

Using Software Reuse to Develop a Z Specification Tool

A Thesis
in TCC 402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science Computer Science

by

Roy J. Bodayla

March 24, 1997

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in Humanities Courses.

Approved _____ (Technical Advisor)
John Knight

Approved _____ (TCC Advisor)
Melvin Chernov

1. ABSTRACT.....	3
2. INTRODUCTION.....	4
2.1 SUMMARY OF WHAT, HOW, AND WHY	4
2.2 PROBLEM DEFINITION.....	4
2.2.1 <i>Context</i>	4
2.2.2 <i>Concepts</i>	5
2.3 LITERATURE REVIEW	6
2.4 RATIONALE FOR AND SCOPE OF THE PROJECT.....	6
2.5 OVERVIEW OF THE CONTENTS OF THE REST OF THE REPORT	7
3. PROJECT ORGANIZATION.....	8
3.1 INTRODUCTION.....	8
3.2 PRELIMINARY RESEARCH.....	8
3.3 CHOICE OF EDITOR.....	9
3.4 DESIGN APPROACH.....	9
3.5 CONSULTATION WITH ORA CANADA	10
4. INTERFACE - 1ST AND 2ND PROTOTYPE	11
4.1 INTRODUCTION.....	11
4.2 1 ST PROTOTYPE - FLOATING WINDOW.....	11
4.3 2 ND PROTOTYPE - CREATING THE ASCII FILE.....	12
4.3.1 <i>Using Microsoft Word Styles</i>	13
4.3.2 <i>Extracting Z Text</i>	13
5. INTERFACE 3RD PROTOTYPE.....	14
5.1 SETTING UP WINDOWS	14
5.1.1 <i>Specification Window</i>	14
5.1.2 <i>Structured Output Window</i>	15
5.2 THE Z TOOL CONTROL PANEL	16
5.2.1 <i>Insert Schema</i>	16
5.2.2 <i>Exit</i>	16
5.2.3 <i>Z/EVES Commands</i>	16
5.2.4 <i>Interactive Z/EVES</i>	17
5.2.5 <i>Z/EVES Errors</i>	18
6. DYNAMIC DATA EXCHANGE (DDE) AND THE Z/EVES STUB	19
6.1 DYNAMIC DATA EXCHANGE (DDE).....	19
6.2 THE Z/EVES STUB.....	20
7. CONCLUSION.....	22
7.1 SUMMARY	22
7.2 INTERPRETATION.....	22
7.3 RECOMMENDATIONS.....	23
8. ANNOTATED BIBLIOGRAPHY.....	25
9. APPENDICES	28
9.1 1 ST PROTOTYPE - FLOATING WINDOW.....	28
9.1.1 <i>Open Microsoft Word</i>	28
9.1.2 <i>Open New Word Document</i>	29
9.2 THE Z TOOL CONTROL PANEL	30

9.2.1 <i>Form Load</i>	30
9.2.2 <i>Insert Schema</i>	32
9.2.3 <i>Exit</i>	32
9.2.4 <i>Convert Schemas</i>	33
9.2.5 <i>Convert Selected Schemas</i>	34
9.2.6 <i>Add Converted Schemas</i>	34
9.2.7 <i>Z/EVES Errors</i>	35
9.2.8 <i>Interactive Z/EVES</i>	35
9.3 INTERACTIVE Z/EVES.....	36
9.3.1 <i>Execute Command</i>	36
9.3.2 <i>Clear</i>	36

1. Abstract

Computer scientists use formal specifications to verify the requirements of a computer system before they build it. There are currently tools available to type-check these specifications, which ensure that they are complete. Complete specifications decrease the likelihood that a system will be unprepared for an input that could cause it to crash.

The concept of complete specifications is imperative in computer systems that are safety-critical, such as airplanes and nuclear reactors. Unfortunately, the current, very discouraging method to write and type check a specification involves handwriting the formal Z notation with its special symbols and then typing their ASCII representation into the type-checker. The only way to obtain a printed specification with the symbols is to run it through a text formatter, such as LaTeX. Our goal was to integrate a type-checker with an editor that displays the Z symbols. This tool allows the user to edit his specification and run it through the type-checker without having to write anything down and without having to change programs.

We accomplished this goal by using software reuse. Creating an editor and a type-checker from scratch would have required all of the manpower in the computer science department. By using Microsoft Word as our editor and Z/EVES as our type-checker, we were able to create a tool with two undergraduates, Steve Ziegler and myself. We wrote our own code in Visual Basic, which is the “glue” that holds our software tool together. Through this project I hope our tool will make it easier to write specifications and thus encourage computer scientists to write formal specifications for all projects. This will lead to better and safer software.

2. Introduction

2.1 Summary of what, how, and why

For my thesis, I have successfully tested the limits of software reuse by using Visual Basic to link Microsoft Word with the Z/EVES type-checker. This software will allow a user to write a formal specification using a Word interface and check the correctness of it using the Z/EVES tool. This thesis serves two purposes. It reuses code from Microsoft and ORA Canada, held together by my own code to create a new software application. It will also enable the user to type-check the document as it is being written, this will save time during the writing of specifications.

