

Determining the Feasibility Of Building a Windows Z Specification Tool

A Thesis
in TCC402

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

Stephen Vincent Ziegler
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 C. Knight

Approved _____ (TCC Advisor)
Melvin Cherno

Abstract

I have determined that it is feasible to build a Windows Z Specification tool. Through the use of rapid prototyping, I have discovered that a Windows Z Specification tool can be built using Visual Basic, Microsoft Word and Z/EVES. The report describes the problem, scope and rationale of the project. It also describes the interface and functionality of the tool. There is a discussion on the prototyping methodology that was used and how the two major groups of prototypes were built. Each of the initial prototypes and communication prototypes is described in detail. The final chapter presents a summary and interpretation of the findings and also provides recommendations for future work.

Table of Contents

Abstract	i	
Table of Contents	ii	
List of Figures	iii	
1. Chapter 1 - Introduction	1-1	
1.1 Thesis Statement	1-1	
1.2 Problem Definition	1-1	
1.3 Literature Review	1-3	
1.4 Rationale and Scope	1-4	
1.5 Overview of Contents	1-5	
2. Chapter 2 - Z Tool Requirements	2-1	
2.1 Problem Definition	2-1	
2.2 WYSIWYG Development Environment	2-2	
2.3 Usable Interface	2-5	
2.4 Z/EVES Analytical Engine	2-6	
3. Chapter 3 - Prototyping Methodology	3-1	
3.1 Determine Feasibility	3-1	
3.2 Rapid Application Development	3-1	
3.3 Modules	3-2	
3.4 Plan For Prototypes	3-2	
4. Initial Prototypes	4-1	
4.1 Risks Addressed	4-1	
4.2 Use of OLE	4-1	
4.3 Z Font Characteristics		4-1
4.4 Locate Z Text	4-2	
4.5 Draw Schemas	4-4	
4.6 Extract Z Schemas	4-6	
4.7 Convert Schemas To LaTeX Format	4-9	
5. Communication Prototypes	5-1	
5.1 Risks Addressed	5-1	
5.2 Control Panel	5-1	
5.3 Visual Basic Z/EVES Stub	5-2	
5.4 First Z/EVES Stub	5-4	
5.5 Second Z/EVES Stub	5-5	
5.6 Z/EVES With DDE Interface	5-5	
6. Conclusion	6-1	
6.1 Summary	6-1	
6.2 Interpretation	6-1	
6.3 Recommendations	6-3	
7. Appendix	7-1	
A. Annotated Bibliography	7-1	
B. Z Character Set	7-4	
C. ZFONT1.DAT Contents	7-6	
D. Control Panel	7-8	
E. Splash Form	7-13	
F. Interactive Z/EVES Form	7-14	
G. Converter Class	7-15	
H. Module File	7-28	

List of Figures

Figure 2-1: Overall Z Tool Architecture	2-1
Figure 2-2: Z/EVES LaTeX Interface	2-3
Figure 2-3: WYSIWYG Z Tool Interface	2-4
Figure 2-4: Z Boxes	2-5
Figure 2-5: Toolbar	2-6
Figure 2-6: Specific Z Tool Architecture	2-7
Figure 4-1: Document With Text and Schemas	4-3
Figure 4-2: Schema Drawn Using One Style	4-5
Figure 4-3: Schema Drawn Using Two Styles	4-6
Figure 4-4: Schema Extraction and Conversion Process	4-8
Figure 5-1: Visual Basic Z/EVES Stub	5-3
Figure 5-2: Z/EVES With DDE Interface	5-5
Figure 6-1: Final Z Prototype	6-2

Technical Advisor's Evaluation

Date: March 24, 1997

To: John C. Knight, Technical Advisor

From: Stephen Vincent Zielger, Student

Subject: Technical Advisor's Evaluation of my Thesis

Return Completed Evaluation to: Melvin Chernob, TCC Advisor

The role of the technical advisor on the undergraduate thesis is a very important one. To help judge fairly the overall quality of the undergraduate thesis, please comment on the technical aspect of my work and the working relationship I established with you. Please rate the technical quality of the completed work according to the following check list, using a scale from one to five.

5=excellent; 4=good; 3=average; 2=poor, but still acceptable; 1=unacceptable

Please feel free to add any notes below or on the back of this sheet.

- | | |
|-----------|---|
| 1 2 3 4 5 | I. Did I maintain an appropriate working relationship with you throughout the project? |
| 1 2 3 4 5 | II. Based on the following criteria, how do you rate the overall technical quality of this thesis? |
| | III. Detailed Evaluation: |
| 1 2 3 4 5 | a. Are the subject and scope of the project clearly stated? |
| 1 2 3 4 5 | b. Is the conclusion consistent with the statement of subject and scope? |
| 1 2 3 4 5 | c. Is the conclusion adequately supported by data, theory, or calculations in the body of the paper? |
| 1 2 3 4 5 | d. Is the level of work reported consistent with what you agreed would be a reasonable project? |
| 1 2 3 4 5 | e. Does the paper show unusual merit as a result of my contribution, such as imaginative conception, innovative approach, or insights into the subject? |
| 1 2 3 4 5 | f. Does the paper show skills in handling engineering or other technical concepts at a level appropriate to an undergraduate degree candidate? |
| 1 2 3 4 5 | g. Does the bibliography show a sound background in the literature appropriate to the subject? |
| 1 2 3 4 5 | h. Was this thesis part of a project course in the major field? |
| 1 2 3 4 5 | i. Would you like to discuss my work with my TCC Advisor? |

Approved _____, Technical Advisor Date _____

TCC Advisor's Evaluation

Author's Name: Stephen Vincent Ziegler

TCC Advisor: Melvin Chernov

Technical Advisor John C. Knight

5=excellent; 4=good; 3=average; 2=poor, but still acceptable; 1=unacceptable

5 4 3 2 1

Overall Design: This project is imaginatively conceived, thoroughly researched, convincingly expressed, and/or potentially significant.

