previous stimuli that have been received by the system prior to the stimulus being documented. For example, consider the requirement for a simple system that returns a one or a zero, alternating between them. The history of the stimuli received by the system is important because it determines what the response for this stimulus will be.

Although stimulus history conceptually contains a record of all previous stimuli, only a portion may be relevant. Also relevant is the initial condition of the system. In the previously mentioned example, if we know that the system initially returns a zero, we need only know how many stimuli have been received; if an even number, a zero will be returned, otherwise a one.

The starting point for the history of each stimulus should be carefully selected. Only enough stimulus history is needed to establish a single response for the stimulus. Once this point has been established, the history of each stimulus should be documented along with the resulting response.

The stimulus and stimulus history can cause several functionally different stimulus data transitions as they pass through the system. As an example, assume a user enters a query transaction in a personnel system which is supposed to display information about an employee. If the information was entered through a previous build transaction, the transition that occurs when the query transaction for that employee is entered is the display of information about the employee. If it has not been entered through a previous build transaction, the transition will be the display of some message indicating that no information is available on this employee. Each of these transitions must be documented because in this way the stimulus and response are related. These transitions are caused by user-view transactions or operations such as the build and query transactions mentioned in the example.

Transaction analysis must be performed at the black box level to determine whether a system will answer all the needs of the customer requirement. The first step in transaction analysis is to decide the context of the system. Context is defined as the user's environment where the system will reside and the changes it will invoke on that environment. Next, the context of the requirement is narrowed by deter-

mining if all of the user-view transactions, in terms of stimulus histories, exist to supply the information needs of the user by way of responses. While this transaction analysis is being done, a verification should be made to ensure that the definition of the problem includes enough information in the transaction to cover system integrity, which includes system security, system operability, system auditability, system reliability, and system capability.[1] Again, the needs of the system are here defined; a solution is not presented.

Establishing the highest-level black box for a system is important because this level is the true "user view" of the system. This statement means the analyst will not be describing the details of stimuli but will be concerned with the highest-level demands the user will be placing on the system. The requirements point of view should not dictate a specific design direction.

The BDG view of the requirement is helpful in understanding the requirement, but it is essential that good narrative documentation is used to back up these graphics. The BDL, shown in Figure 4, is a more formal presentation medium and is helpful in adding precision, although still in need of narrative accompaniment.

Developing the state box view of a requirement

Once the black box view of the system has been developed, the next step is to develop the state box view even if the final published documentation of the system will be left in the black box form. The state box view allows the completeness of the transitions that were described to be checked and allows the analyst to discover whether or not he or she understood the stimulus history.

The state box view is depicted in Figure 5. Here the functional definition is:

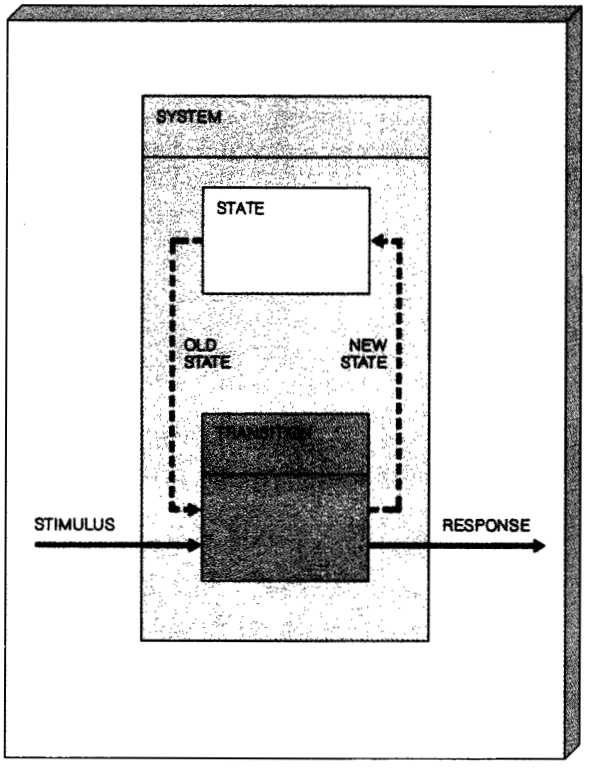
(Stimulus, Old State) → (Response, New State)

In refining a state box from a black box, the first step is to examine the stimulus history and abstract from it the state data that should be retained by the system, because the stimulus history is not part of the state box view. The key is to determine the minimal information needed to fully represent the impact of the stimulus history on the black box behavior. The purpose of the state data is to define the retained data necessary to support the system.

Figure 4 BDL description of black box

```
define BB  < BB name >
   stimulus
      < stimulus name > : < type >
   response
      < response name > : < type >
   proc
      < procedure statement >
   corp
```

Figure 5 State box view



The behavior of the black box is now restated as the transition part of the state box. The transition part

Figure 6 State box BDL

```
define SB  < SB name >
    stimulus
        < stimulus name > : < type >
    response
        < response name > : < type >
    state
        < state name > : < type >
    transition
        description
```

is a simpler view of the initial black box of the system since the total stimulus history is no longer considered.

Next, the system must have transaction closure. Satisfying this condition helps to determine whether or not the state data are sufficiently complete. Mills describes satisfying this condition as follows:

"The condition of transaction closure is satisfied if the transactions are sufficient to generate all state data, and the state data are sufficient to generate all the transactions."[1]

Another way of showing transaction closure for the state selected is to ensure that the following interactions between the transactions and the state data exist in some form:

1. Initialization. There must be some way for the state data to be created.
2. Add. There must be some way for new data to be added to the existing state data.
3. Delete. There must be some way for data to be deleted from the existing state data.
4. Update. There must be some way of updating existing state data.
5. Query. There must be some way of using the state data.

The above five transactions may be required for each piece of state data created that expands and contracts like a file. If a single entity that cannot expand or contract comprises the state data, the initialization, update, and query transactions are required. The initialization is especially important to consider since the initial state should be specified so that the reader understands what form the data take from the very beginning. The other four transactions could be combined in various ways.

If this condition is satisfied, it means that we have been complete in describing our transactions in the black box view and have selected a proper state to support these transactions. If the list of transactions is incomplete, the analyst must return to the black box view and update the list of transactions, then examine these new transactions against the stimulus history.