2.2 Problem Definition

2.2.1 Context

My technical advisor, Professor John Knight, has done extensive work with reusable software.¹ He is one of the leaders of the Software Reengineering Group at the University of Virginia. Their goal is to save time during the development of new applications by reusing code that was originally intended for something else.

One of Professor Knight's interests is also in safety-critical applications. This would be an application that cannot afford to make a mistake, such as the flight controls of an airplane or the control system of a nuclear reactor. Work is currently being done developing the control system for the nuclear reactor here at UVA. One of the steps in developing the control system is writing a formal specification. The specification language

¹ Knight, John, "Development of Certification Methods for Reusable Software Parts", SEAS Report No. UVA/530805/CS93/101, March 1993.

we are using at UVA is Z. Several tools available can check the correctness of a specification.² For this project we have chosen to use the Z/EVES type-checker available from ORA Canada.

We are using Visual Basic as a development tool to create software which will be able to write and type-check specifications. Instead of writing our own text editor, we are reusing Microsoft Word code to provide us with an editor. By undertaking this thesis I am experimenting with software reuse while creating a new Z specification tool.

2.2.2 Concepts

A specification is the list of the requirements for a project. For example, if you wrote a specification for a door, you might say that it would have to have a handle, hinges, and a window. Writing a specification in natural language isn't always a good idea because of its ambiguous meanings. For the door specification, the reader is not sure whether I meant a doorknob or a plain handle. A formal specification is a specification that is written in mathematical notation. Mathematical specifications are non-ambiguous. There is only one meaning. A type-checker such as Z/EVES can check the mathematical correctness of a specification. It helps to ensure that the writer has written a correct and complete specification. For a safety critical system such as the nuclear reactor product, it is evident that it is crucial to have a correct specification.

Reusable code is code that is written in such a way that it can be used in many different applications. Also called object-oriented, it is code that is basically an object that can be dropped into different programs to be used. An example would be code that could

sort lists. It may be used to sort test scores or it may be used to sort weights. The sort is written in such a way that it can accomplish both tasks. Then the sort can be used for multiple jobs. This saves time because the code can be written once, but used many times.

2.3 Literature Review

The book Z An Introduction to Formal Methods [2] provides a basic understanding of the Z formal method. This book is quite necessary because my background in formal methods is quite limited. It provides examples and exercises that I can work through to increase my understanding of Z. In the same vein, I searched for practical applications of Z. In the journal article “Formal Specification and Documentation of Microprocessor Instruction Sets” [1], J.P. Bowen discusses the use of Z to define a microprocessor based system. The journal article “Rapid Prototyping of Software Systems Using Prolog” [3] by S. Ekambareshwar discusses the method of generating a rapid prototype for software that has been formally specified using Z. These articles helped to give me an idea about how Z is currently being used in industry.

In order to learn more about Z tools, I needed to go to the World Wide Web to find state of the art tools. The site “Z Soup” [7] provides a listing and explanation of most of the current Z tools that are available. For each tool it lists the manufacturer, the functionality of the tool, and what operating system it runs on (Sun or Windows).

2.4 Rationale for and Scope of the Project

In the past, a user who wanted to use the Z/EVES type-checker needed to convert their specification, with its special mathematical symbols, into ASCII characters (the

² “Z Soup” URL = “http://www.scranton.com/~bschnell/Z_soup.html/#B”

characters on the keyboard). By combining the power of Microsoft Word with Z/EVES we have an editor where the user doesn't need to make any conversions. They can type in their specification with the special mathematical symbols using a Microsoft Word interface. Then they can type-check it from Word by pressing one of the buttons that we have added to the interface. My contribution is to design the interface by which a user can control Z/EVES from inside the Word interface.

2.5 Overview of the Contents of the Rest of the Report

Chapter Three will cover the Project Organization. It will discuss the preliminary research that we did and the design approach that we took. It will also discuss our choice of editor and our consultations with the designers of Z/EVES, ORA Canada. Chapter Four will discuss the beginnings of our user interface. It will cover the first and second prototypes. Chapter Five will cover our final version of the interface. It will discuss how we set up Microsoft Word and how we designed the Z Tool Control Panel. Chapter Six will provide background on Dynamic Data Exchange. It will also cover the use of the Z/EVES stub to simulate Z/EVES. Chapter 7 is the conclusion. It will contain a summary of the results of this report in addition to an interpretation of them. I will also provide my recommendations. Chapter Eight is an annotated bibliography. Chapter Nine contains the Appendices and holds all of the Visual Basic code that I have written with brief explanations.

3. Project Organization

3.1 Introduction

All software projects require a fair amount of planning, and ours was no exception. We began by doing some preliminary research, to discover what other types of software tools are available. If a tool existed that could accomplish our goals, than we would not waste our time creating another one. Our next job was to determine what type of editor we would link to our type checker. We also needed to decide what our design approach would be.

3.2 Preliminary Research

One of our first jobs was to find out what types of Z specification tools were currently available. We were looking for a tool which has functionality such as type-checking and theorem proving while providing the user with an editor to write his specification. We used several Internet search engines such as Altavista and Yahoo to try to find out as much as possible about current Z tools. There was only one tool which was close to providing the functionality that we desired, the CaDiZ tool by York Software Engineering. When we investigated further, we found that the status of the tool was suspicious, and that it may not include all of the functionality that York claims on its web page. At this point we decided to develop our own tool. We chose the Z/EVES specification tool by ORA Canada. This tool has type checking and theorem proving capabilities, but lacks an editor. All input to the tool is through the use of ASCII files.

