ELECTRONIC **WORKSHOPS IN COMPUTING**

Series edited by Professor C.J. van Rijsbergen

D. J. Duke , University of York, UK and A.S. Evans, University of Bradford, UK (Eds)

2nd BCS-FACS Northern Formal Methods Workshop

Proceedings of the 2nd BCS-FACS Northern Formal Methods Workshop, Ilkley, 14-15 July 1997

Using Graphical Icons to Build Z Specifications

C.N. Yap and M. Holcombe

Published in collaboration with the British Computer Society

Using Graphical Icons to Build Z Specifications

Chih Nam Yap

Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK

Mike Holcombe

Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK

Abstract

The Z specification language is now widely taught in universities [1]. However, many novices find that, although they may understand specifications written by other people, they are unable to produce their own, satisfactorily. One of the reasons is the lack of suitable tools for these novices to learn Z systematically. Another is the mathematical nature of the syntax of the language. The same problems can be found when one considers the poor "take up" of formal methods in industry.

This paper describes a new tool called VisualiZer which can allow users to write Z specifications by creating and manipulating visual entities without worrying about the possibly, daunting, mathematical symbols. Furthermore, in order to help users' understanding of the concepts behind Z, the tool is designed around an environment with graphical on-line help facilities to guide users through the process of building correct and complete Z specifications in a methodical way.

1 Introduction

The Z specification language is a kind of modelling mechanism to aid the understanding of computer related systems. The application domains of the language include software products [2], operating systems [13], hardware devices such as oscilloscopes [3], or even safety-critical systems such as The Cabin Intercommunication Data System (CIDS) of the Air-bus A330/340 system [9].

The Z specification language is based on first-order predicate calculus and set theory. The language, itself, utilises quite a lot of powerful mathematical symbols, which can be used to describe a system model in a precise and succinct way. That is why a recent survey [1] has revealed that Z is rated as the most popular specification language used in industry; it is also considered by many Universities in UK as the prime specification language to be taught.

In recent years, a few experiments have been conducted in universities to find out the result of teaching formal methods, especially Z, to students. Results from Green [8] showed that students who have learned Z understand specifications from book examples, may not know how to actually start writing Z specifications themselves from a set of English requirements. Another experiment conducted in the University of Greenwich (see Finney [5, 6]) revealed that students already trained in discrete mathematics and the Z notation performed very poorly in reading and understanding existing Z specifications.

Students' poor performances are due to many reasons. Green's reason was that Z is just a language, no explicit method has been imposed on it, which makes it relatively easy to understand but very difficult to use. Whereas Finney's reason was that people find formal specifications difficult to read because of the large use of mathematical symbols, inappropriate variable names and the lack of English comments to support the understanding of specifications. These results indicate that there are two outstanding issues which should be solved in order to remedy the situation. First of all, a method to guide users to write Z specifications is needed, and secondly, the introduction of new techniques to reduce the amount of mathematical symbols used in a specification is required. Clearly, the former needs the indoctrination of correct concepts and the latter needs some kind of media such as a tool to help students to construct Z specifications without worrying about the use of daunting mathematical symbols.

At present, there are many "so-called" formal methods tools available in the market either purchasable or free of charge which users can download from the Internet. Most of these tools help users to produce pretty-print formal documents, type check and some even provide proof facilities. To manipulate these tools, a substantial knowledge of formal methods is required; in other words, the target of these tools is expert users. Clearly, people who are just beginning formal methods will have problems with these tools that need formal methods experience, what they really need is tools that can teach or help them to understand formal methods rather than help them to produce pretty printed formal documents. Unfortunately, tools so far available do not pay much attention to this issue.

The desire to build a system which can help novice users to understand the Z language, and on the other hand help expert users to produce pretty-print specifications dates back to 1993, this system is called the *Visual Z* system.

2 The Visual Z System

The Visual Z system is an integrated system. It will be made up of a set of applications which provide tool support for users wishing to develop and analyse Z specifications. These applications are independent to each other but they all access a common database. Applications like the *VisualiZer* helps users to create visual specifications; TextualiZer allows users to write conventional textual Z specifications; *Animator* is used to animate specifications; *Prover* is for specification proofing and *Converter* is to convert specifications into code.

The main objective of the Visual-Z system is to provide tools to help users, especially novices, to understand the concepts behind the Z language. This is achieved by imposing methods and visual mechanisms, such as the drag and drop of icons, to allow users to create complete Z specifications in easy and appropriate ways. For novice users, they can use the VisualiZer to create Z specifications graphically and then use other tools such as Animator to animate the specifications, or use Prover to produce some simple proofs. The finished specifications can be converted into textual form using the TextualiZer and then read by an expert user. Or alternatively, expert users produce teaching materials using the TextualiZer and store the materials in the database, novice users can then browse these materials from the database using the VisualiZer.

At present, the focus of the Visual Z system is on the VisualiZer application. The user interface design of the application has undergone many task analyses and usability tests. This paper will illustrate the features of the user interface design of VisualiZer and shows how the tool can help users to produce a simple Z specification.

3 How the VisualiZer Application Works

One of the novices' difficulties in writing Z specifications is that they usually do not know the sequence of stages involved in producing a specification. Although writing a Z specification is an iterative process, there is still a sequence of 4 stages to follow[1]. These are:

1. Specify Types.
   - Define Basic Types.
   - Define Compound Types.
   - Define Free Types.
2. Specify Data Abstraction.
   - Define Data Schemas.
     - Schema Inclusion.
     - Define Attributes (or state space).
     - Define Invariant.
3. Specify Procedural Abstraction.
   - Define Procedural Schemas.
     - Schema Inclusion.
     - Define Input/Output Attributes.
     - Define Predicates.
4. Specify Schema Calculus.

The design team of the VisualiZer application has paid a lot of attention to this problem. The solution reached is that a *Restriction Method* should be imposed in order to guide users through the sequences defined above. The restriction method is part of the user interface of the VisualiZer. The method has been integrated in the user interface in various forms and each of them will be described in subsequent sections below, when appropriate.

The VisualiZer application consists of two main parts: a system menu and an application window. The application window is sub-divided into two portions, the top portion is called the navigation menu area and the lower portion is called the workbench. Figure 1 is a schematic picture of VisualiZer.

[1] For some complex specifications, this sequence of stages may no longer be valid.

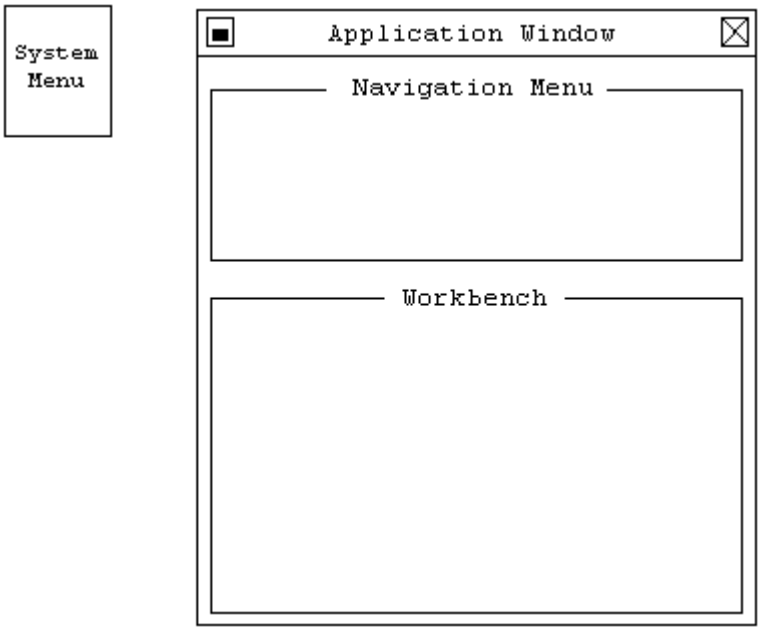

Figure 1: Schematic of VisualiZer

3.1 Menu

3.1.1 System Menu

When the VisualiZer is activated, a *System Menu* will appear at the top left corner of the monitor screen. This system menu is part of VisualiZer but it is not within the application window. The basic functions of the system menu are for users to define the name of specifications, data and procedural schemas, free types definitions and schema calculus. The menu also provides other facilities such as commands to open a specification, save a specification or quit. Figure 2 shows the system menu of VisualiZer.