5 4 3 2 1

Introduction: Establishes meaningful context and significance of project for multiple audiences; grounds research thoroughly in literature; clearly defines problem and scope; explains key concepts; previews structure of rest of report.

5 4 3 2 1

Methods: Adequately justifies and clearly describes project design.

5 4 3 2 1

Results: Presents data with verbal and visual clarity; emphasizes central research finding; fully explains and illustrates design and use of final product or conclusion.

5 4 3 2 1

Interpretation: Conclusions follow logically from data and/or argument; reaches synthesis which is related to context established in introduction; assesses ethical and social impact; recognizes limitations of this research and makes recommendations for future work.

5 4 3 2 1

Organization: Logically divided into unified and coherent chapters, sections, and paragraphs; transitions link each chapter to the whole and establish flow between sentences and paragraphs.

5 4 3 2 1

Graphics: Includes enough appropriate and clearly understandable figures and tables which are well integrated with text, adequately interpreted, clearly labeled and captioned, large enough and printed darkly.

5 4 3 2 1

Documentation: Adequately supports argument with references cited in the text, using correct and consistent style; bibliography shows thorough depth and breadth and is correct in form.

5 4 3 2 1

Style: Clear, concise, readable, mature, and polished; minimum of grammatical errors.

5 4 3 2 1

Mechanics: Chapter and section headings meaningful and consistent; proper use of formatting conventions; pages numbered; capitalization standard; minimum of typographical errors.

Supporting Documentation:

5 4 3 2 1

Title: accurate, complete and concise.

5 4 3 2 1

Abstract: miniature of report, summarizes argument and conclusions

5 4 3 2 1

Table of Contents and List of Figures: neat and complete.

5 4 3 2 1

Appendices: informative, and neatly presented.

CURRENT GRADE: _____ **GRADE AFTER REVISION:** _____

1. Chapter One - Introduction.....	9
1.1. Thesis Statement	9
1.2. Problem Definition	9
1.3. Literature Review	10
1.4. Rationale and Scope	10
1.5. Overview of Contents	11

Chapter One - Introduction

Thesis Statement

I have determined that it is feasible to build a Windows Z Specification tool. Through the use of rapid prototyping, I have discovered that a Windows Z Specification tool can be built using Visual Basic, Microsoft Word and Z/EVES. This Windows-based tool, in contrast to the existing DOS-based tools, provides a user-friendly interface that uses the natural Z language and characters.

Problem Definition

Context

The specification phase is the stage of the software life cycle where software engineers detail what the software is supposed to do. It is one of the most important phases in the software life cycle because errors in the specification phase will trickle down through the design, implementation and testing phases (<http://www.cis.ohio-state.edu/hypertext/faq/usenet/z-faq/faq-doc-1.html>). To correct these errors, software engineers must start all over and repeat the specification, design, implementation and testing phases.

Researchers have developed formal specification methods to help software engineers with the specification process. These formal specifications are based on mathematics, not natural language, to reduce ambiguity between personnel on the development team. Formal methods also provide exact and proven theorems for the specification. Z (pronounced "zed") specification is one formal technique to detail software requirements. The technique is based on set theory and first order predicate logic which uses special characters and mathematical symbols (http://www.scranton.com/~bschnell/zed_soup.html).

There are many tools on the market that help software engineers create Z specifications for large software projects. These tools provide type checking and also perform theorem proving to ensure that the specification is complete. However, the tools are DOS-based and do not provide an interface that uses the natural Z language. These tools use the awkward LaTeX representation of the Z language. Therefore, developers must maintain two versions of the specification: a modern word processor version with the appropriate Z symbols and the LaTeX version which can be analyzed by the current Z tools. I have researched the feasibility of building a tool that will allow software engineers to develop Z specifications in Microsoft Word using the Z symbols and then automatically convert the specification to the LaTeX format for analysis.

Concepts

The Z specification technique was developed by the Programming Research Group at Oxford University (<http://www.cis.ohio-state.edu/hypertext/faq/usenet/z-faq/faq-doc-1.html>). Researchers worked on the formal method during the late 1970s. The method has now gained popularity in Europe and some parts of the United States.

Z is a specification language not a programming language. Therefore, it has no compiler or linker. The Z Soup page (http://www.scranton.com/~bschnell/zed_soup.html) offers a simple explanation of Z. A Z specification is a collection of schemas which contain the entities and relationships of the specification.

The following is a schema named Personnel. The schema consists of the schema name, signature and predicate.

Personnel	(Schema Name)
employees : . PERSON boss-of : PERSON \downarrow PERSON salary : PERSON \downarrow .	(Signature Section)
. salary = employees ie : employees @ salary(e) < salary(boss-of e)	(Predicate Section)

The signature is the top half of the schema and contains the declarations of the set-theoretic objects. The predicate contains the relationships between the entities in the signature. Each predicate statement must hold true for the entities. Schemas can be inherited from other schemas to produce larger specifications. Functions can be defined on the schemas to model data structures.

Literature Review

Many of the current Z tools have Web pages that describe the software and provide an overview of the capabilities. The following is a list of some of the major Z tools available.

fuzz Package - <ftp://comlab.ox.ac.uk/pub/Zforum/fuzz>

The fuzz package is a set of tools which provides syntax and type checking of Z specifications. The package uses the LaTeX style format.