3.3 Choice of Editor

Professor Knight had been writing his Z specifications using Framemaker, and recommended this editor as a good place to start. The Microsoft Windows version of Framemaker has a disk that allows programmers to use Object Linking and Embedding (OLE) to access the functionality of Framemaker. To make sure that this would work for us, we tried one of the example programs that came with the disk. We had many difficulties with this simple program, and actually found an obvious mistake. Framemaker is usually run on the UNIX operating system, and it appeared that there were still some difficulties with the Windows version. Some of the code actually indicated it was for Windows version 3.1, and we were using Windows 95. At this time we decided to go with another editor.

One of our goals was to develop a tool that could be run on a PC using a Windows 95 interface. We decided to try Microsoft Word as our interface, using Microsoft Visual Basic to access the functionality of Microsoft Word. Since we were using Windows 95, Word, and Visual Basic, all Microsoft products, we figured that our chances of success were much greater. Microsoft developed OLE technology, and there was a substantial amount of literature available that described how to use Visual Basic to drive Word. We stayed with this editor for the duration of the project.

3.4 Design Approach

Our goal was to design a prototype which would demonstrate that the key functionality of this software tool could be attained. Therefore, our tool did not have to include every single piece of functionality, but it needed to show that every major concept

was possible. It needed to take the basic process of writing and type checking a specification from beginning to end. The user of our prototype would need to write the specification, type check it, and learn of any errors.

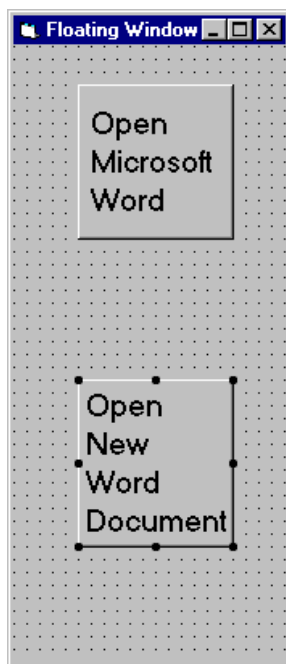
3.5 Consultation with ORA Canada

For our project to be successful, it was crucial to be in frequent contact with the developers of the Z/EVES specification tool, ORA Canada. We had one teleconference and many electronic mails discussing the ways in which our editor could communicate with their tool. The designers at ORA Canada were very helpful, and actually added some functionality to their own specification tool to help it interact with our editor. This became very important when we were using Dynamic Data Exchange (DDE).

4. Interface - 1st and 2nd Prototype

4.1 Introduction

We developed three prototypes of our Z tool. The goal of the first prototype was to set up Microsoft Word so that we could write specifications using the Z notation. The second prototype was to see if we could successfully extract Z text from a document and convert it into ASCII text. Before we could test the more robust features of Z/EVES, we had to first make sure we could achieve the basic functionality. After successfully creating the first two prototypes, we proceeded with a third prototype which included the functionality of the first two in addition to more features of Z/EVES.



4.2 1st Prototype - Floating Window

I developed our first prototype in **Figure 4-1** using Microsoft Visual Basic. When this program runs, this form is displayed by itself. By clicking on the button “Open Microsoft Word”, the user can open Microsoft Word. This takes advantage of Object Linking and Embedding (OLE). Though OLE, I can make function calls to Microsoft Word through Visual Basic. This essentially allows me to control Microsoft Word through Visual Basic. Visual Basic can almost be seen as a “remote control” for Microsoft Word.

Figure 4-1

After Microsoft Word opens, this form continues to sit on top of Word. Usually, inactive windows are not visible. The active Word window would sit on top of the inactive form. In later prototypes, this form would eventually become the

control panel that would hold the buttons to access Z/EVES. An important part of this prototype was to see if we could always have the control panel rest on top of all the other windows, even when it is inactive. This way, the user is always aware of what Z/EVES functions are currently available. The ability to let the inactive window lie on top of the active windows is not a normal capability of Visual Basic. I found the code to accomplish this through a news group on the Microsoft web site.³

The second button opens a new Word window in the copy of Microsoft Word that is currently open. It then types the word “like” into the new document and activates the thesaurus. This button allowed us to make sure we could access the Word document from Visual Basic and access different features of Word such as the thesaurus.

4.3 2nd Prototype - Creating the ASCII File

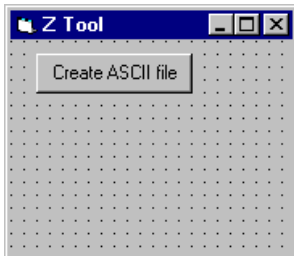


Figure 4-2

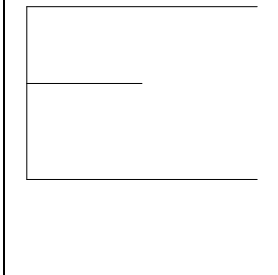
Steve Ziegler developed the second prototype in **Figure 4-2**, which extracted the Z text and created the ASCII file necessary for Z/EVES. In order to explain how he accomplished this, I need to explain how Microsoft Styles work and how we used them to extract Z text.