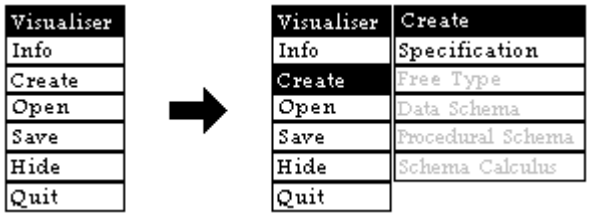

Figure 2: System Menu

In Figure 2, some of the menu items such as “Data Schema” are disabled. This is because users should create a specification before they can create the other elements in the menu. This is an example of imposing the restriction method described above.

Users can create new specifications by clicking the menu item “Create” then the menu item “Specification”. Then a pop-up dialogue box will appear in which the user can enter the specification name. Once a new name is entered, an application window will appear on the screen to act as the workspace of the new specification.

3.1.2 Navigation Menu

The navigation menu is for users to navigate within the application window. Each of the 4 stages to write Z specifications is represented as an icon within the menu and users can enter any of these stages by selecting an icon. However, since the sequence of these 4 stages is important, restriction methods also form part of the user interface of the menu. This is done by imposing restrictions on users by allowing them to select felicitous icons at different occasions. For example, at the very beginning of the constructing a specification, VisualiZer only lists three icons in the menu as shown in Figure 3.

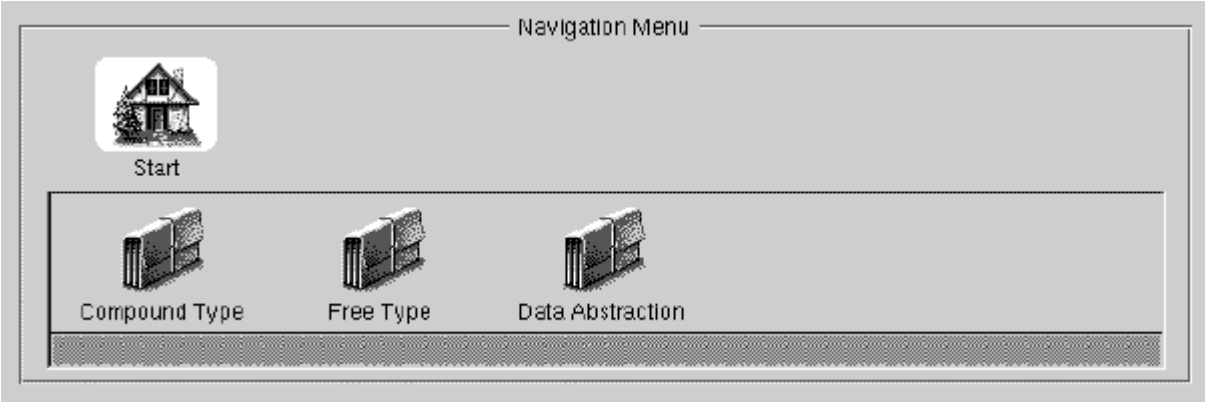

Figure 3: The Icons Representing the First Three Stages in the Sequence

New icons representing stages later in the sequence will appear only when users have performed some required tasks. For example, the icon representing the "Specify Procedural Abstraction" stage will only appear after the user has defined at least one data abstraction schema. (The creation of a data schema will be described in section 3.2)

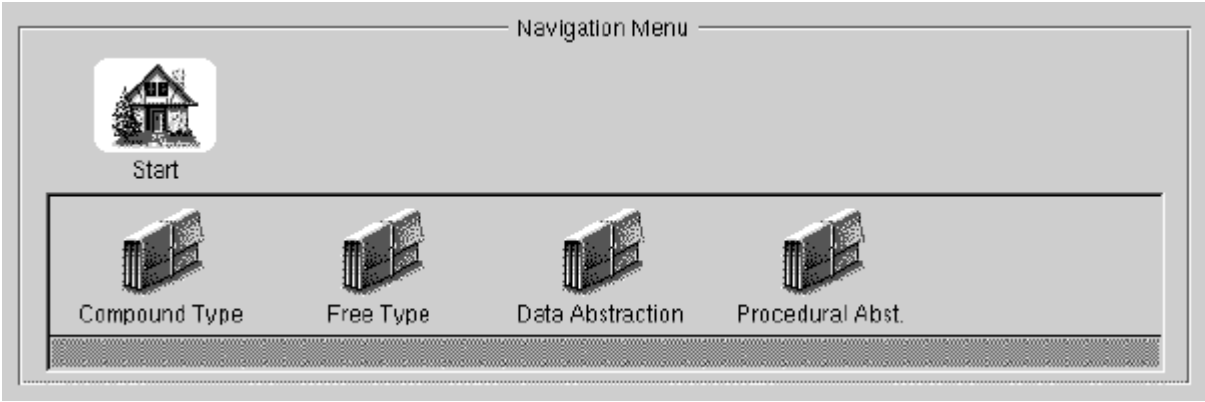

Figure 4: Extra Icon "Procedural Abst" Appears Only After Users Have Defined One Data Abstraction Schema

To switch to one of the stages such as "Compound Type", the user simply needs to double-click on the icon labelled "Compound Type" and this switches the system into the "Compound Type" mode as shown in Figure 5.

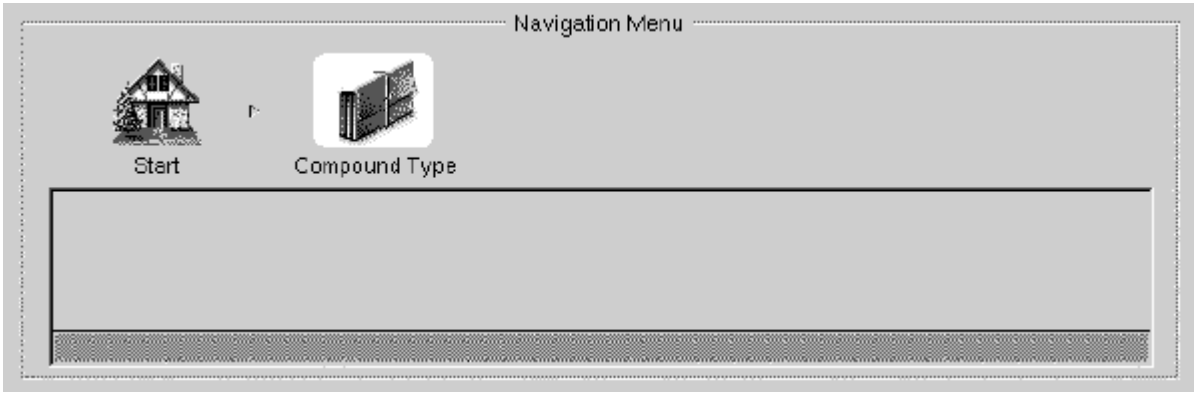

Figure 5: Compound Type Mode

3.2 The Procedure for Creating a Z Specification in the VisualiZer Application

3.2.1 First Stage – Specifying Types

Most of the system specifications are usually generated from a set of informal requirements. System designers need to understand the requirements by studying documents and communicating with customers, including potential users. Once the requirements of the system have been studied thoroughly, system designers will, more or less, start writing the specifications of the system. Of course, the chances of misunderstanding requirements still exists at this stage, but these misunderstandings might be revealed by the later stage of the system development. Writing specifications may be considered as the starting point of revealing those misunderstandings.