Zola - <http://www.cis.ohio-state.edu/hypertext/faq/usenet/z-faq/faq-doc-6.html>

Zola is a Z tool for Sun workstations. The tool gives developers support for the development, proving and maintenance of specifications. It also provides a What You See Is What You Get (WYSIWYG) interface which means that developers can create Z documents in a format similar to Word documents. When the developer prints the file, the output will look just like what is on the screen. Zola provides a type checker and a theorem prover. The Zola tool provides the functionality that the UVA Software Engineering Group is looking for but the software does not run on PCs.

CADiZ - ftp://ftp.cs.york.ac.uk/hise_reports/cadiz

The CADiZ tool seemed to be a good solution to the Z specification problem. The documentation stated that CADiZ is a suite of tools that interfaces with Microsoft Word. It provides a simple WYSIWYG interface in Word. It also provides type checking directly from the Word interface. However, it is not clear if the CADiZ tool has even been completed. Also, the tool does not seem robust. The tool developed in Visual Basic could provide an interface to many different Z tools and not just the one provided with CADiZ.

Z/EVES - <http://www.ora.on.ca/z-eves/welcome.html>

The Z/EVES tool provides syntax and type checking, schema expansion, precondition calculation, domain checking and general theorem proving. The Z/EVES tool is available on many platforms including Windows 95. The tool needs LaTeX style input and can read files written in this format. The Z specification tool has an interface with the Z/EVES package.

Rationale and Scope

As stated earlier, various companies have built Z specification tools, but only allow the LaTeX representation to be used. I have researched the possibilities of building a tool which allows the user to build a specification using the Windows-based Z language and then use the tool to automatically convert it to the DOS-based LaTeX format. This will allow the software engineer to concentrate on building the specification and not worry about simple syntax errors that can arise from using the LaTeX format. This

tool will also allow the engineer to use existing Z specification tools to analyze the specification. The scope of this project is to determine the feasibility of building such a Z tool and to build prototypes of the tool to address major risks.

Overview of Contents

The Introduction describes the problem, scope and rationale of the project. Chapter Two describes the interface and functionality of the tool. Chapter Three details the prototyping methodology that was used and explains the two major groups of prototypes built. The fourth chapter explains the initial prototypes in detail. Chapter Five explains the final set of prototypes and illustrates the module communication. The final chapter presents a summary and interpretation of the findings and also provides recommendations for future work.

2. Chapter Two - Z Tool Requirements	12
2.1. Modern Word Processor	12
2.2. WYSIWYG Development Environment	12
2.3. Usable Interface.....	15
2.4. Z/EVES Analytical Engine.....	16
Figure 2-1: Overall Z Tool Architecture.....	12
Figure 2-2: Z/EVES LaTeX Interface	13
Figure 2-3: WYSIWYG Z Tool Interface	14
Figure 2-4: Z Boxes.....	15
Figure 2-5: Toolbar	16
Figure 2-6: Specific Z Tool Architecture.....	17

Chapter Two - Z Tool Requirements

Professor Knight developed most of the software requirements for the tool. He developed the idea of building a easy-to-use Z specification tool that could run under Windows. The requirements were modified as different prototypes were built but the overall structure of the system has remained the same. Figure 2-1 is a diagram of the overall system architecture. The Z tool is a control system which runs and manipulates a modern word processor and a Z analytical engine.

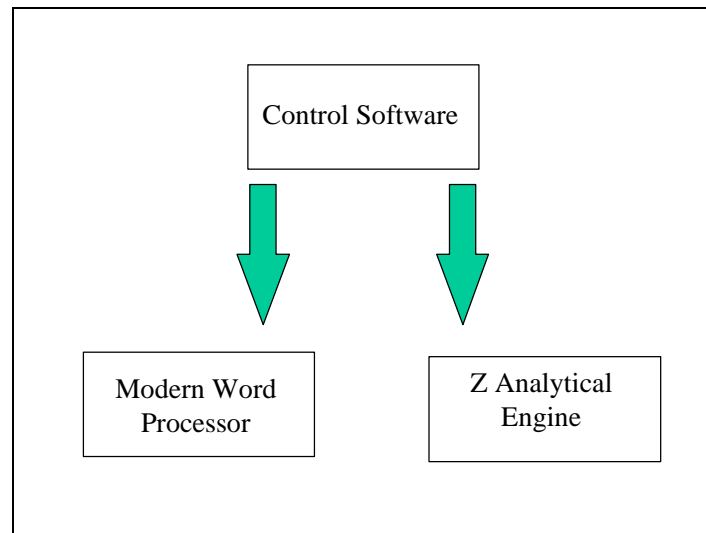


Figure 0-1: Overall Z Tool Architecture

Modern Word Processor

This first requirement of the system is that it provides modern word processing power to the user. The user must be able to develop a professional specification document. Microsoft Word provides powerful processing power and a programmable interface through Visual Basic.