³ Microsoft URL = “www.Microsoft.com”

4.3.1 Using Microsoft Word Styles

In order to create the graphical schemas used by Z specification writers, I made

Figure 4-3



use of the Microsoft Word facility to create custom paragraph styles. A paragraph style is a combination of character formats and paragraph formats that are identified by a specific style name.

Using styles it is possible to create types of paragraphs which have their own fonts and borders. I created paragraph styles which

looked like **Figure 4-3**, which are the schema formats used by Z specification writers. By selecting the appropriate style from Microsoft Word, a user could insert a blank schema into their document. Any text written inside this blank schema was automatically Z text, since I set the default font in this style to Z text.

4.3.2 Extracting Z Text

Steve wrote the algorithm which extracts the Z schemas from within a Z document. Steve's code locates Z text by locating the Z styles that I created. It makes use of a Microsoft Word Basic function which can identify different types of styles. Steve's code then converts the extracted Z text into ASCII. This ASCII code is then copied to a new file. This new file is now in the format used by the Z/EVES type checker.

5. Interface 3rd Prototype

Our third and final prototype contains all of the functionality that we wished to include in our project. It takes all of the features of the first two prototypes and combines them with the functionality of Z/EVES. In the first prototype, the program would execute and the control panel would appear. The user would then press a button on the control panel to open Microsoft Word. In the 3rd prototype, I modified this so that when the program executes, Microsoft Word automatically opens with our custom control panel floating on top. The rationale for this was that the user would never be using the control panel without using Microsoft Word. The rest of this section describes how we set optimize Microsoft Word to work with our code and how we developed the final version of our control panel.

5.1 *Setting Up Windows*

5.1.1 Specification Window

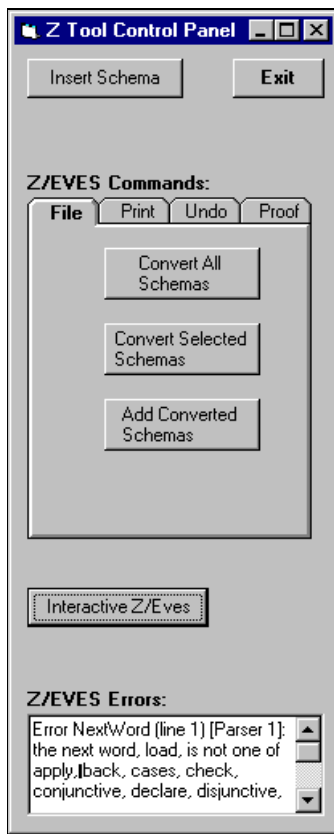
In order to make sure that the Z styles were always available, we created a “Z Document” template which we added to the template directory. There are several templates to chose from when starting a new Word Document, such as a Letter, Memo, or Manuscript. The standard choice is Normal, which is essentially a blank document. We added our new Z styles to the Normal template and saved it as a “Z Document”. When the program runs, Microsoft Word opens with a blank Z Document. This way, a user can immediately begin typing in their Z specification.

5.1.2 Structured Output Window

After running the converted specification through the Z/EVES type-checker, Z/EVES responds with two streams of output. The first is the set of error messages. These are sent to a text box on the control panel which will be discussed later. The second stream of output is a set of diagnostics and other information, the exact content depends upon what the user asked Z/EVES to do. To display this structured output, we created a second Microsoft Word window. This window is placed directly below the specification window and is titled “Output.”

5.2 The Z Tool Control Panel

The Z Tool Control Panel in **Figure 5-1** allows us to access all of the functionality of Z/EVES. It includes the ability to insert schemas, access the Z/EVES type-checker, and display error messages from Z/EVES.



5.2.1 Insert Schema

The “Insert Schema” button inserts a blank schema into the document. I accomplished this by calling Word Basic functions from my Visual Basic code to insert the Z styles that we had already developed and installed in the templates directory. The user can simply type their Z text inside the schema. The schema will automatically expand in case the schema is longer than the allotted space.

5.2.2 Exit

The “Exit” button exits the Z Tool. It shuts down Microsoft Word and the Z Tool Control Panel.

Figure 5-1

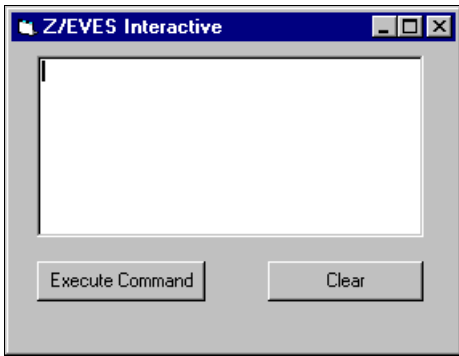
5.2.3 Z/EVES Commands

The Z/EVES Command consists of a file folder system of buttons which access specific Z/EVES functions. The buttons in the “File” section consist of commands which convert the Z/EVES specification written in the “Specification Window” to the ASCII text

needed by the type-checker. The first button, “Convert All Schemas”, converts all of the schemas in the document that are in the “Specification Window.” The second button, “Convert Selected Schemas”, only converts those schemas that have been highlighted by the user. The third button, “Add Converted Schemas”, sends the schemas that have been converted to ASCII to the Z/EVES tool that is currently running. The tabs “Print”, “Undo”, and “Proof” contain buttons which access different characteristics of Z/EVES. For this prototype they have no code associated with them.