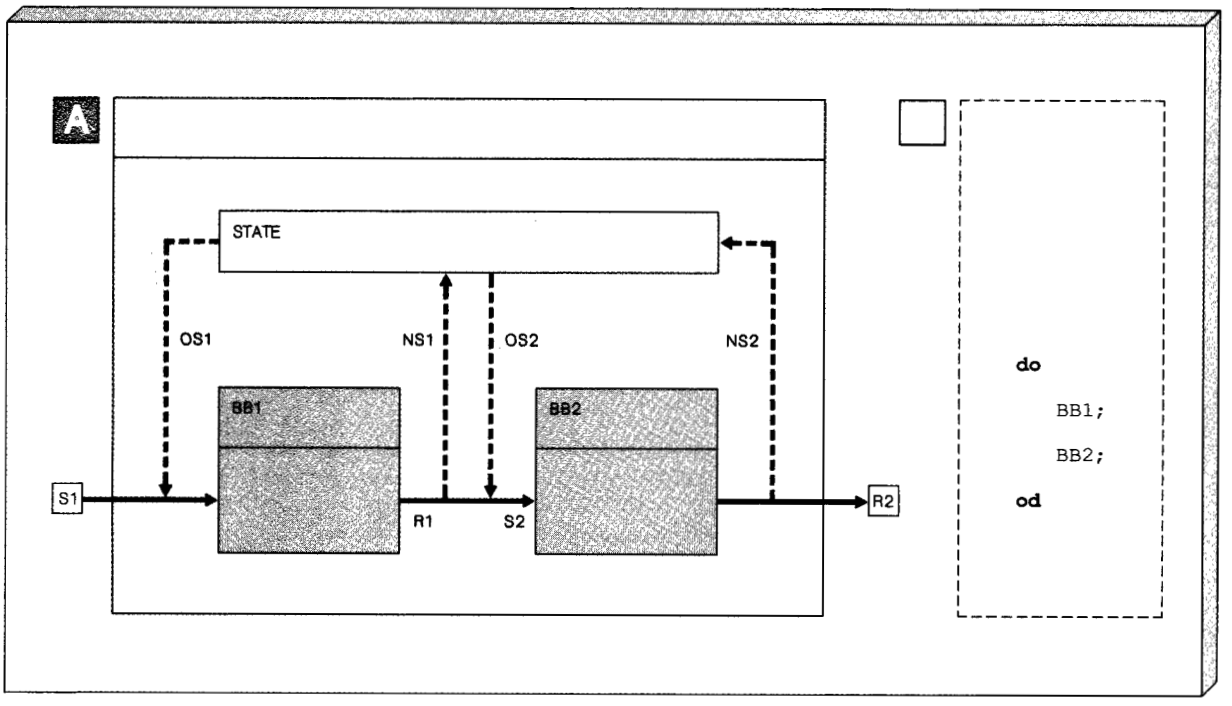
Again, these graphics should be backed up with a good narrative that describes the transactions and the state with enough detail for the reader to understand them. They can also be described in BDL, which has the format for a state box as shown in Figure 6. Note that the defined state box can be verified as a correct expansion of the black box by converting the state to stimulus history in a new black box and comparing the two black boxes for equivalence.

Developing a clear box view of a requirement

Prior to developing the clear box view (or procedural definition) of a requirement, the different transactions and retained state data as defined in the black box and state box views should be known for the system. The process of specifying a procedural definition will force the analyst to continue examining the interaction of transactions. An example of transaction interaction in a simple inventory system is the requirement that a purchase order must be prepared when an item is removed from the inventory and the inventory reaches a reorder point. The system is to contain update, manual reorder, and automatic reorder operations. In this example there is interaction between an update transaction followed by an automatic reorder transaction, where the automatic reorder transaction depends on the action of the update transaction. If in the black box and state box view this interaction was not identified, an addition to the definition of the behavior of the update transaction must be made so that it will show the appropriate response.

The process of expanding a state box into a clear box is accomplished by taking the single transition func-

Figure 7 (A) The sequence clear box; (B) the sequence clear box BDL



tion specified in the state box and describing it in terms of the interaction of multiple smaller transitions. The transition part of the state box can be expanded into a procedural structure by one of four primitive steps, resulting in a clear box with the same behavior as the transition part of the state box. The resulting clear boxes are described below.

The functional definition of a clear box is:

(Stimulus, Old State) → (Response, New State) by procedure

Part A of Figure 7 shows the sequence clear box. In this structure the box BB1 must come before the box BB2 after the system receives the stimulus S1. BB1 also receives as input the state data of the system as it receives the stimulus S1. The response of BB1, which is R1, contains a possible update to the state data NS1 and the stimulus S2 for BB2. BB2 also receives the updated state data as input. The response of BB2 contains a possible update to the state data along with the response R2, which is the external response of the system to the stimulus S1. A sequence structure contains two or more boxes.

The BDL to support the sequence portion of the clear box is shown in Part B of Figure 7. The remaining portion of the BDL is the same as the state box except with CB instead of SB.

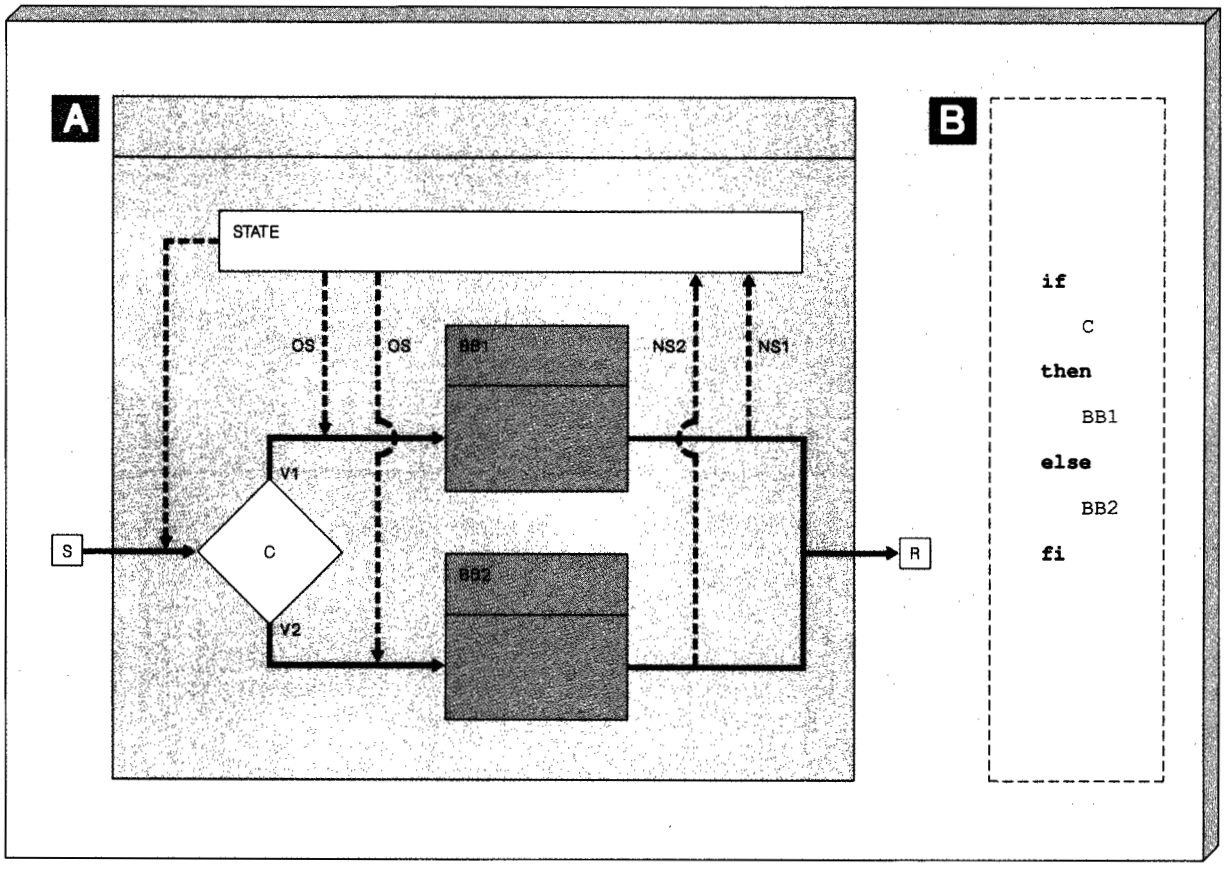
Figure 8A depicts the alternation (IF) structure. In this structure the stimulus S and the state data are used by the condition C to determine which path will be followed through the system. Whichever path is selected, the box BB1 or BB2 will receive as input the stimulus S and the state data. The output of the box will potentially update the state and provide the external response R of the system.