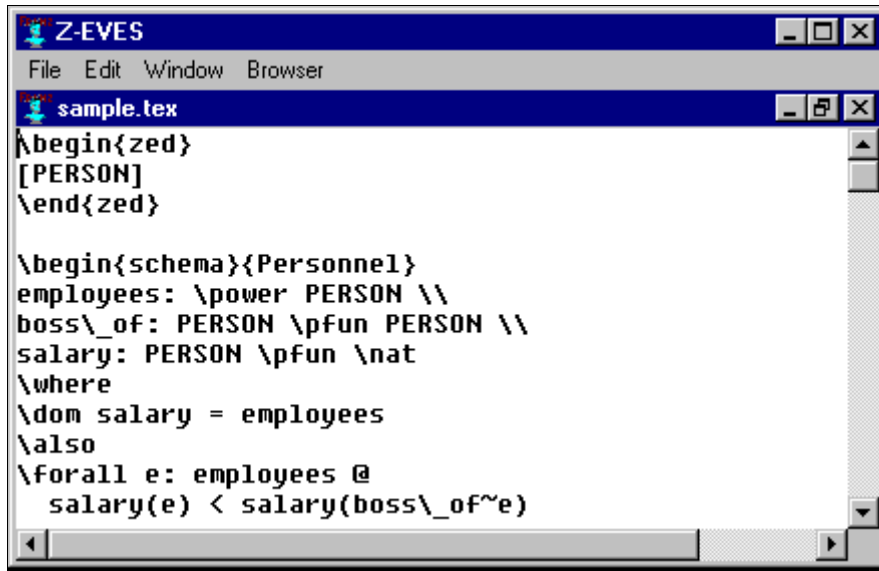
Microsoft Word allows the user to build large specification documents. Software engineers can use Word to produce documents that contain headers, footers, page numbers, automatic outlining and other professional features. Also, Microsoft Word is compatible with Microsoft Excel spreadsheets and Microsoft Access databases. Users also have modern file capabilities when using Word. Word provides the user with standard file editing capabilities, such as editing, saving, copying and deleting documents. The user can also integrate multiple Word documents into one document. Microsoft Word has full printing capabilities including a print-preview functionality.

Microsoft Word has a full Object-Linking and Embedding (OLE) Automation interface. This allowed the development team to manipulate Microsoft Word through the control software. By using Word, we employed large-scale component software reuse. The development team could focus efforts on addressing the major risks of the Z tool instead of trying to build a modern word processor that is compatible with the Z tool.

WYSIWYG Development Environment

The Z tool must provide a What You See Is What You Get (WYSIWYG) development environment. This means that when the user prints the document from the word processor, the printed document will look exactly like the document on the screen. As stated in Section 1.2.1, the existing Z analytical tools only provide the awkward LaTeX interface.

Figure 2-2 shows the LaTeX interface for Z/EVES.



```
Z-EVES
File Edit Window Browser
sample.tex
\begin{zed}
[PERSON]
\end{zed}

\begin{schema}{Personnel}
employees: \power PERSON \
boss\_of: PERSON \pfun PERSON \
salary: PERSON \pfun \nat
\where
\dom salary = employees
\also
\forall e: employees @
  salary(e) < salary(boss\_of~e)
```

Figure 0-2: Z/EVES LaTeX Interface

As shown in the figure, the LaTeX style does not use the standard Z characters and schemas but uses DOS-based characters to represent the Z symbols. For example, the LaTeX format uses the strings `\power` and `\pfun` to represent the Z characters \cdot and ϕ .

Figure 2-3 shows a WYSIWYG interface for the Z tool. The WYSIWYG interface uses the standard Z font and characters. The figure shows a standard schema including the vertical and horizontal lines for the schema boundaries.

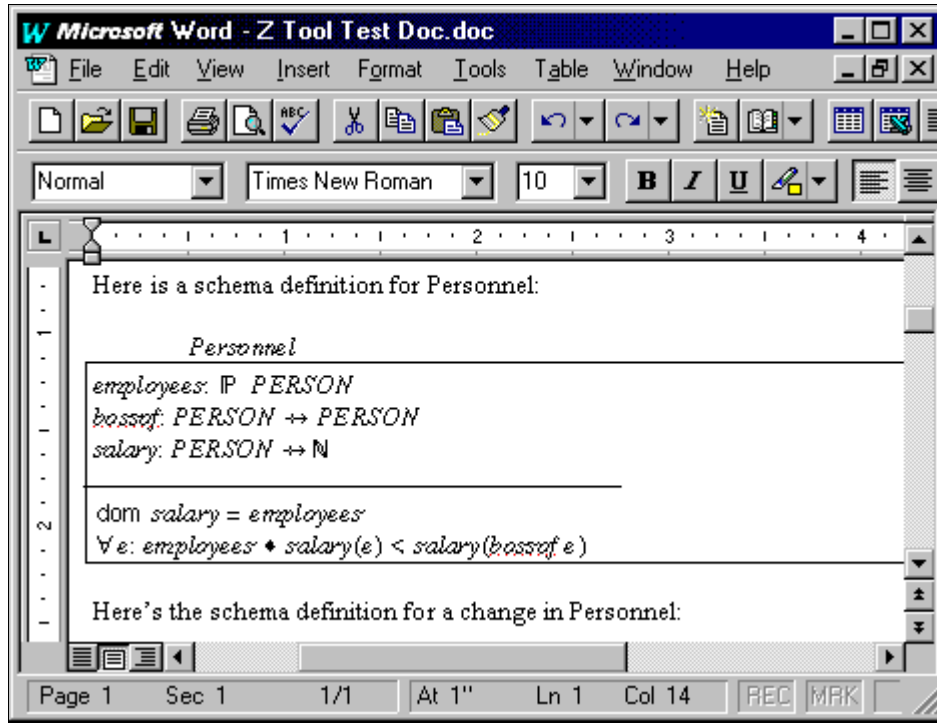


Figure 0-3: WYSIWYG Z Tool Interface

The Z tool must have a button on the interface to insert empty Z schemas into the document. The user can enter text into the schema and the schema must expand as more lines are added.

The WYSIWYG development environment needs to be able to handle the three main types of Z boxes: axiomatic, schema and generic. Figure 2-4 is a diagram of each Z box.