5.2.4 Interactive Z/EVES

In our group of Z/EVES Commands we included many of the common uses of the Z/EVES type-checker. There are many more commands for Z/EVES than we had room to place on the control panel. To allow the user to access these commands we added the



button “Interactive Z/EVES” to the control panel.

Pressing this button brings up the window shown in

Figure 5-2. This dialog box acts as the command

line input device that the stand-alone version of

Z/EVES has. The user can type in their command

and press the button “Execute Command.” If he

Figure 5-2

wishes to execute another command, he can press the

“Clear” button first, type in the new command, and then press “Execute Command” again.

We also found from speaking with several Z/EVES users that some people actually prefer

to type in their own commands. The “Interactive Z/EVES” window allows these users to

type in any Z/EVES commands, even the ones we have already provided buttons for. The

“Z/EVES Interactive” form always floats on top of Microsoft Word, even when not active. The user can close the window by clicking in the corner.

5.2.5 Z/EVES Errors

The text box “Z/EVES Errors” displayed in the Z Tool Control Panel (**Figure 5-1**) displays all output from the error stream of the Z/EVES tool. There is a scroll bar on the side of the window so that the user can scroll through all error output.

6. Dynamic Data Exchange (DDE) and the Z/EVES Stub

Dynamic Data Exchange (DDE) is the method by which two Windows programs can communicate with each other. It is the predecessor to OLE, and is useful in cases where both programs are not OLE compatible. For this project, we use DDE to communicate between Z/EVES and our Z Tool. Steve Ziegler wrote the code for DDE communication and developed the Z/EVES stub, which simulates the Z/EVES type-checker. We used the stub instead of the actual Z/EVES tool because Z/EVES had not yet been modified by ORA Canada to work with our program.

6.1 *Dynamic Data Exchange (DDE)*

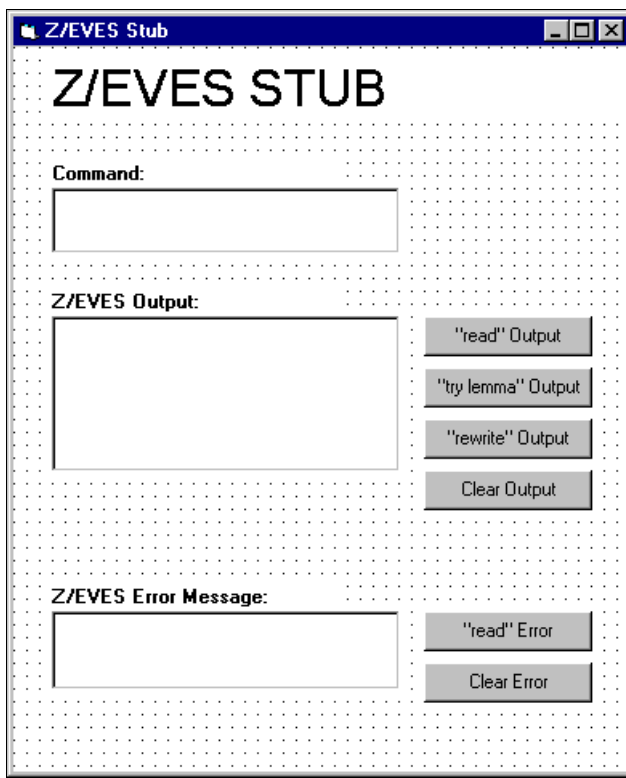
Dynamic Data Exchange (DDE) is the means by which our Z Tool communicates with the Z/EVES stub, which is simulating Z/EVES. After Z/EVES has been modified to work with our program, it will take the place of the Z/EVES stub and the relationship between our tool and Z/EVES will remain the same as it is between our tool and the Z/EVES stub. The easiest way to picture DDE is to picture a conversation between two parties. The party with data to share is called the **server**. The party that is receiving the data is called the **client**. It is possible for both parties in a conversation to play both roles, if both are sending and receiving information.

When the user sends the converted Z text to the Z/EVES type-checker, our Z tool sends a DDEInitiate message to Z/EVES, initiating the conversation. Here, our tool is the client and Z/EVES is the server. Z/EVES sends a DDEAcknowledge message back to the Z tool, effectively agreeing to the conversation. The Z tool sends the ASCII file containing the schemas in a DDEPoke message. The Z/EVES tool processes the file, and

sends the error output and structured output in two separate DDEPoke messages back to the Z tool. The client, the Z tool, sends a DDEAcknowledge message to indicate that the data has arrived. The Z tool then sends a DDETerminate message to terminate the conversation. This is the basic way that our Z tool and the Z/EVES stub communicate with each other.