The BDL to support the alternation (IF) portion of the clear box is shown in Figure 8B.

Figure 9A contains the alternation (CASE) structure. This structure operates exactly the same as the IF structure except for the number of different possible paths through the structure. The BDL to support the alternation (CASE) portion of the clear box is shown in Figure 9B.

Figure 10A is the iteration structure. This structure allows looping in the system. When the system re-

Figure 8 (A) The alternation (IF) clear box; (B) the alternation (IF) clear box BDL



ceives a stimulus S, the stimulus and the state data are used by the condition C to determine whether or not the system will loop. If it does not loop, S becomes the response R of the system. If it does loop, S becomes the stimulus along with the state data for the box BB1. BB1 then provides a potential update to the state data along with a response R1. S becomes R1, and the condition C uses this new S along with the updated state to determine whether the system will continue to loop. Whenever the condition C is no longer met, the latest S and old state contribute to the external response of the system.

The BDL to support the iteration portion of the clear box is shown in Figure 10B.

Figure 11A is the concurrent clear box structure.[5] In this structure the stimulus S and the state data are received simultaneously by the transactions BB1 and BB2 (there could be more than two of these). Each transaction provides a possible update to the state data and an external response.

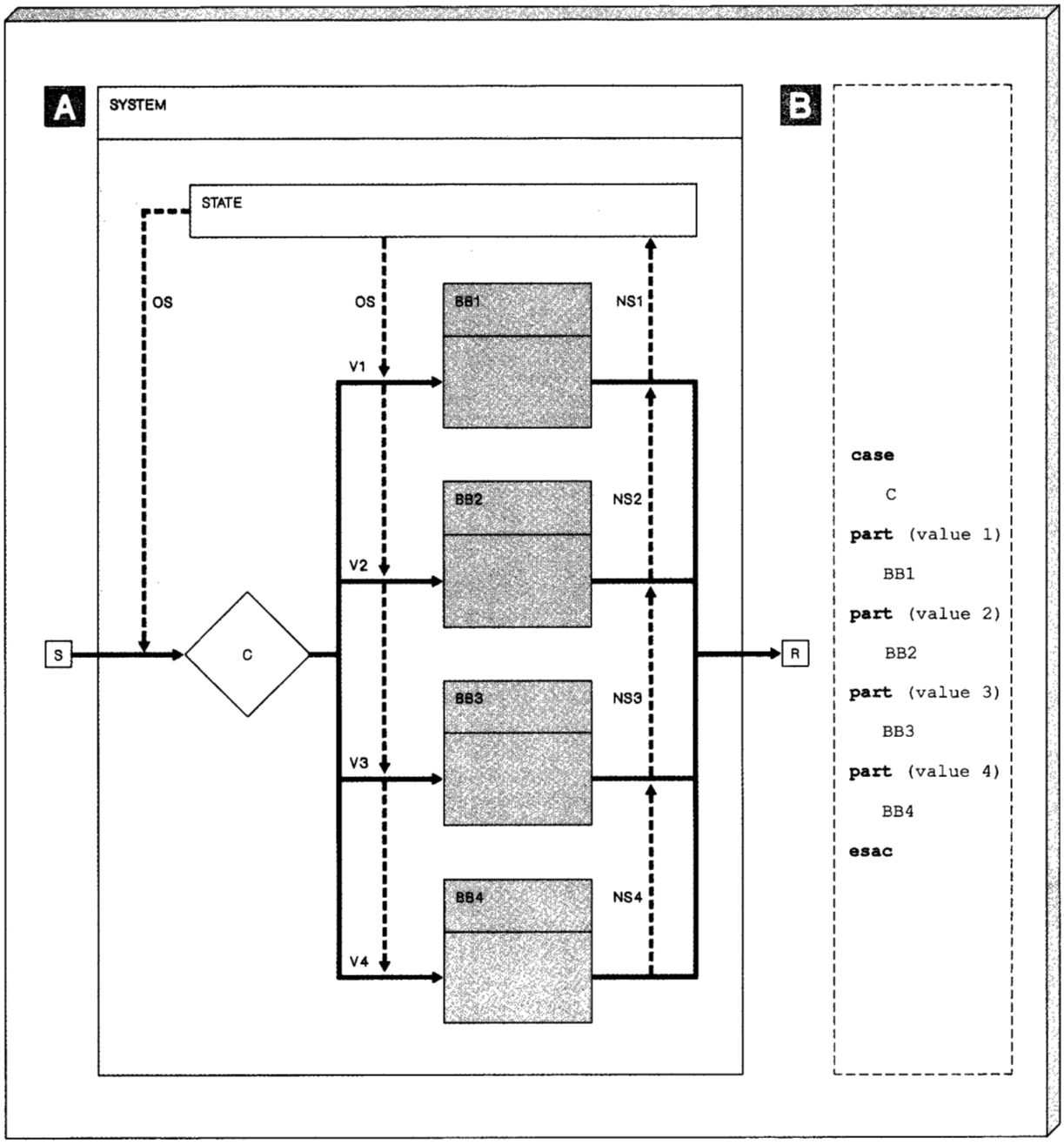
If sequencing of the updates to the state (NS1 and NS2) or the external responses (R1 and R2) is necessary, it is described in the RESOLVE function.[5]

The BDL to support the concurrent portion of the clear box is shown in Figure 11B.

Note that each clear box introduces new black boxes to continue the stepwise refinement and verification process. Clear boxes are verified to be correct expansions of their state boxes by reducing the effect of their procedure parts to a single step in a new state box and comparing the state box for equivalence.