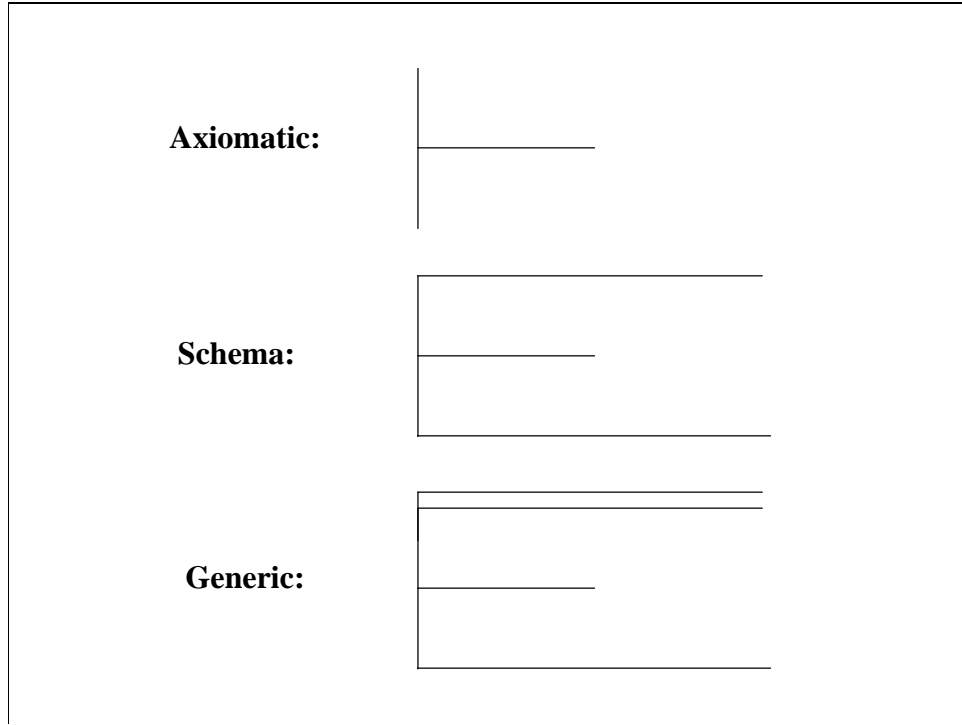


Figure 0-4: Z Boxes

Usable Interface

The Z tool must also have a usable graphical user interface (GUI). The current Z tools use a command line editor to run the tool. The user must be familiar with all of the tool's commands in order to run the tool. A GUI for the Z tool will allow novice users to simply click on buttons to add schemas to the document or to type check the schemas. The tool must also allow expert users to enter Z/EVES command strings to perform complex theorem proving.

The GUI can be developed in Visual Basic (VB). Developers can quickly develop complex GUIs using Visual Basic. Developers can also write code to perform OLE Automation which can control Microsoft Word.

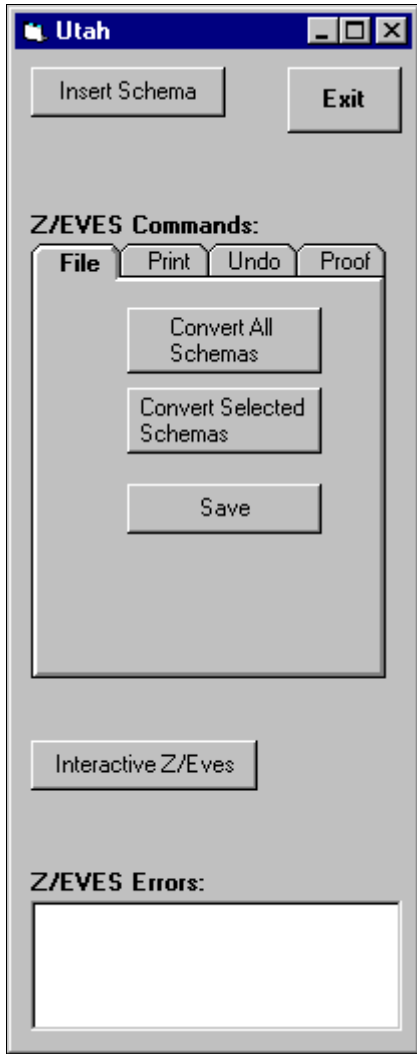


Figure 0-5: Toolbar

The tool should provide a “floating” toolbar which contains the necessary buttons to run Z/EVES. The toolbar will “float”, or stay on top, of Microsoft Word so the user can use the Z tool and Word simultaneously. Figure 2-5 is a diagram of the prototype toolbar. The toolbar has an “Insert Schema” button which places an empty schema at the mouse position. The toolbar also has a Z/EVES tabpane which contains buttons to run most of the simple Z/EVES commands. The commands include converting schemas to the LaTeX format, printing schemas and performing proofs.

The tool also has a button which displays an interactive Z/EVES form. Expert users can type specific Z/EVES commands on this form to perform complex proofs. This allows the software engineer to use the full Z/EVES functionality. Any errors that occur during type checking or theorem proving will appear the “Z/EVES Errors:” text box.

Z/EVES Analytical Engine

The tool must interface with the Z/EVES analytical tool built by ORA. Professor Knight determined that Z/EVES is one of the best Z tools on the market and felt that Z/EVES would be a powerful analytical tool. The Z tool allows both novice and expert Z users to create detailed and exact Z specifications.