6.2 The Z/EVES Stub

The Z/EVES Stub in **Figure 6-1** was created by Steve Ziegler as a way for our Z tool to simulate communicating with the Z/EVES program. Clicking on one of the four



buttons next to the “Z/EVES Output” dialog box sends one of four predetermined messages to the “Output” window on our Microsoft Word interface. I wrote the code which places the text inside the window. The first three button simulate a “read” output, a “try lemma” output, and a “rewrite” output. The text is also placed in the “Z/EVES Output” text box on the Z/EVES stub. The last button, clear output, removes the

Figure 6-1
output from the text box in the stub.

The bottom dialog box and two error buttons simulate error messages. The top button simulates a “read” error. This predetermined error message is sent to the text box “Z/EVES Error” on the Control Panel. The error message also appears in the “Z/EVES Error Message” box on the stub. The bottom button, clear error, clears the text box on the stub and the “Z/EVES Error” text box on the Control Panel.

It was key that we were able to simulate passing data back and forth from our Z tool to the Z/EVES stub. Our interface and conversion code would be useless without the ability to communicate with Z/EVES. By using the stub, we were able to make sure that by using DDE we could send data to the stub and send structured output and error messages to the correct locations in our interface.

7. Conclusion

7.1 Summary

Steve Ziegler and I began our thesis project doing research to determine if there was an effective Z tool commercially available. Our criteria was that it would have to contain an editor for the user to write a specification using the special Z symbol. This tool would also have to include a type-checker. After determining that no such tool existed, we looked into the feasibility of developing our own.

At the encouragement of Professor Knight, we made Framemaker our initial choice as editor and Z/EVES our choice as type-checker. We found Framemaker difficult to manipulate and determined it would not serve our needs. We changed the editor to Microsoft Word. Steve and I also determined that it would be too difficult to complete an entire, robust software tool, and that it would be better to first build a prototype to determine the feasibility of a large scale project.

I took on the responsibility for developing the user interface, Steve handled the conversion code and the Dynamic Data Exchange (DDE) code. After several discussion with the company that created Z/EVES, ORA Canada, we were finally able to develop a tool that could access all of the functionality of Z/EVES, while providing the user with the familiar interface of Microsoft Word.

7.2 Interpretation

Our work has generated interest amongst those people that write specifications. The fact that two undergraduates can develop such a powerful tool is a testament to software reuse. In my examination of Framemaker, I found it very difficult to adapt to use

in our application. I found that Microsoft Word was much more suited to our needs, but was still not as powerful as I would have liked. It is my hope that Word 97 provides developers with much more powerful tools to access its functionality.

One of the drawbacks to our tool is that it requires 64 megabytes of RAM. This is because the Z/EVES type-checker was originally designed for UNIX machines and the PC version is a very large program. Since our tool is designed for use by universities and corporations, the additional expense of RAM is actually quite small. I hope that this nominal cost will encourage more people to use our tool to write Z specifications. This in turn will lead to software of higher quality, and more importantly, software that is more reliable.

7.3 Recommendations

Although we have successfully built a Z tool prototype, this matter warrants additional study. One area to be studied is in the matter of Microsoft Word. Word 97 has just been released within the last month and I am under the impression that the Word Basic code has been totally revamped. If this is indeed true, a more robust Word Basic would aid our tool incredibly. It will allow us the ability to draw better schemas on the page and to write better conversion code.

I believe that it will take several people to get the conversion code “bug free”. It appears to be a very difficult task to get the Z text converted to ASCII exactly right. We will need very detailed notes from ORA Canada to aid whoever takes this on. When the conversion code is working, it will be necessary to test our tool out on very large specifications, such as those written by the National Security Agency. In addition, user

testing is also very important. People who write specifications and people who have never written specifications before need to be tested. The interface will need to be remodeled so that it can be used by both groups. Our main goal was to build a tool that people will use. This includes encouraging people who have never written specifications before to write them. If the final version of our Z tool is used to teach people to write Z specifications, then we will have accomplished something very noteworthy.

8. Annotated Bibliography

- 1) Bowen, J.P., “Formal Specification and Documentation of Microprocessor Instruction Sets”, *Microprocessing & Microprogramming*, August 1987.

An example of Z used to define a microprocessor based system. The use of Z allows for the possibility of verification of the instruction set. This example provides insight into current applications of the Z notation.

- 2) Diller, Antoni. Z An Introduction to Formal Methods, John Wiley & Sons, 1994.

This book introduces the Z notation. It provides examples and exercises. It is the book I am using to learn about the structure and syntax of Z.

- 3) Ekambareshwar, S., “Rapid Prototyping of Software Systems Using Prolog”, *Conference on Computing Systems and Information Technology 1989*, August 1989.

This paper is about the method of generating a rapid prototype for software that has been formally specified using Z. It discusses how example specifications can be transformed into Prolog procedures. This paper provides an additional example of current uses of the Z notation.

- 4) Knight, John, “Development of Certification Methods for Reusable Software Parts”, SEAS Report No. UVA/530805/CS93/101, March 1993.

This paper discusses technology that will permit the certification of reusable software parts. Reusable software is much cheaper than building new software. I am building this tool using reusable software, so it is interesting that I can use my own tool to help certify the reusable parts.