A transition may be used in more than one place while the clear box view is being defined. A depen-

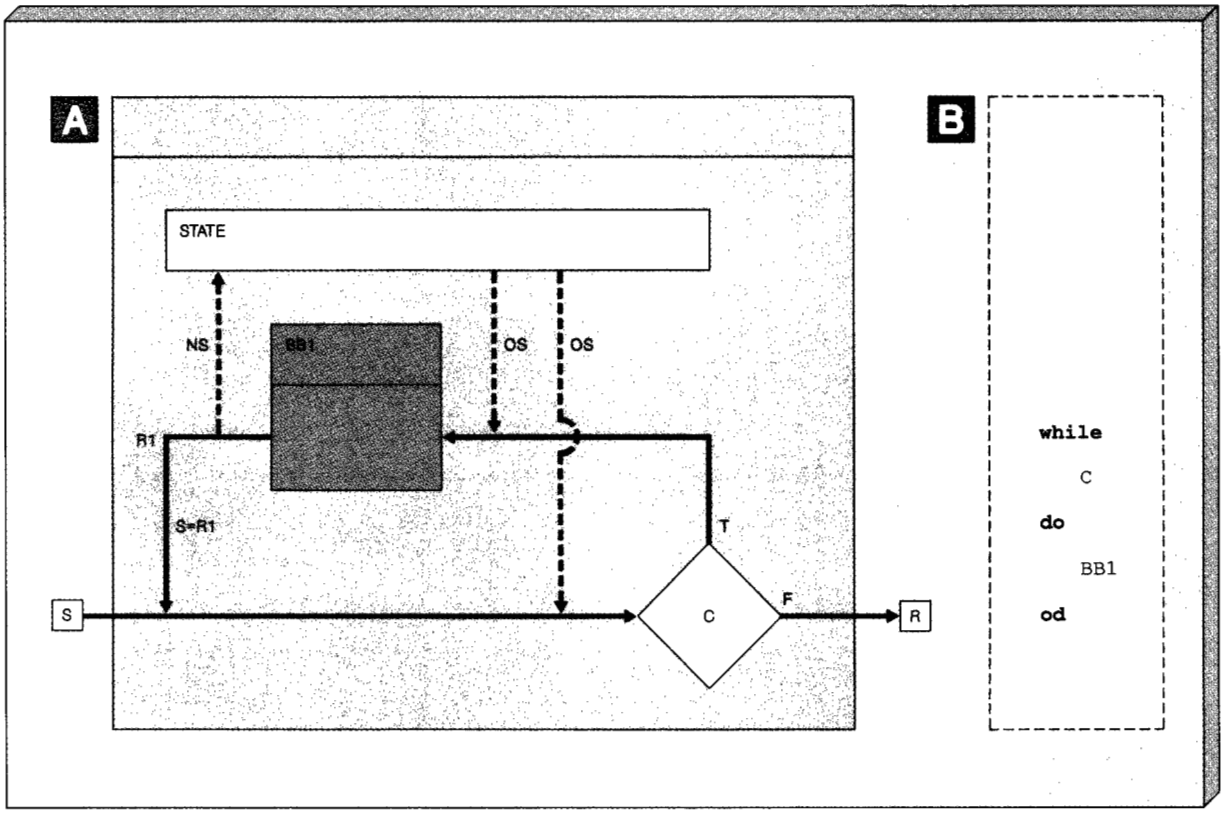
Figure 9 (A) The alternation (CASE) clear box; (B) the alternation (CASE) clear box BDL



dency tree should be developed showing the relationship between the transitions. As the definition of the system grows, these "common" use transitions will help minimize redundancy and increase the ability of readers to understand the requirement. Redundancy leads to inconsistencies, which is a major problem in communicating requirements.[4]

Keep in mind that we want to make the definition of the problem complete; we should not propose an

Figure 10 (A) The iteration clear box; (B) the iteration clear box BDL



implementation solution. The clear box structures supply the means for examining the interactions of the different parts of the problem. In this way it may be discovered that the definition of a particular part of the problem was not complete and thus save some rework later.

The last problem we face with each clear box is whether or not the individual transitions we have described as part of our procedural definition are clear enough to stand on their own with any further definition. This problem is discussed in the next section.

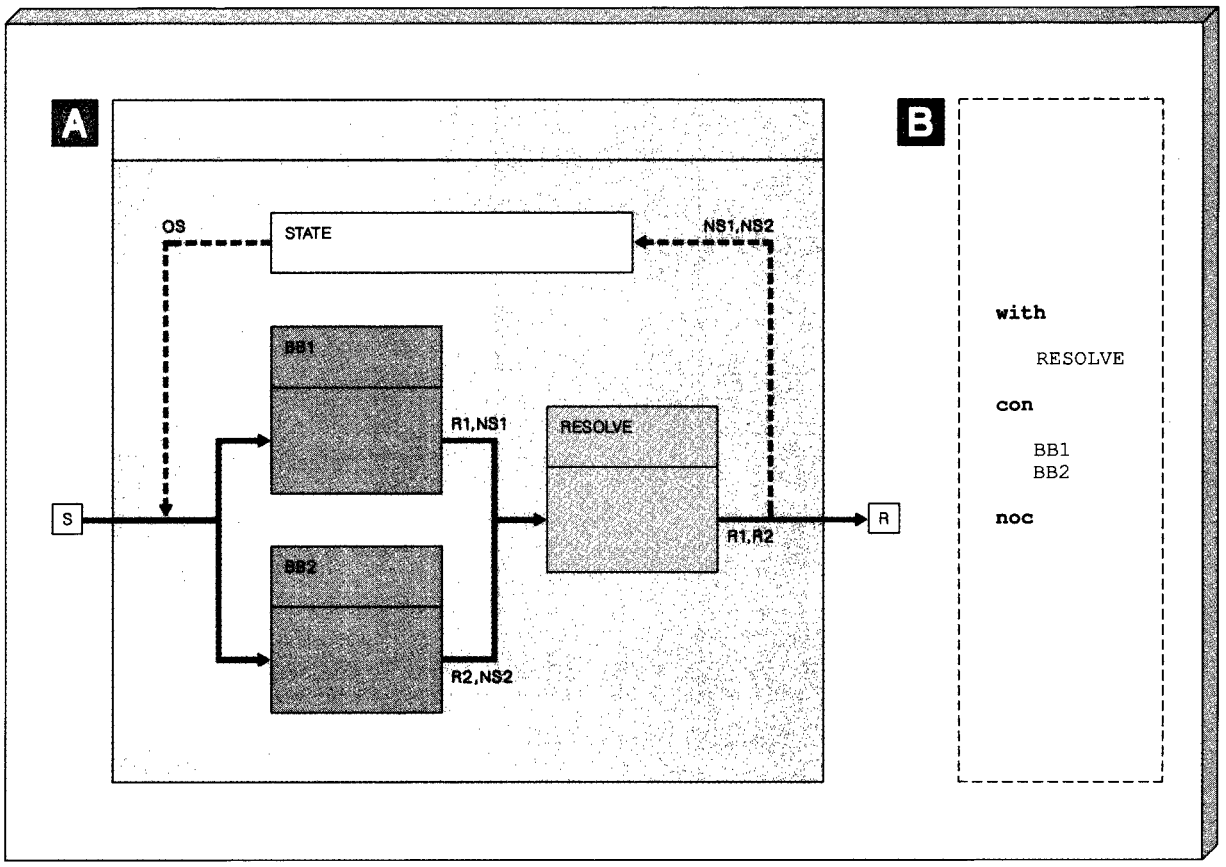
Using the structures to define the whole problem

A sample hierarchy of the definition of a large requirement is shown in Figure 12. The black box (BB) in the level designated by (1) is the system-level definition. The BB in level (1) has been refined into a state box (SB) view and a clear box (CB) view. The CB in level (1) had seven different transitions in its procedural definition which were described as black boxes. The clear box BBs become the starting point for the next refinement. Two of these BBs were refined further because of their complexity.