Basic Types

Normally, the first step in writing a new Z specification is to define a set of basic types. Defining basic types is an iterative process. Defined basic types may be changed at any time if system designers find that the basic types are not suitable or are not correctly defined. This is fine when a specification is written on paper, but it is a disaster when the specification is written using computer means, such as the VisualiZer. This situation occurs because basic types are considered to be the basic building blocks of a specification. Removing an existing basic type might make a specification no longer useful and a re-write might be necessary. Because of this, the VisualiZer provides a library of pre-defined basic-type icons for users to use. Users can rename any basic type icon by editing it on the screen but cannot remove them from the system. However, the library of icons may not be capable of representing all different kinds of basic types. Future work on the VisualiZer could be the inclusion of a facility which could allow users to drag and drop icons from other applications to the VisualiZer in order to enhance the icon library.

Basic type icons are shown to users in a scrollable area within the workbench of the application window. Figure 6 shows a portion of basic type icons in the Drag area of the application window.

Figure 6: Basic Type Icons

Compound Types

A compound type is actually a combination of basic types. From the system designer's point of view the creation of a compound type makes the specification much easier to write and from the reader of the specification's point of view it is easier to understand.

To create a compound type, the user must first enter the Compound Type mode as shown in Figure 5. Once the VisualiZer is in this mode, users can create as many compound types within the workbench as they need. The procedure for creating a compound type is that the user first clicks on one of the buttons labelled "SET", "FUN" within the workbench. The consequence of this is that an empty compound type shell will appear in the working area of workbench. Next, the user needs to drag basic type icon(s) onto that empty shell. A panel will then appear and ask for the name of the compound type. The procedure stops when a name is entered. For example, supposing one would like to create two compound types,

```
Department ==[2] P PERSON
University ==  P Department
```

Figure 7 shows what has happened when the basic type "PERSON" is dragged into the empty shell to form the compound type "Department", and Figure 8 shows the two completed compound types.

[2] In Z convention, two equal signs "==" are used for the definition of a compound type.

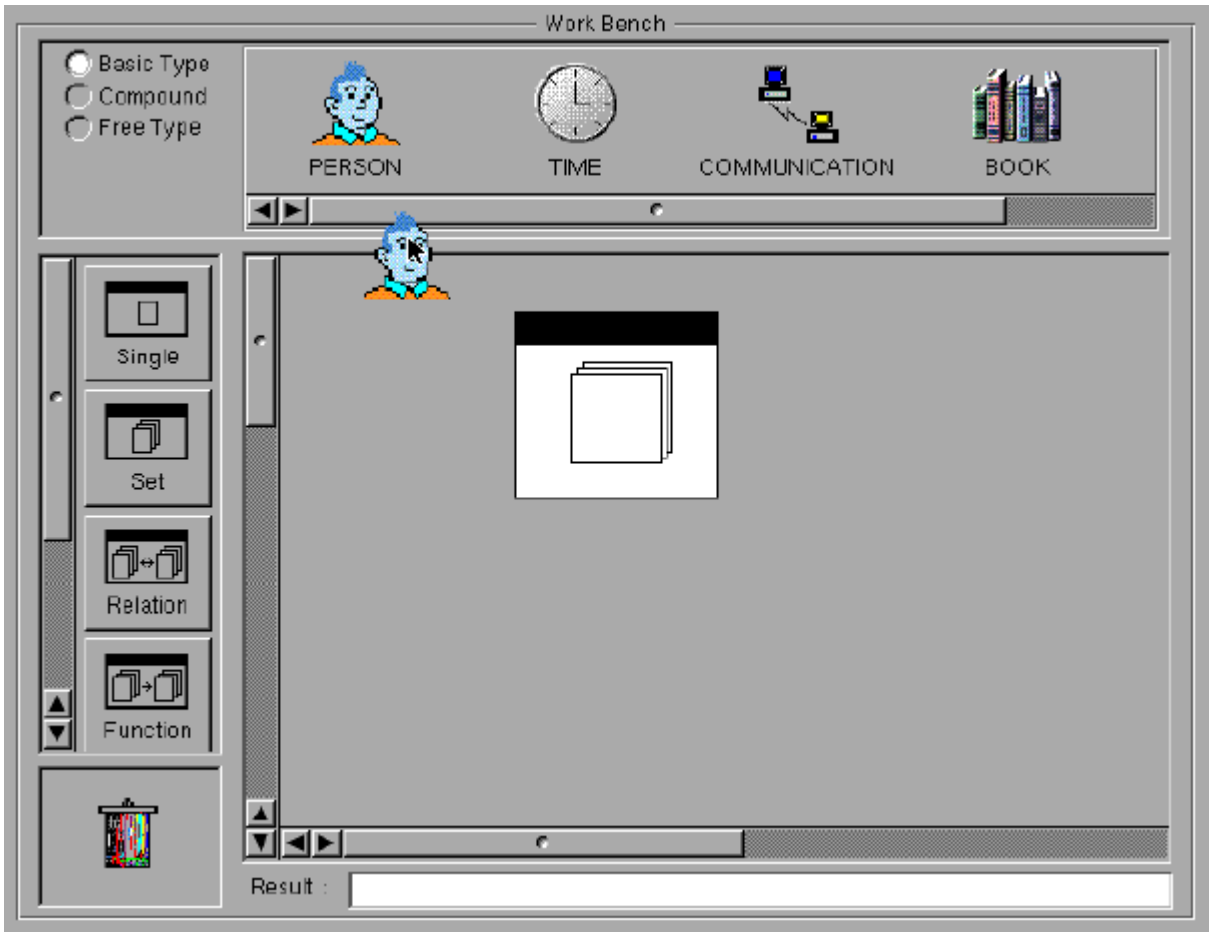

Figure 7: When a Basic Type "PERSON" is Dragged into the Empty "Power Set" Shell

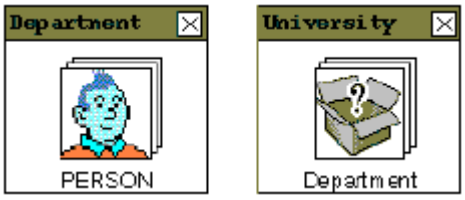

Figure 8: Two Completed Compound Types "Department" and "University"

The compound type "University" is actually formed by the compound type "Department". Because of this it has a special icon (an opened box with a "?" mark) to represent its formation type. Users can reveal what actually formed the compound type "University" by clicking on the special icon, the content of "University" will be expanded. Minimisation is also possible by clicking on the inverted triangle as shown in Figure 9.