- 5) Spivey, J.M., “Specifying a Real-Time Kernel”, *IEEE Software*, September 1990.

This is an application of formal methods to a safety-critical system. It is an example of the Z notation used to model a system. It demonstrates that mathematical techniques are useful in finding design flaws. Another example of practical uses of Z.

- 6) “York Software Engineering” URL = “<http://www.yse.com>”.

Demonstrates the CADiZ tool produced by York Software Engineering (YSE). YSE claims that CADiZ is a WYSIWYG editor that is available for Windows and provides type checking capability. This is a tool in development that may accomplish part or all of our goal. Upon inspection, the CADiZ tool does not appear to be complete so we are not fully considering it as an option at this time.

7) “Z Soup” URL = “http://www.scranton.com/~bschnell/Z_soup.html/#B”

This World Wide Web page contains information on different types of Z tools that are currently available. It also contains links to the companies that provide these tools.

9. Appendices

This section contains the code that I have worked on for this project. It does not include the conversion code, the DDE code, or the Z/EVES stub code, all three of which were written by Steve Ziegler and are best explained by him in his technical report.

9.1 1st Prototype - Floating Window

This is the code for the 1st prototype, the Floating Window.

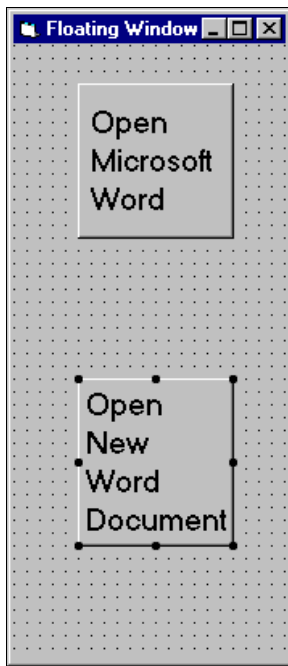


Figure 9-1
top.

9.1.1 Open Microsoft Word

This code creates a new OLE object, Microsoft Word. It opens the new copy of word and places the floating window on

```
Private Sub Command1_Click()  
Set WordObj = CreateObject("Word.Basic")  
WordObj.AppShow  
X = WordObj.AppMaximize()  
If X = 0 Then  
    WordObj.AppMaximize  
End If  
  
success% = SetWindowPos(Form1.hWnd, HWND_TOPMOST, 0, 0, 0, 0,  
    FLAGS)  
  
End Sub
```

9.1.2 Open New Word Document

This code opens a new Word document inside the open copy of Microsoft Word. It then inserts the word like into the document and looks it up in the Microsoft Word thesaurus.

```
Private Sub Command2_Click()  
  
Dim MyWord As Object  
Set MyWord = GetObject("", "Word.Basic")  
  
MyWord.FileNew  
MyWord.Insert "like"  
MyWord.ToolsThesaurus  
  
End Sub
```

9.2 The Z Tool Control Panel

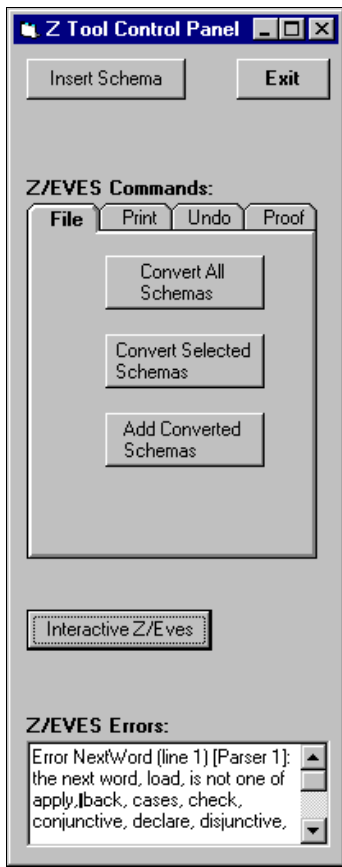


Figure 9-2

9.2.1 Form Load

This is the code that executes when the program starts. The code opens a copy of Microsoft Word and places the floating Control Panel on top. It also creates the two open windows in Word, the top one for writing Z specifications and the bottom one for the structured output that comes back from Z/EVES.

```

Private Sub Form_Load()

"so link event won't fire on form load
StartMeUp = True

ShowWaitForm

success% = SetWindowPos(WaitForm.hWnd, HWND_TOPMOST, 0, 0, 0, 0, FLAGS)
DoEvents
Call SetWaitForm("Starting Word ...", 1, 3)

Set WordObj = CreateObject("Word.Basic")

' Fire up word
WordObj.AppShow
x = WordObj.AppMaximize()
If x = 0 Then
    WordObj.AppMaximize
End If

Call SetWaitForm("Initializing Documents...", 2, 3)
'set up output window
WordObj.filenewdefault
WordObj.FileSaveAs Name:="Output.doc"

' Set up Writing Window
WordObj.FileNew Template:="Z:\ms\msoffice\Templates\Z Document.dot", NewTemplate:=0
WordObj.FileSaveAs Name:="Project Utah.doc"
WordObj.Style "Z Text"

WordObj.WindowArrangeAll

success% = SetWindowPos(Form1.hWnd, HWND_TOPMOST, 0, 0, 0, 0, FLAGS)

Call SetWaitForm("Running Z/EVES...", 3, 3)
Shell (ZEVESPath)

ZEVESError.LinkMode = vbLinkAutomatic
ZEVESOutput.LinkMode = vbLinkAutomatic

Unload WaitForm
Unload SplashForm

End Sub