The major question that arises when viewing the hierarchy in Figure 12 is: How many levels must be refined before the definition of the requirement is complete? The answer to this question depends upon the complexity of the BBs of the CB in the last refinement. The following are some criteria to be considered:

1. After the first refinement, there exists a clear box that has several transitions identified (in the form of some structure). The clear box description, for completeness, will document these transitions as black boxes (stimuli, stimulus history, responses). If there are no additional transactions to be discovered or no additional state data to be identified that would require additional transactions (for transaction closure), refining is finished.

Figure 11 (A) The concurrent clear box; (B) the concurrent clear box BDL

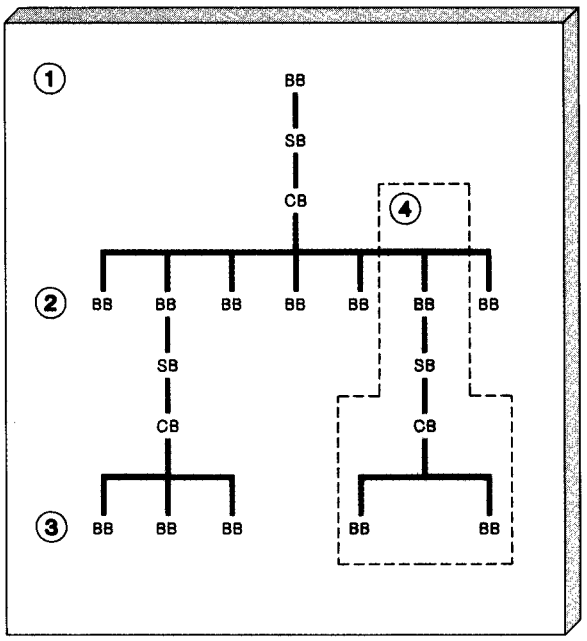


2. If there is any uncertainty about the questions raised in the clear box expansion, each of the black boxes needs to be refined from that expansion that bears on the uncertainty. That means a state box, and possibly a clear box, must be created for each transition. If the transition of the state box is clear (and simple), there is no reason to create a clear box from the state box.
3. It will be certain that an additional level of refinement is unnecessary only if the refinement process is carried to one more level than needed to establish the system requirements.
4. Complexity of the overall system is probably not a determining factor in the number of levels required. Neither is system size.

Another question might be, "Should definition of the requirement always start at the top level?" The answer to this question is "no," because it may not always be possible to start at that level. Information may come as a series of major user requirements, and there is a desire to place them all in a single system. Work can be done on each of these seemingly independent requirements, but the work must progress upward by first combining these requirements into a single CB view of the system. It is possible to move upward to the SB view by abstracting. This SB view is very valuable because it helps in looking for redundant data in the overall requirement definition. There is no unique starting place for defining the requirement for a system because it will depend upon many different attributes of the problem. Whether work starts with the top level BB and that level is iteratively updated as the levels below it are created, or the top level BB is found by abstracting upward, the level should be completed because it will be the view communicated to the user.

If the top level BB is created by abstracting upward, transaction analysis should be performed when the

Figure 12 Sample requirement hierarchy



level is defined. This will help narrow the context of where the system will reside.

As the requirement is refined into the level (2) SB views, new state data common to two or more different SBs at this level may be discovered. The decision can be made to migrate this common data up to the level above to make the data more visible to the reader and also to guard against redundancy. However, it is desirable to leave the state data at the point of origination unless a move will reduce redundancy.

A need might exist to describe part of the system that must be isolated from the rest because:

1. That part of the system will reside apart from the rest of the system;
2. That part is subject to change in a future release of the system; or
3. There will be different versions of that part for different environments.

In Figure 12 the area indicated by (4) shows where one leg of the hierarchy must be isolated for one of these reasons. By having this pointed out during requirements specification the designers will have a clear message of a major impact on their design.

Example: Workstation printer requirement

Within this section is an illustration of the use of box structures in defining (and refining) requirements for a workstation printer. This example builds from the following (incomplete) statement of capabilities:

- The printer will operate in a workstation environment where it will be connected to a workstation that is similar to a personal computer.
- The printer will have standard fonts and print control features available in the hardware that can be referenced.
- The printer will be able to print custom fonts that can be created and maintained by the user.
- The printer will have the ability to print graphics.

We will go through the black box and state box views for all of these requirements and through the clear box view that supports the custom fonts part of the requirements. Notice that some of the facts are not known at all, and rather than take a wild guess, the notation TBD (to be determined) is inserted so information can be filled in later. Also included with the example is a glossary of terms and items, which provides a place for the data items to be defined and for additional information to be added to them as the requirement is refined. The glossary will also help with the consistency issue (items being defined in multiple places under the same name or different names).

Comments about the requirement will be inserted at the point where the comment is being made and will be distinguished from the rest of the text.

Workstation printer requirement black box.

Comment—The first step is to determine the stimulus response pairs. These pairs are shown with the same numbers on each side of the black box for the system in Figure 13. The stimuli contain multiple parts and are so labeled on the diagram and detailed in the text.

1. Print Text

   Comment—The three subheadings—stimulus parameters, response, and behavior—used for each stimulus response pair, provide a place for an in-depth description.

   a. Stimulus parameters (stimulus history)

      Comment—The second step taken during the definition of the black box is to determine the stimulus

history for each of the stimulus and response pairs. The stimulus history is shown in parentheses after the name of the data item that is part of the stimulus. Further definition of the stimuli can be found in the glossary.

- Text-data (codes established in font entered through operation 3 or as part of the standard fonts)
- Font-id (identification for font previously entered into system through Create/Update or as part of the standard fonts identifications)

Comment—The following two stimuli come from outside the software system. These could be viewed as other black boxes with which this system must communicate.

- Standard fonts (available from hardware)
- Standard print controls (available from hardware)

b. Response—Printed copy or error message

c. Behavior

Comment—The third step in defining the black box is to describe the behavior that will tie each stimulus to its response. The behavior is stated in terms of the stimulus and response pair using the stimulus history. BDL is used in this description.

if no font-id is specified
then use the last font specified in the stimulus history
else
  if the font matching the font-id (either a standard font or one specified in the stimulus history through a Create/Update stimulus) is specified
  then use this font to print the text-data
  else return an error message
  fi
fi

2. Print Graphics

a. Stimulus parameters (stimulus history)

Comment—The requirement was quite vague about what printing graphic files meant. It means additional information must be found to support this area.