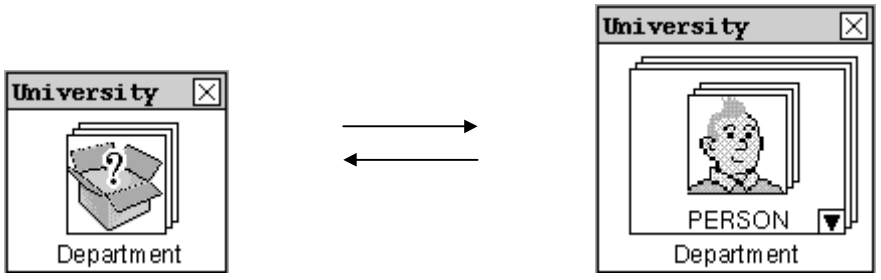

Figure 9: Revealing the Content of the Compound Type "University"

Free Types

In almost all system designs, the generation of output messages plays an important role because these messages act as a means of informing users of the system what has happened inside the system. Defining a free type in a Z specification is a succinct way of specifying system output messages. Most of the free type definitions are in enumerated form; i.e. a free type is defined by listing all its associated elements. Of course, a free type can be used for other purposes as well, such as defining a recursive structure [14]. But the current version of the VisualiZer only allows users to create enumerated free types.

To create a free type in the VisualiZer involves two parts, the first part is to define the name of the free type and the second is to define all its associated elements. Defining the name of a free type is done via the system menu. Once the name is defined, an icon representing that free type will appear in the navigation menu. The next step is to click on that icon in order to switch the VisualiZer into a mode where the user can define all the associated elements of that free type. Figure 10 shows a free type called REPORT-SYS and its elements "ok" and "unknown".

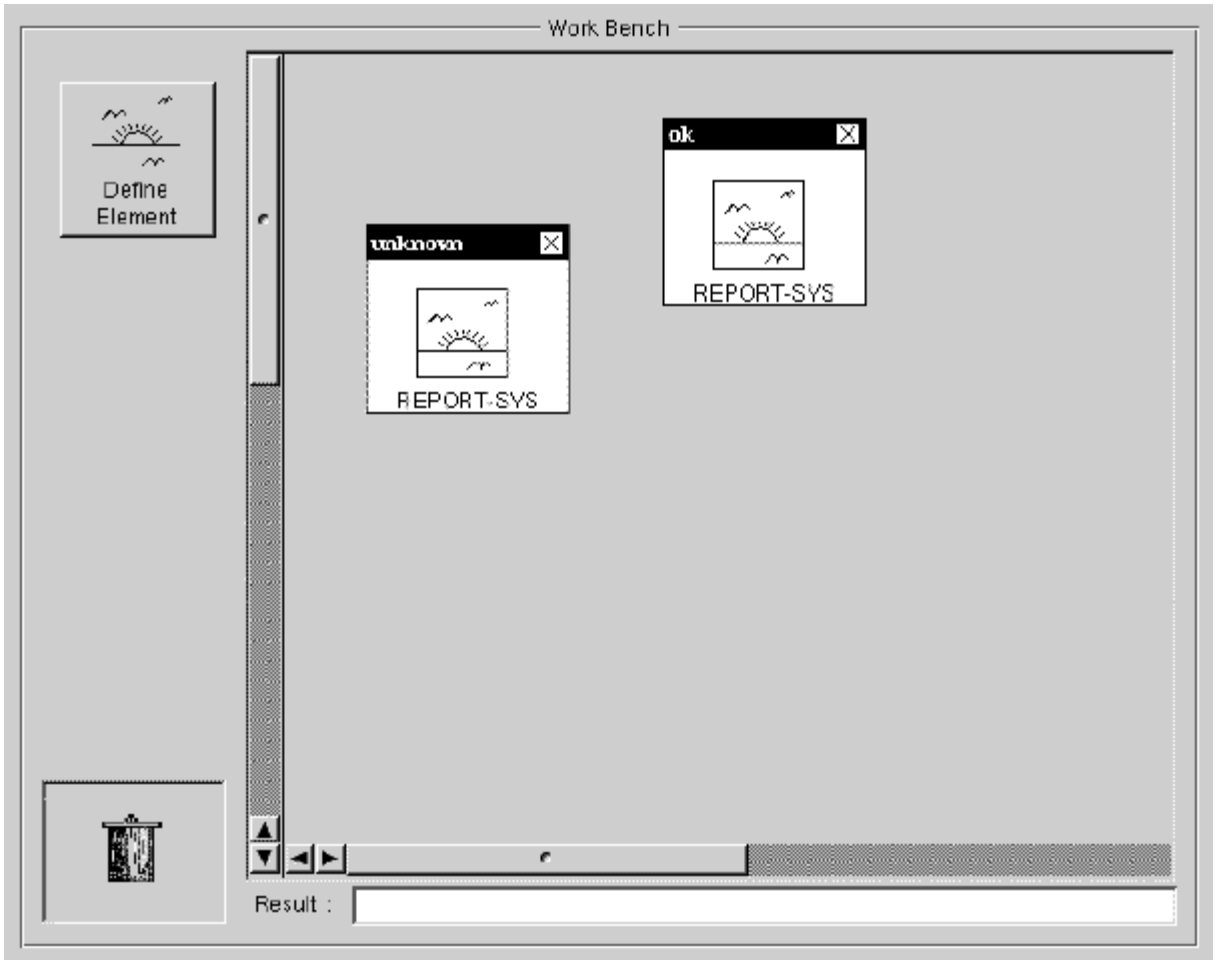

Figure 10: REPORT-SYS ::= unknown | ok

3.2.2 Second Stage – Specifying Data Abstraction

In conventional textual Z specification creation, defining the data abstractions of a system is one of the important steps. This is because data abstraction is used to encapsulate the states of the system and it also describes the relationships between these states. To define the data abstraction of a system using the VisualiZer, the user has to first create one or more data schemas. Data schemas can be created by selecting the menu items "Data Schema" from the system menu and followed by entering a name in a pop-up dialogue box.

Each defined data schema has associated sub-tasks to be performed. These sub-tasks include defining attributes (states) and invariants as shown in Figure 11.