The tool allows novice users to incrementally develop schemas and type-check them. Novice users can use the Z/EVES tabpane buttons to invoke Z/EVES commands behind the scenes. If an error occurs, the Z/EVES error message will be posted in the error textbox on the toolbar. Any Z/EVES output such as diagnostics or theorem results, will appear in a separate Word document within Microsoft Word. As stated in the previous section, expert users can use the “Interactive Z/EVES” window to perform complex tasks.

After the basic requirements for the tool were known, a specific architecture for the tool was made. Figure 2-6 shows the specific layout. The control software is written in Visual Basic. Microsoft Word is the modern word processor for the system. Finally, Z/EVES provides all of the Z type checking and theorem proving. The Control Panel uses OLE Automation to communicate to Word. The Control Panel also uses Dynamic Data Exchange (DDE) to communicate to Z/EVES.

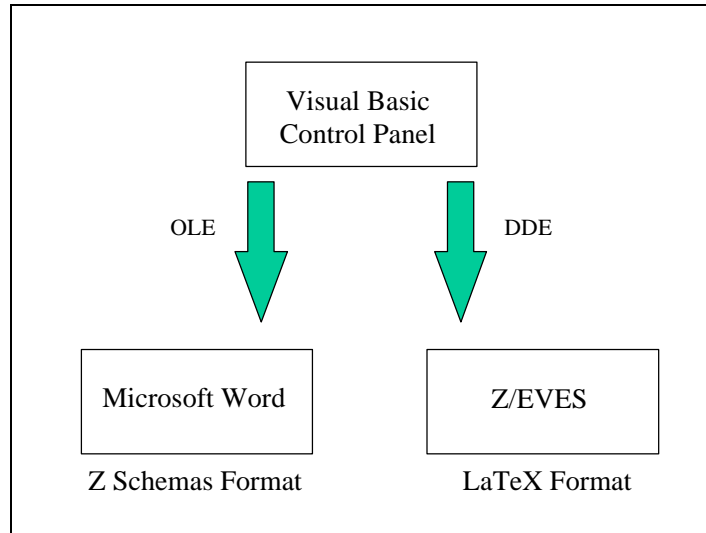


Figure 0-6: Specific Z Tool Architecture

2. Chapter Two - Z Tool Requirements12

2.1. Modern Word Processor12
 2.2. WYSIWYG Development Environment12
 2.3. Usable Interface.....15
 2.4. Z/EVES Analytical Engine.....16

Figure 2-1: Overall Z Tool Architecture.....12
 Figure 2-2: Z/EVES LaTeX Interface13
 Figure 2-3: WYSIWYG Z Tool Interface14
 Figure 2-4: Z Boxes.....15
 Figure 2-5: Toolbar16
 Figure 2-6: Specific Z Tool Architecture.....17

Chapter Two - Z Tool Requirements

Professor Knight developed most of the software requirements for the tool. He developed the idea of building a easy-to-use Z specification tool that could run under Windows. The requirements were modified as different prototypes were built but the overall structure of the system has remained the same. Figure 2-1 is a diagram of the overall system architecture. The Z tool is a control system which runs and manipulates a modern word processor and a Z analytical engine.

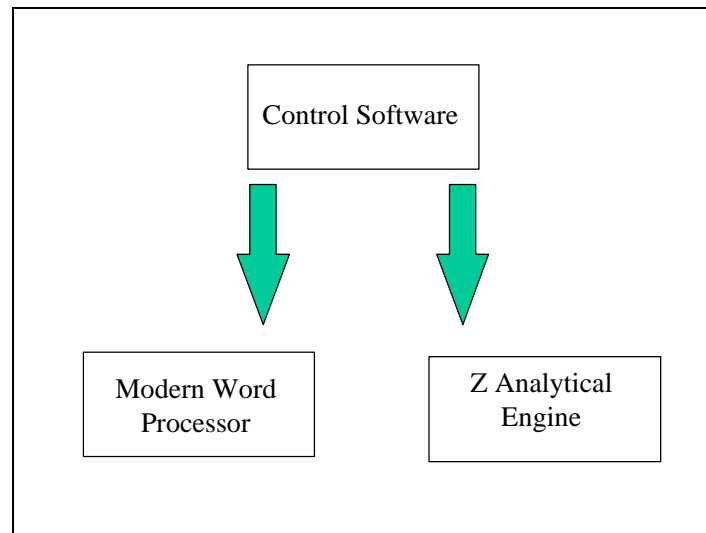


Figure 0-1: Overall Z Tool Architecture

Modern Word Processor

This first requirement of the system is that it provides modern word processing power to the user. The user must be able to develop a professional specification document. Microsoft Word provides powerful processing power and a programmable interface through Visual Basic.