- Graphic-data (TBD)
- Graphics mode (TBD)

b. Response—Printed copy

c. Behavior

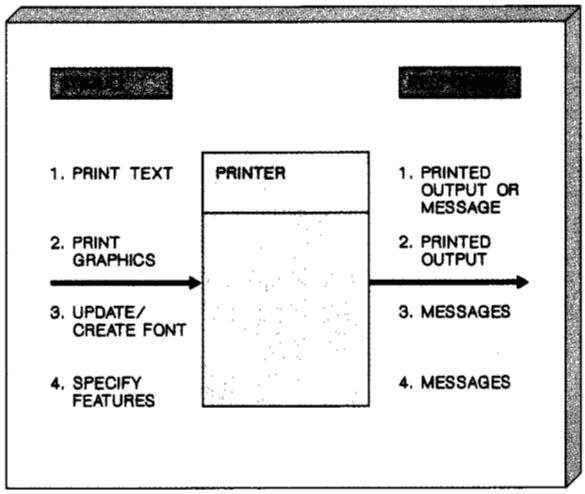
- Print graphic-data in the graphics mode. TBD

3. Update/Create Font

a. Stimulus (stimulus history)

- Request
  - Create (if the font identified by font-id has not been specified in the create stimulus history or is not a standard font)
  - Update (if the font identified by font-id has been specified in the create stimulus history and an update of the font is desired)
  - Delete (if the font identified by font-id has been specified in the create stimulus history and removal of the font is desired)
- Images
  - New-images (no history)
  - Replace-images (if the font identified by font-id was specified in the create stimulus history)
- Font-id (see request and images above)

Figure 13 Printer black box



- Standard fonts (available from the hardware)

b. Response—Message indicating success or failure

c. Behavior

- Create request

if the create request was specified, and font-id is not one of the standard fonts and is not one that has been specified in the create stimulus history

then the new font-id, along with its characters and their mappings, will be entered into the system, and a message indicating that a new font has been entered will be returned to the user

else an error message will be returned for this create request

fi

- Update request

if the update request was specified and the font-id is the identification for a font that is in the create stimulus history

then the referenced font characters and their mappings will be entered into the system, and a message indicating that the font has been changed will be returned

else an error message will be returned for this update request

fi

- Delete request

if the delete request was specified and the font-id is the identification for a font that is in the create stimulus history

then a message indicating that the font has been deleted will be returned

else an error message will be returned for this delete request

fi

4. Specify Features

a. Stimulus (stimulus history)

- Font-id (identification of a font that is either a standard font or one that is in the stimulus history of the create request)
- Print features (available from hardware)
- Standard fonts (available from hardware)

b. Response—Message indicating success or failure

c. Behavior

if the requested features are available from the hardware and the font identified by font-id is one which is in the stimulus history of the create request or is a standard font

then a message indicating that the printer settings have been made will be returned

else an error message will be returned

fi

Comment—Once the black box has been described, ensure that the description includes all of the requirements. Systems much more complex than this will require clumping (abstracting) of the parts of the requirement until a manageable number of stimulus response pairs (seven or fewer) is left.

Workstation printer requirement state box.

1. Print Text

Comment—The first step in creating the state box view (Figure 14) of a black box is to derive the state data from the stimulus history of each stimulus. With necessary stimulus history items retained in the state, the stimulus history can be dropped from the stimulus parameters.

a. Stimulus parameters

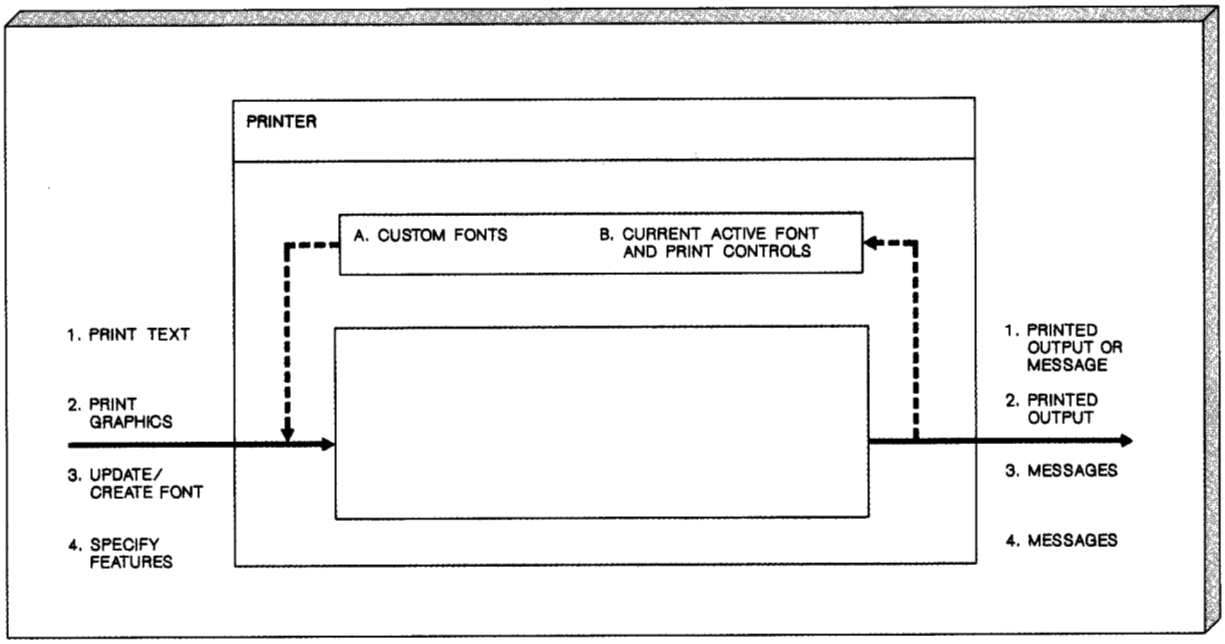
- Data
- Font-id
- Standard fonts
- Standard print controls

b. Response—Printed copy or error message

c. States

Comment—The following two pieces of state data are used to support this stimulus and response pair. The standard fonts and standard print controls are not included since they come from outside the system. Also, it is good to note the action that is taken with the state data, such as INPUT, OUTPUT, or

Figure 14 Printer state box



UPDATE. The state data will also be described in the glossary along with their initial states. Eventually we will want to make sure that there is transaction closure for every part of the state data.