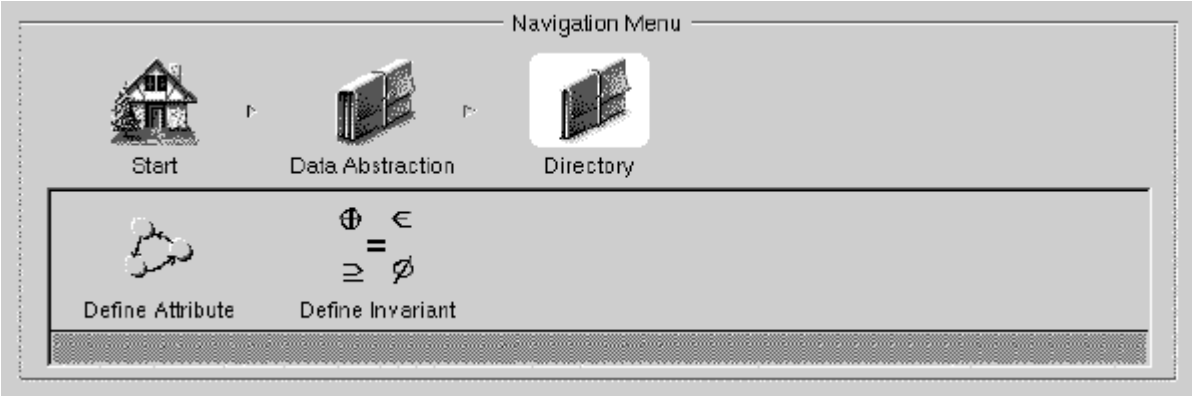

Figure 11: The Two Icons Represent the Sub-tasks of the Data Schema "Directory"

Define Attributes

The creation of an attribute is similar to the creation of compound type. First the user chooses a mathematical data structure such as a set or a function from the workbench by selecting a button in the command area and this will yield an empty attribute shell[3]. Next, the user drags basic type icon(s) or compound type icon(s) from the drag area and drops them onto the empty shell. A simple panel will prompt users for the name of the attribute. However, if the user is creating a function type attribute, a dialogue box with graphical aids will appear instead of a simple panel. This dialogue box will ask for the name and the function type of the attribute. By selecting different radio buttons, users can see different explanations of function types and from there decide what to choose. Once the name of the attribute and the function type (such as a partial surjection function) have been chosen, the completed attribute will be shown within the workspace.

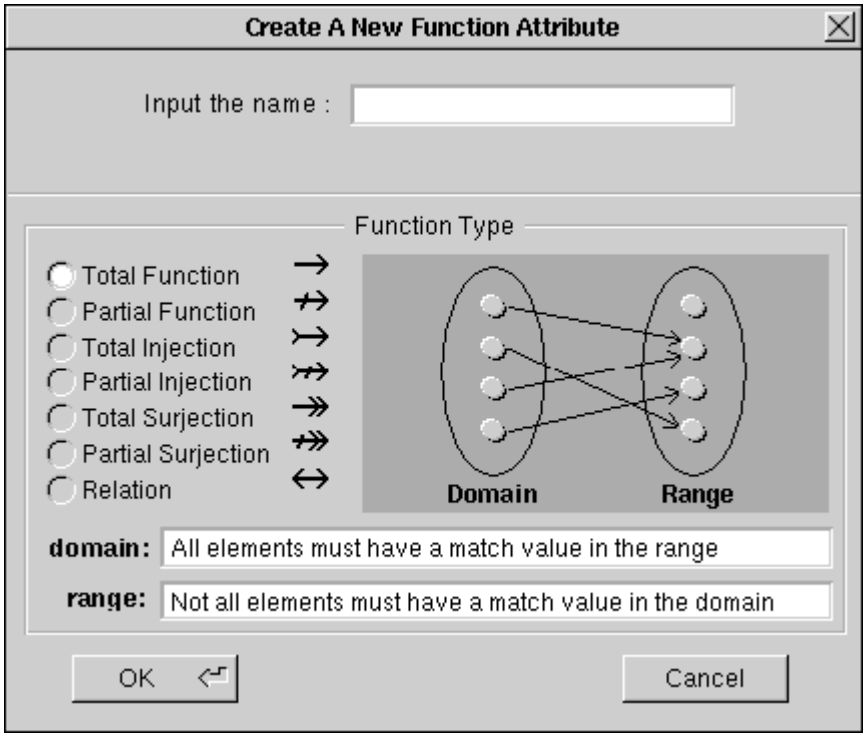

Figure 12: Dialogue Box with Graphical Aid

[3] An attribute shell is used to create an attribute, whereas a compound type shell is used to create a compound type. Both of them look very similar.

Figure 13 shows how a set attribute *AllUsers*[4] and a function attribute *Password*[5] are represented in VisualiZer. Note that the attribute "AllUsers" shown in Figure 13 looks very similar to the compound type "Department" shown in Figure 8. In the real application, they are differentiated by the colour of the title bar. A compound type has a light-grey title bar whereas an attribute has a black title bar.

Figure 13: Representation of *AllUsers* and *Password*

Define Invariant

In conventional Z specifications, there are usually some attached invariants which globally define axioms "that are true for all the procedural abstractions." Invariants can only be created after the user has defined the attributes of the system in the specification. The way an invariant is created in the VisualiZer is easy and intuitive. Furthermore, there are well explained graphical aids to help users to understand the operators they have chosen to link two or more attributes in order to form mathematical expressions.

There are basically two kinds of operators that can be found in most Z specifications, unary or binary operators. Unary operators require only a single operand and examples of unary operators are set cardinality (#) and function domain (dom). Most of the set operators such as union ($\cup$) and subset ($\subseteq$) are binary operators, these operators need two operands. The application of unary and binary operators in VisualiZer "is different. VisualiZer provides on-the-spot buttons for users to "apply an unary operator to an attribute and a merge-and-choose mechanism for applying a binary operator to two attributes.

When an attribute is used "for defining an "invariant, all the unary operators which are applicable to that attribute will be automatically listed inside the attribute. For example, in Figure 14, "one could see that there are four unary operators namely: state after, count, dom and ran waiting for the user to select. Figure 14 also shows what has happened to the *Password* attribute after the domain operator has been selected. Note the change that has occurred at the title bar.

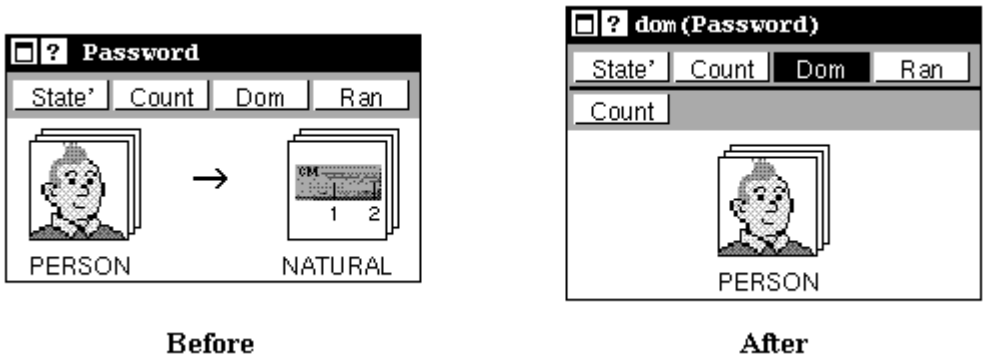

Figure 14: Apply the domain operation to the attribute *Password*

In Figure 14, the result of applying the domain operation "to the attribute *Password* will end "up in a set of PERSON. Because this result is a set, it is possible to "apply another layer of unary operators to this set. For example, one could select "he second layer's "Count" operator to achieve the operation *# (dom (Password))*.

Sometimes users may need to know the meaning of each unary operator inside the attribute, "a panel with a graphical aid is available if the "?" icon at "he top left corner of an attribute is selected.