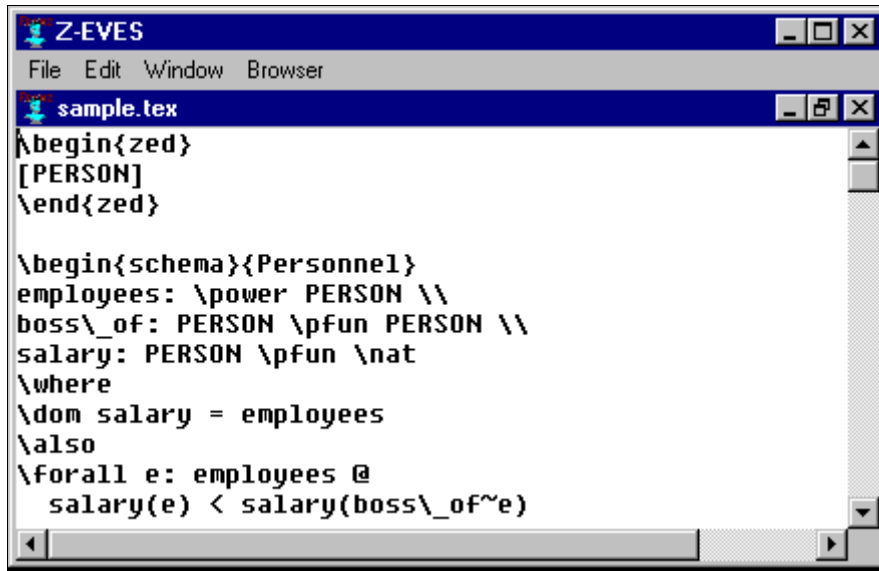
Microsoft Word allows the user to build large specification documents. Software engineers can use Word to produce documents that contain headers, footers, page numbers, automatic outlining and other professional features. Also, Microsoft Word is compatible with Microsoft Excel spreadsheets and Microsoft Access databases. Users also have modern file capabilities when using Word. Word provides the user with standard file editing capabilities, such as editing, saving, copying and deleting documents. The user can also integrate multiple Word documents into one document. Microsoft Word has full printing capabilities including a print-preview functionality.

Microsoft Word has a full Object-Linking and Embedding (OLE) Automation interface. This allowed the development team to manipulate Microsoft Word through the control software. By using Word, we employed large-scale component software reuse. The development team could focus efforts on addressing the major risks of the Z tool instead of trying to build a modern word processor that is compatible with the Z tool.

WYSIWYG Development Environment

The Z tool must provide a What You See Is What You Get (WYSIWYG) development environment. This means that when the user prints the document from the word processor, the printed document will look exactly like the document on the screen. As stated in Section 1.2.1, the existing Z analytical tools only provide the awkward LaTeX interface.

Figure 2-2 shows the LaTeX interface for Z/EVES.



```
Z-EVES
File Edit Window Browser
sample.tex
\begin{zed}
[PERSON]
\end{zed}

\begin{schema}{Personnel}
employees: \power PERSON \
boss\_of: PERSON \pfun PERSON \
salary: PERSON \pfun \nat
\where
\dom salary = employees
\also
\forall e: employees @
  salary(e) < salary(boss\_of~e)
```

Figure 0-2: Z/EVES LaTeX Interface

As shown in the figure, the LaTeX style does not use the standard Z characters and schemas but uses DOS-based characters to represent the Z symbols. For example, the LaTeX format uses the strings `\power` and `\pfun` to represent the Z characters \cdot and ϕ .

Figure 2-3 shows a WYSIWYG interface for the Z tool. The WYSIWYG interface uses the standard Z font and characters. The figure shows a standard schema including the vertical and horizontal lines for the schema boundaries.

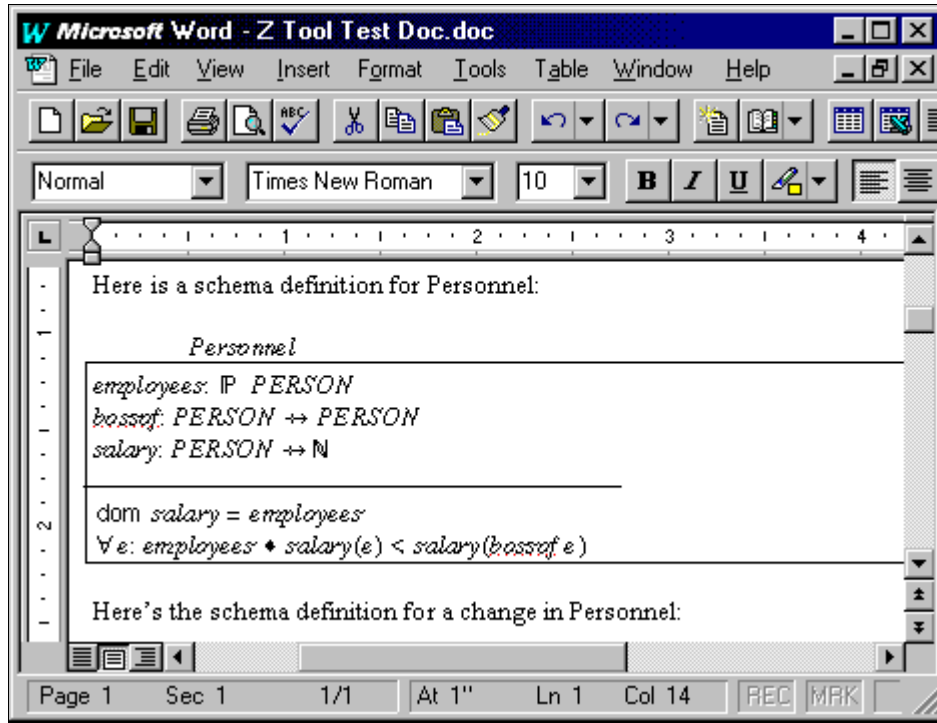


Figure 0-3: WYSIWYG Z Tool Interface

The Z tool must have a button on the interface to insert empty Z schemas into the document. The user can enter text into the schema and the schema must expand as more lines are added.

The WYSIWYG development environment needs to be able to handle the three main types of Z boxes: axiomatic, schema and generic. Figure 2-4 is a diagram of each Z box.