- Custom fonts (INPUT ONLY)
- Current active font and print controls (INPUT, UPDATE)

d. Behavior

Comment—The second step for creating the state box is to restate each of the behaviors described in the black box view in terms of the stimulus and response pair and the state data.

if no font-id is specified
then use the current active font
else
  if the requested font exists in the standard fonts or in the custom fonts
  then take the referenced font from the standard fonts or custom fonts and use it to set the current active font and to print the data
  else return an error message
  fi
fi

2. Print Graphics
   a. Stimulus parameters
      - Graphic-data
      - Graphics mode
   b. Response—Printed copy
   c. State—TBD
   d. Behavior
      - Print the file in the graphics mode. TBD

3. Update/Create Font
   a. Stimulus parameters
      - Request
        - Create
        - Update
        - Delete
      - Images
        - New-images
        - Replace-images
      - Font-id
      - Standard fonts
   b. Response—Message indicating success or failure
   c. State
      - Custom fonts (INPUT, UPDATE)

d. Behavior

- Create request

if the create request was specified, and the font does not currently exist as a standard font

then the new font, along with its characters and their mappings, will be entered into the system, and a message indicating that a new font has been entered will be returned to the user

else an error message will be returned to the user

fi

- Update request

if the update request was specified, and the referenced font exists as one of the custom fonts

then the referenced font characters and their mappings will be entered into the system, and a message indicating that the font has been changed will be returned

else an error message will be returned to the user

fi

- Delete request

if the delete request was specified, and the font-id specified exists as one of the custom fonts

then the referenced font will be removed from the custom fonts, and a message indicating the font has been deleted will be returned

else an error message will be returned for this delete request

fi

4. Specify Features
   a. Stimulus parameters
      - Font-id
      - Print features
      - Standard fonts
      - Standard print controls

   b. Response—Message indicating success or failure
   c. State
      - Custom fonts (INPUT ONLY)
      - Current active font and print controls (UPDATE)

   d. Behavior

if the requested font exists in either the standard fonts or custom fonts

then the requested features including the font will be used to set the current active font and print controls, and a message indicating the new settings will be returned

else an error message will be returned

fi

Comment—The third step is to check for transaction closure for each piece of state data. Below are the five criteria suggested earlier for transaction closure:

1. Initialization—both described in the glossary
2. Add
   - Custom fonts—Create request
   - Current active font—Only one can be active at a time. This can be set through the print text request or through the specify feature request.
3. Delete
   - Custom fonts—Delete request
   - Current active font—It is replaced by the action of the print text request or the specify feature request.
4. Update
   - Custom fonts—Update request
   - Current active font—It can be changed by replacing it.
5. Query—Both can be used by the print text and specify features requests.

Workstation printer requirement clear box.

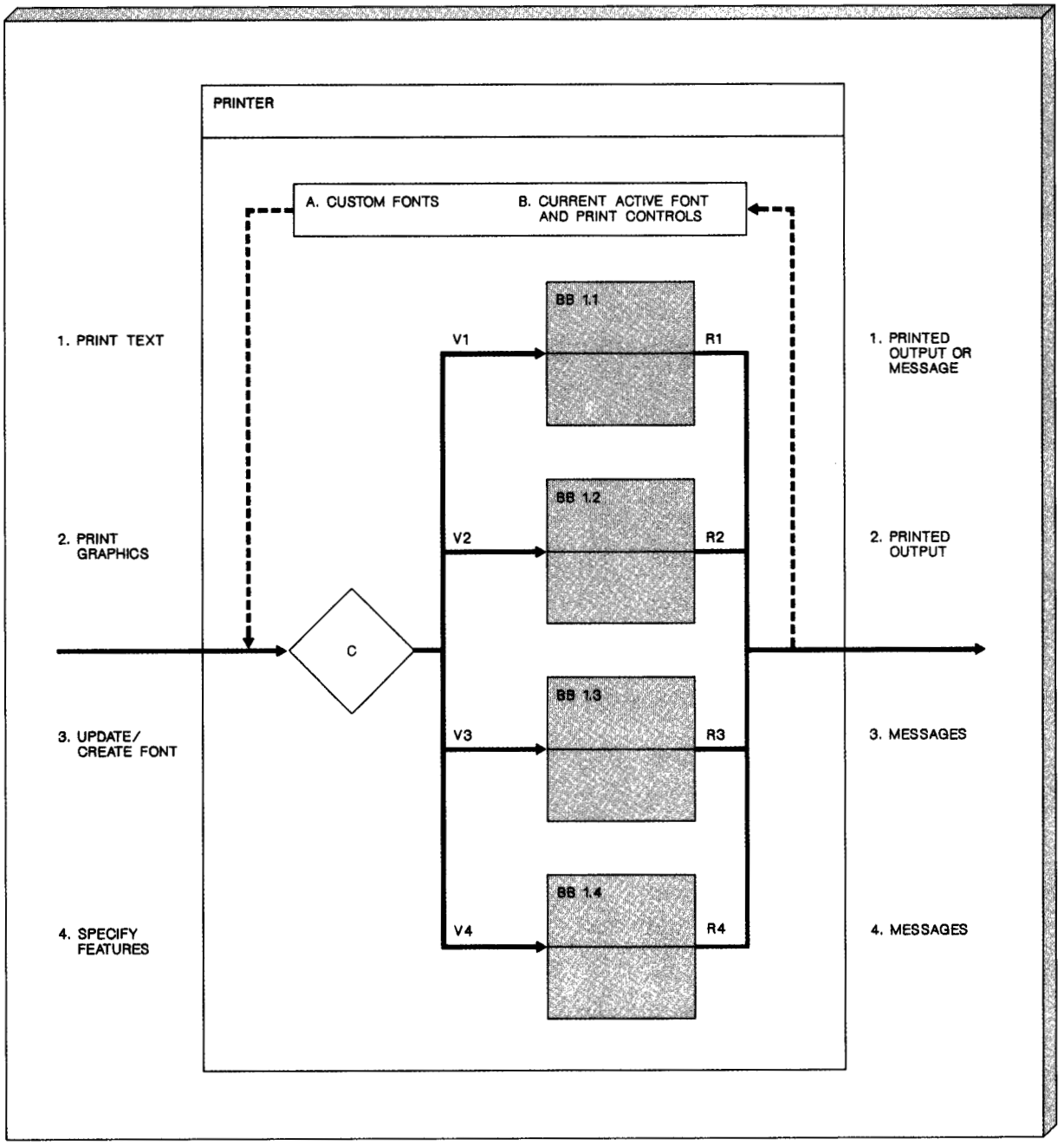
Comment—The clear box will show the procedural representation of the state box. The most typical representations at this level are alternation, concurrent, or sequence. If a combination of these is used, keep the reader in mind and keep the structure simple.

Only the Update/Create Font will be documented in this example.

Comment—The first step in defining the clear box for a state box is to decompose the transitions from stimulus to response into procedural steps. In this case the procedural steps for the stimulus response pairs are independent of one another.

The second step is to use one of the structures to describe that interaction. The structure selected for the example is an alter-

Figure 15 Printer requirement clear box



nation structure since each of the paths is independent and will not be occurring concurrently.