[4] AllUsers : P PERSON

[5] Password : PERSON $\rightarrow$ NATURAL

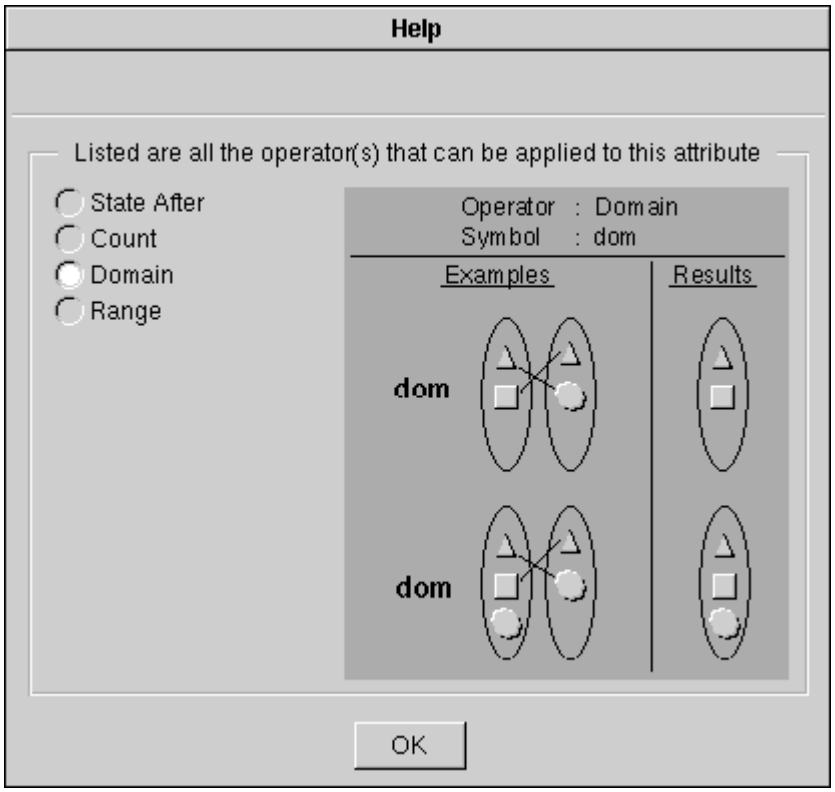

Figure 15: Help Panel for Unary Operator

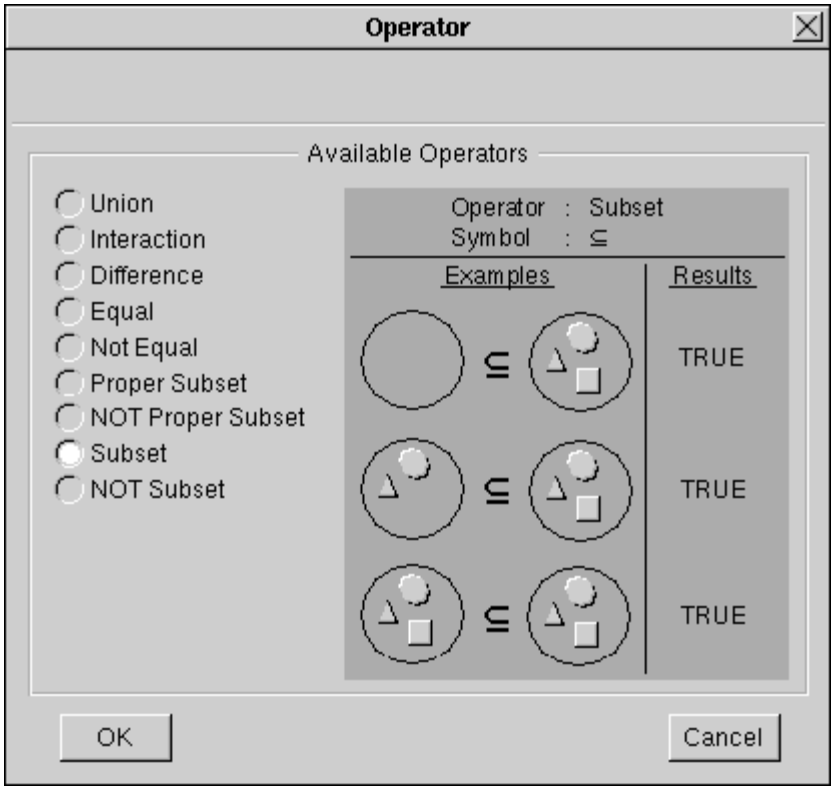

Figure 16: Panel for Users to Choose Set Operator

To form an expression such as *dom(Password)* $\subseteq$ *AllUsers* in VisualiZer, one needs to move the Password attributes on top of the attribute *AllUsers* and then release the mouse button. A panel with a list of possible binary operators in which these two attributes can be applied will be enlisted together with graphical explanations. See Figure 16.

Once an operator is chosen, a larger window-like icon will encapsulate both the *Password* and *AllUsers* attributes together with the word "Subset" in between them. See Figure 17.

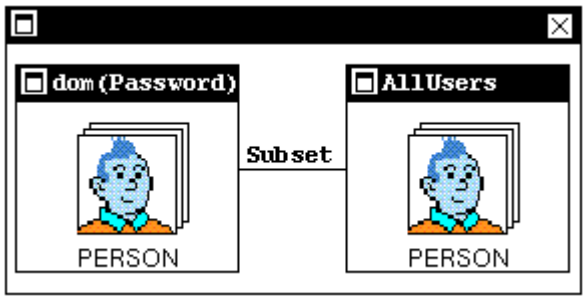

Figure 17: Complete Representation for *dom(Password)* $\subseteq$ *AllUsers*

One important thing to note is that the result of moving *Password* on top of *AllUsers* is different from moving *AllUsers* on top of *Password*. For example, if the icon *AllUsers* is on top of *Password* and the subset operator is chosen, the system will treat it as *AllUsers* $\subseteq$ *dom(Password)*, which is totally different from *dom(Password)* $\subseteq$ *AllUsers*.

A fairy simple type checking algorithm is also built within VisualiZer. The system basically checks for the basic type embedded within each attribute and decides whether both attributes can be merged together to form an expression. For example, if the attribute *Password* is defined as *MAN* $\rightarrow$ *NATURAL*, then moving *dom(Password)* towards *AllUsers* will be rejected by the system because the system treats *MAN* and *PERSON* as different basic type. Explanations on why the operation is rejected will be given to users via a pop-up panel.

To solve screen display problems, users can minimise both *AllUsers* and *Password* by clicking the minimise icon at the top left corner and the result is as shown in Figure 18.

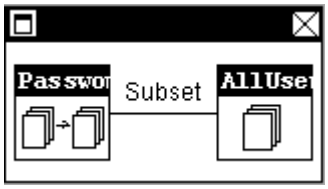

Figure 18: Minimise both AllUsers and Password Attributes

Quantifier

Mathematical expressions which involve quantifiers can also be specified in the VisualiZer. Quantifier expressions are done within a quantifier panel. Each quantifier panel is divided into two portions. The top portion of the quantifier panel consists of two rectangular variable boxes; each of these boxes is for users to specify the type of a variable. Beside each variable box there are three different kinds of quantifiers namely $\forall$, $\exists$ and $\exists_1$ for users to choose. As an example, in Figure 19, the top portion of the quantifier panel represents the expression $\forall x : PERSON$.