```

9.2.2 Insert Schema

This button inserts the custom styles we created into the document in order to create blank schemas.

```
'WordObj.Style "Z Text"
'WordObj.Font "Times New Roman"
'WordObj.Insert "_____"
'WordObj.Font "Z"

'WordObj.Insert "      "
'WordObj.Font "Times New Roman"
'WordObj.Insert "_____ "
'WordObj.Insert "_____"
'WordObj.InsertPara

'WordObj.Style "Z - Schema Signature"
'WordObj.InsertPara
'WordObj.InsertPara
'WordObj.Font "Times New Roman"
'WordObj.Insert SchemaMiddleLine
'WordObj.InsertPara

'WordObj.Style "Z - Schema Predicate"
'WordObj.InsertPara
'WordObj.InsertPara
'WordObj.Style "Z Text"
'WordObj.LineUp 1
'WordObj.LineDown 1
'WordObj.LineUp 5
```

9.2.3 Exit

This button closes the Z tool and Microsoft Word.

```
Private Sub Command5_Click()

    Form_Unload (0)
    End

End Sub
```

9.2.4 Convert Schemas

This button converts all of the schemas in the document into the ASCII text needed by the Z/EVES type-checker.

```
Private Sub Command1_Click()

    Dim lo_Converter As New Converter

    Call ShowWaitForm

    If lo_Converter.Setup(App.Path & "\z_latex.tex", 1, 1, WordObj) = 0 Then
        Me.SetFocus
        MsgBox ("Can't setup converter class.")
        Exit Sub
    End If

    Call SetWaitForm("Extracting all schemas ...", 1, 2)
    If lo_Converter.ExtractAllSchemas() = 0 Then
        Me.SetFocus
        MsgBox ("Can't extract Z styles.")
        Exit Sub
    End If

    Call SetWaitForm("Converting all schemas ...", 2, 2)
    lo_Converter.ConvertSchemas

    WaitForm.Hide

End Sub
```

9.2.5 Convert Selected Schemas

This code converts the schemas that have been highlighted by the user to the ASCII text needed by Z/EVES.

```
Private Sub Command19_Click()
    Dim lo_Converter As New Converter

    Call ShowWaitForm

    If lo_Converter.Setup(App.Path & "\z_latex.tex", 1, 1, WordObj) = 0 Then
        Me.SetFocus
        MsgBox ("Can't setup converter class.")
        Exit Sub
    End If

    Call SetWaitForm("Extracting Selected schemas ...", 1, 2)
    If lo_Converter.ExtractSelectedSchemas() = 0 Then
        Me.SetFocus
        MsgBox ("Can't extract Z styles.")
        Exit Sub
    End If

    Call SetWaitForm("Converting selected schemas ...", 2, 2)
    lo_Converter.ConvertSchemas

    WaitForm.Hide

End Sub
```

9.2.6 Add
Converted
Schemas
This
button sends
the converted
schemas to the
Z/EVES type-

checker.

```
Private Sub Command2_Click()
```

```
    ZEVESCommand.Text = "read " & Chr$(34) & App.Path & "\z_latex.tex" & Chr$(34) & ";"  
    ZEVESCommand.LinkMode = vbLinkManual  
    ZEVESCommand.LinkPoke
```

```
End Sub
```

9.2.7 Z/EVES Errors

This text box prints out any errors that are sent to it from the Z/EVES stub.

```
Private Sub ZEVESError_Change()
```

```
    MsgBox "New Error found!"
```

```
End Sub
```

9.2.8 Interactive Z/EVES

This button brings up the Interactive Z/EVES dialog box.

```
Private Sub Command4_Click()
```

```
Form2.Visible = True
```

```
success% = SetWindowPos(Form2.hWnd, HWND_TOPMOST, 0, 0, 0, 0, FLAGS)
```

```
End Sub
```

9.3 Interactive Z/EVES

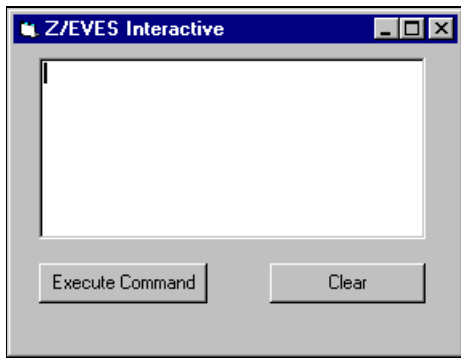


Figure 9-3

9.3.1 Execute Command

This button executes the command that the user has typed into the dialog box.

```
Private Sub Command1_Click()  
  
    txtCommand.LinkMode = vbLinkManual  
    txtCommand.LinkPoke  
  
End Sub
```

9.3.2 Clear

This button clears the dialog box.

```
Private Sub Command2_Click()  
  
    txtCommand.Text = ""  
  
End Sub
```