Comment—The use of BB 1.*n* in the clear box shown in Figure 15 could be replaced with a name. Transactions within the boxes should not be described; instead the text following the diagram should be used for the description.

Comment—What is new in the description that follows is the section describing the condition "C." This directs the reader to the correct path in the alternation structure of Figure 15. Also, the BB 1.*n* tags are related to the proper behaviors.

1. Print Text

. . .

2. Print Graphics
   . . .
3. Update/Create Font
   a. Stimulus parameters
      - Request
        - Create
        - Update
        - Delete
      - Images
        - New-images
        - Replace-images
      - Font-id
      - Standard fonts
   b. Response (R3)—Message indicating success or failure
   c. State—Custom fonts (INPUT, UPDATE)
   d. Behavior (BB 1.3)
      - Create request

        if the create request was specified, and the font does not currently exist as a standard font
        then the new font, along with its characters and their mappings, will be entered into the system, and a message indicating that a new font has been entered will be returned to the user
        else an error message will be returned to the user
        fi

      - Update request

        if the update request was specified, and the referenced font exists as one of the custom fonts
        then the referenced font characters and their mappings will be entered into the system, and a message indicating that the font has been changed will be returned
        else an error message will be returned to the user
        fi

      - Delete request

        if the delete request was specified, and the font-id specified exists as one of the custom fonts
        then the referenced font will be removed from the custom fonts, and a message indicating the font has been deleted will be returned
        else an error message will be returned for this delete request
        fi

4. Specify Features
   . . .

Comment—The third step in defining the clear box is to examine each of the black boxes that have been defined within the clear box to determine if enough detail was given for the reader to understand this part of the requirement. If there is not enough information available for a particular box, that box will become the black box for refinement into a lower-level state box and clear box.

Workstation printer requirement glossary.

Comment—The glossary provides a place for the data descriptions and functional descriptions that relate to this requirement. As more information is needed about a particular piece of data, it can be added here.

The glossary contains the description and the elaboration of the data. If the description contains (global), the data type is used in more than one place and will be described under global data descriptions.

1. Classes of stimuli description
   a. Print Text
      1) Text-data—Print file created using code of font
      2) (global) font-id
      3) Standard fonts—These are fonts that are available from the hardware and can be referenced by the font-id for each particular font.
      4) Standard print controls—These are the print controls available from the hardware and can be referenced by the proper (global–standard fonts) features keyword and value for each desired print characteristic.
   b. Print Graphics
      1) Graphic-data
      2) Graphics mode
   c. Update/Create Font
      1) Request—One of the following three would be valid:

a) Update—Keyword for indicating that an update to a custom font is desired.
b) Create—Keyword for indicating that the operation for creating a new custom font is desired.
c) Delete—Keyword for indicating that the operation for deleting a custom font is desired.

2) Images
a) New-images—Entire font character graphics and ASCII mappings
b) Replace-images—Same as new-images except contains only the font character graphics and ASCII mappings for the characters that are to be changed.

3) (global) font-id
4) (global) standard fonts

d. Specify Features
1) (global) font-id
2) Print features—Printer settings such as lines per inch, characters per inch, etc., that are part of the hardware
3) (global) standard fonts
4) (global) standard print controls

2. State data descriptions
a. Custom fonts—Contains the user-entered custom fonts which are referenced by their font-id and contains the characters and their mappings (the code by which the characters will be referenced from within a text file).

Initialization: Empty.

b. Current active font and print controls—Contains the current active font and print features (print features are defined under global data descriptions).

Initialization: At IPL (initial program load) time the printer will be set up with (TBD).

3. Global data descriptions
a. Font-id—Name of font desired
b. Standard fonts—These fonts are available from the hardware and can be referenced by the font-id for each particular font.
c. Standard print controls (for print features)—These print controls are available from the hardware and can be referenced by the proper features keyword and value for each desired print characteristic.

Observations on the printer example. The example contains all three views of the same level of the requirement. There is still work to be done for this level because there are several "TBDs" to be removed. The question about what it means to print graphic text is far from understood, but we have put a place holder into the requirement specification for it and have created a model for the entire system that can be reviewed with users and designers. It may not be necessary to use all three of these views in a particular document. They are meant to show the progression of refinements, and each view has its own audience.

The glossary supplies a "where used" understanding of the data associated with the stimuli, states, and responses. The descriptions of those data are fairly primitive, but they can be enhanced as it becomes necessary. At this stage they should help support consistency by minimizing redundant descriptions and making the document more modifiable.

The next activity would be to supply information for the TBDs and then to consider the need to take each of the BB 1.*n* boxes and decompose them into the next level of BB → SB → CB diagrams.

Conclusion

Box structures allow us to document a requirement using easily understood models. Since these models give the reader a perception of where to look in the models for different pieces of information about the requirement, they help to convey what the requirement is saying to both the user and to the developer who implements it.

A requirement written in this way can be further refined by someone else before a particular implementation is started. If one person documents a requirement in this way, other people could take the requirement and continue to refine it to determine whether they understand all of its subsystems. In this way it becomes the communication vehicle between the requirements writer and the developer.

Finally, a requirement that communicates well will be used and will be maintained as changes occur. This will help to make the product that is delivered be closer to what was initially desired by the user.

Cited references

1. H. D. Mills, R. C. Linger, and A. Hevner, *Principles of Information Systems Analysis and Design*, Academic Press, Inc., New York (1986).
2. R. C. Linger, H. D. Mills, and B. I. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley Publishing Company, Inc., Reading, MA (1979).

3. R. G. Mays, L. S. Orzech, W. A. Ciafella, and R. W. Phillips, "PDM: A Requirements Methodology for Software System Enhancements," *IBM Systems Journal* **24,** No. 2, 134–149 (1985).
4. ANSI/IEEE Std. 830-1984, "An American National Standard IEEE Guide to Software Requirements Specifications," Institute of Electrical and Electronics Engineers, New York (1984).
5. H. D. Mills, "Stepwise Refinement and Verification in Box-Structured Systems," *Computer* **21,** No. 6, 23–36 (June 1988).

General reference

H. D. Mills, R. C. Linger, and A. R. Hevner, "Box Structured Information Systems," *IBM Systems Journal* **26,** No. 4, 395–413 (1987).

John E. Odom *IBM US Education, 500 Columbus Avenue, Thornwood, New York 10594.* Mr. Odom joined IBM in 1980. He is currently part of the Software Development Technology Center where he teaches and consults with IBM organizations on software requirements and design. Since joining IBM, Mr. Odom worked in systems programming and software engineering education. Prior to joining IBM he was an associate professor of computer science at Mankato State University in Minnesota where he spent ten years on the faculty. He also served for two years as the executive manager of a data processing organization in Minnesota. He received his B.S., M.A., and Educational Specialist degrees from Mankato State University in 1968, 1969, and 1974, respectively. Mr. Odom is a member of the IEEE Computer Society.

Reprint Order No. G321-5386.