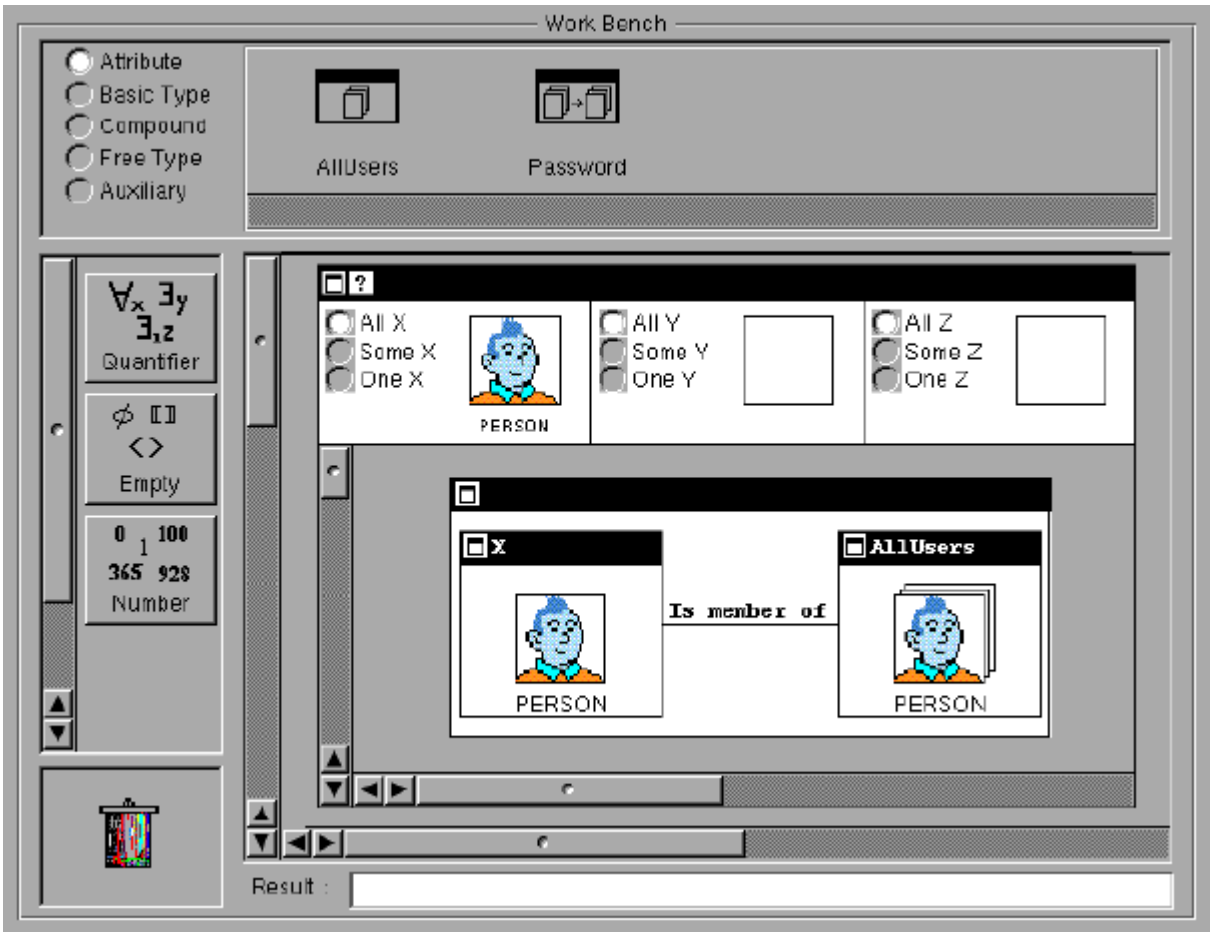

Figure 19: Quantifier Representation

The lower portion of the quantifier panel is a scrollable area for users to create predicates. To use a variable, say X, the user needs to mouse-click on the icon inside the variable box and then drop the icon inside the scrollable area. Attributes defined previously can also be dragged into this area. The procedures for creating new predicates within the scrollable area are exactly the same as for invariant's, but the scope of the predicates is only within the quantifier panel. The predicate expression in Figure 19 is $X \in AllUsers$.

3.2.3 Third Stage – Specifying Procedural Abstraction

The creation of a procedural abstraction is very similar to the creation of a data abstraction. Two kinds of sub-tasks *Define I/O* and *Predicate* can be constructed from the Procedural Schema mode. The procedure for defining I/O (input/output) in VisualiZer is almost the same as defining an attribute. The only difference is users have to explicitly tell the system whether the attribute is input or an output. This is done when the users are asked for the name of the attribute. Figure 20 shown the panel for defining the name of an I/O attribute.

In conventional Z specifications, the Ξ and Δ symbols are usually attached to procedural schemas to indicate whether any attributes have undergone any changes. In the VisualiZer, these symbols are omitted. The way VisualiZer decides whether an attribute has undergone any state changes is via checking whether users have selected the unary operator *state'* for that particular attribute and from there tracing back to the data schema in which this attribute is associated. The system will automatically generate Ξ and Δ symbols and store them inside the database together with that attribute ready for future use.

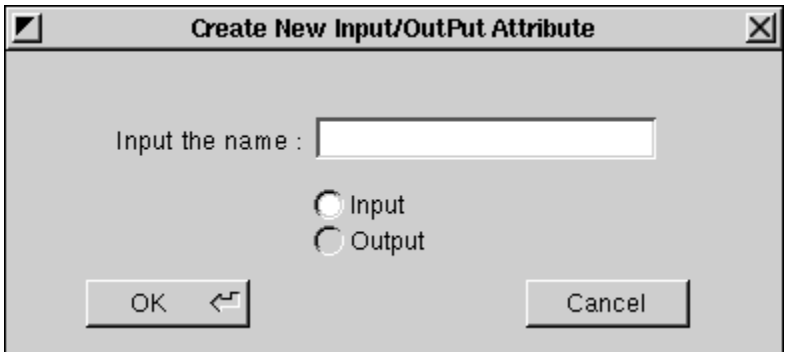

Figure 20: The Panel for Users to Enter Input or Output Attributes

3.2.4 Fourth Stage – Specifying Schema Calculus

The VisualiZer not only allows users to create data abstractions and procedural abstractions, it also allows users to do schema calculus on those abstractions in order to create new abstractions. Schema calculus is a language which can combine abstractions using *AND* or *OR* operators. This section looks at how to define the *StrongFindUser* procedural schema in the application window of VisualiZer. *StrongFindUser* is defined as:

$$\texttt{StrongFindUser} \triangleq (\texttt{FindUser} \land \texttt{Success}) \lor \texttt{NotFound}$$

The VisualiZer has two kinds of table namely, "Conjunction" and "Disjunction" table for users to create abstractions from the existing ones. These tables were originally devised by Leveson [12] for helping engineers to visualise logic statements. As an example, in the Disjunction table, when two or more procedural schema names are selected in the same column then they are considered 'And-ed' together. However, if one set of names in one column is different from the other abstraction(s), then they are considered 'Or-ed' together. Conjunction table operates the other way round.

The procedural schema *StrongFindUser* is expressed in disjunction form, therefore, users should use the Disjunction table to form the schema representation. Listed in the Disjunction table is a list of procedural schemas users have defined in stage 3 i.e., the "Specifying Procedural Abstraction". Figure 21shows how *StrongFindUser* is defined in VisualiZer.

4 Evaluation and Conclusion

Like other tools, VisualiZer does have some limitations. For example, the current version's free type definition does not support "non-enumerated" types such as recursive definitions. Also, complicated operators such as piping and renaming are not supported. We are currently working out how the quantifier panel can be made more flexible so that more than three variables can be defined.

Initially, the user interface designs of the VisualiZer were quite different from those described in this paper. The user interface design has gone through a lot of refinements and corrections since it was first devised. The very first version of user interface design can be dated back to 1993 [15] and many different designs [16] have subsequently been tried out.

A task analysis on the user interface design is on its way to completion [17]. This task analysis is based on the theory of X machines [10, 11] and the Task Action Grammar (TAG) [4, 7]. The idea is to integrate the BNF based language provided by TAG into the X-Machines language in order to model both the system and user model of VisualiZer in a coherent way.

Usability experiments are currently under way on some of the user interface designs. Prominent comments, users' difficulties and errors made have been taken into account and used to help us to refine the current design. The results of the experiments are encouraging with many subjects commenting that the design is easy to use. After performing the experiments, some participants have commented that they gained a clearer idea on how to construct Z specifications. This might suggest that VisualiZer could be used as a teaching tool for the initial learning of Z.

So far, the functionalities given by the VisualiZer should be sufficient for novices to construct simple and complete Z specifications. Although the VisualiZer is still not widely used, it is foreseen that the system is feasible and can provide practical solutions to solve the problem of writing Z specifications.

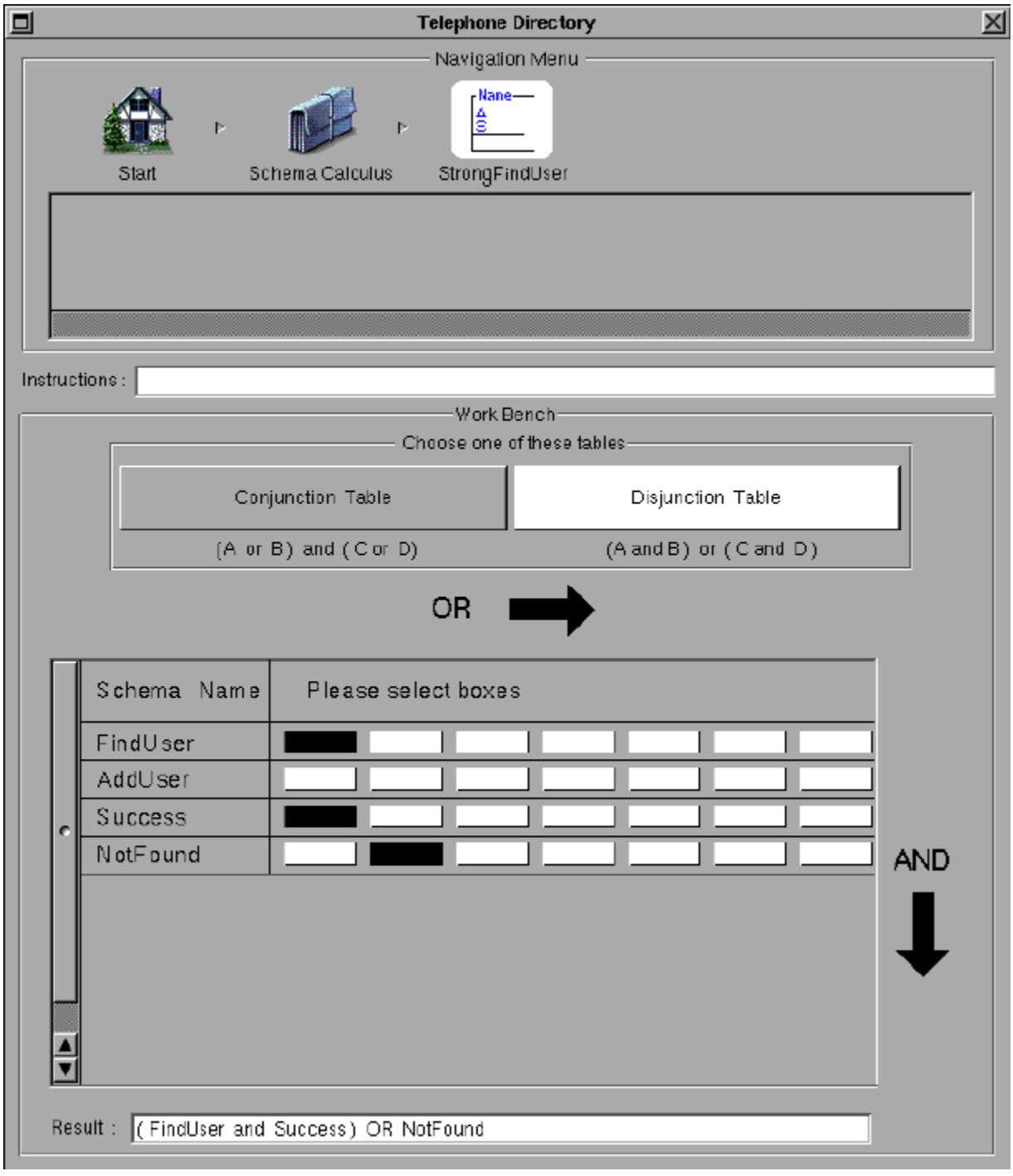

Figure 21: Disjunction Table

5 References

1. Austin S, Parkin G. Formal Methods: A Survey. National Physical Laboratory, 1993

2. Brownbridge D. Using Z to Develop a CASE Toolset. In: Nicholls J (ed) Proceedings of the Fourth Annual Z User Meeting, Oxford 1989. Springer-Verlag, 1990, pp 142-149

3. Delisle N, Garlan D. A Formal Specification of an Oscilloscope. IEEE Software 1990; 7(5):29-36

4. Eberts R. User Interface Design. Prentice-Hall, 1994

5. Finney K. Mathematical Notation in Formal Specification: Too Difficult for the Masses?. IEEE Transaction on Software Engineering 1996; 22(2):158-159

6. Finney K, Fedorec A. An Empirical Study of Specification Readability. In: Dean N, Hinchey M (ed) Teaching And Learning Formal Methods. Academic Press, 1996, pp 117-129

7. Green T, Schiele F, Payne S. Formalisable Models of User Knowledge in Human-Computer Interaction. In: van der Veer, Green T (et al) Working with Computers: Theory Versus Outcome. Academic Press, London, 1988, pp 3-46

8. Green S, Webb J. Specification: Much More Than Learning a Language. In: First International Conference on Software Engineering in Higher Education, Southampton, 1994

9. Hamer U, Peleska J. Z Applied to the A330/340 CIDS Cabin Communication System. In: Hinchey M, Bowen J (ed) Applications of Formal Methods. Prentice-Hall, 1995, pp 253-284

10. Holcombe M. X-Machines as a Basis for Dynamic System Specification. Software Engineering Journal 1988; 3(2):69-76

11. Holcombe M. An Integrated Methodology for the Specification, Verification and Testing of Systems. Software Testing, Verification and reliability 1993; 3:149-163

12. Leveson N, Heimdahl M et al. Requirement Specification for Process-Control Systems. IEEE Transaction on Software Engineering 1994; 20(9):684-707

13. Spivey M. Specifying a Real-time Kernel. IEEE Software 1990; 7(5):21-28

14. Spivey M. The Z Notation: A Reference Manual, 2nd Edition. Prentice-Hall, 1992

15. Yap C N. Implementing Visual Z. Internal Report, University of Sheffield, UK, 1993

16. Yap C N. The Design of the Visual Z System. Internal Report, University of Sheffield, UK, 1996

17. Yap C N. X-Machines Modelling Technique for HCI. Internal Report, University of Sheffield, UK, 1996