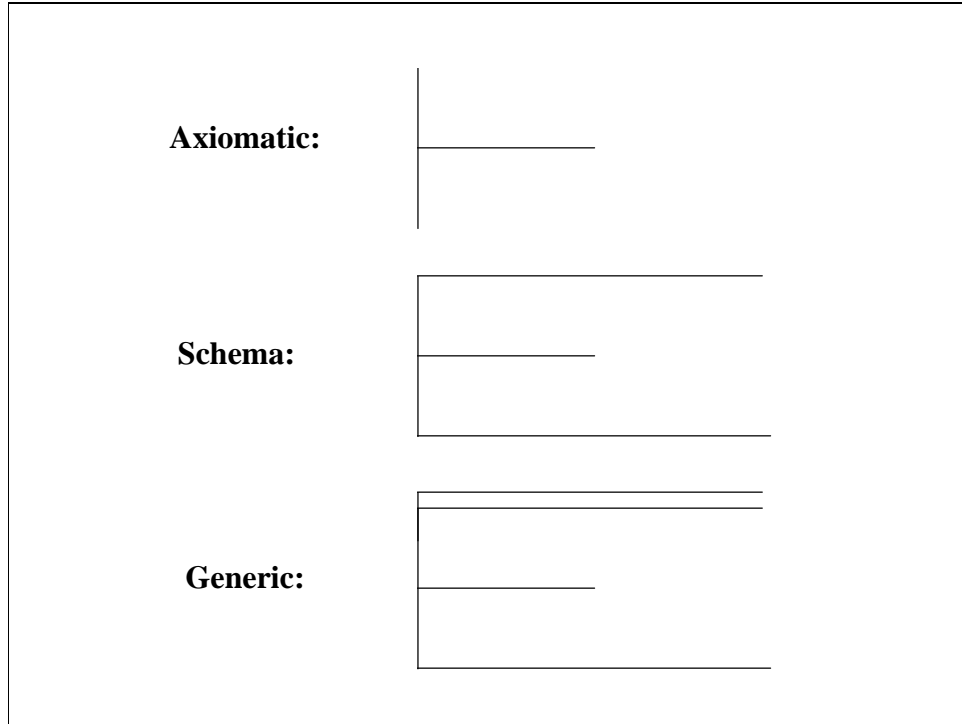


Figure 0-4: Z Boxes

Usable Interface

The Z tool must also have a usable graphical user interface (GUI). The current Z tools use a command line editor to run the tool. The user must be familiar with all of the tool's commands in order to run the tool. A GUI for the Z tool will allow novice users to simply click on buttons to add schemas to the document or to type check the schemas. The tool must also allow expert users to enter Z/EVES command strings to perform complex theorem proving.

The GUI can be developed in Visual Basic (VB). Developers can quickly develop complex GUIs using Visual Basic. Developers can also write code to perform OLE Automation which can control Microsoft Word.

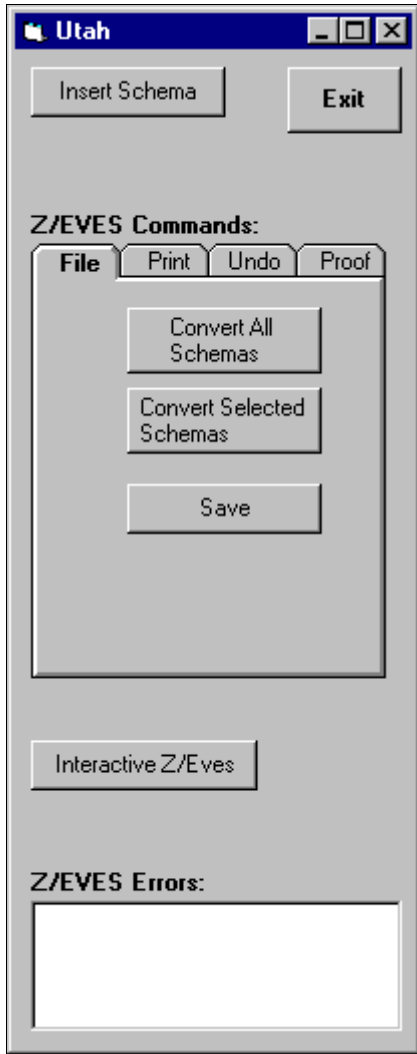


Figure 0-5: Toolbar

The tool should provide a “floating” toolbar which contains the necessary buttons to run Z/EVES. The toolbar will “float”, or stay on top, of Microsoft Word so the user can use the Z tool and Word simultaneously. Figure 2-5 is a diagram of the prototype toolbar. The toolbar has an “Insert Schema” button which places an empty schema at the mouse position. The toolbar also has a Z/EVES tabpane which contains buttons to run most of the simple Z/EVES commands. The commands include converting schemas to the LaTeX format, printing schemas and performing proofs.

The tool also has a button which displays an interactive Z/EVES form. Expert users can type specific Z/EVES commands on this form to perform complex proofs. This allows the software engineer to use the full Z/EVES functionality. Any errors that occur during type checking or theorem proving will appear the “Z/EVES Errors:” text box.

Z/EVES Analytical Engine

The tool must interface with the Z/EVES analytical tool built by ORA. Professor Knight determined that Z/EVES is one of the best Z tools on the market and felt that Z/EVES would be a powerful analytical tool. The Z tool allows both novice and expert Z users to create detailed and exact Z specifications.

The tool allows novice users to incrementally develop schemas and type-check them. Novice users can use the Z/EVES tabpane buttons to invoke Z/EVES commands behind the scenes. If an error occurs, the Z/EVES error message will be posted in the error textbox on the toolbar. Any Z/EVES output such as diagnostics or theorem results, will appear in a separate Word document within Microsoft Word. As stated in the previous section, expert users can use the “Interactive Z/EVES” window to perform complex tasks.

After the basic requirements for the tool were known, a specific architecture for the tool was made. Figure 2-6 shows the specific layout. The control software is written in Visual Basic. Microsoft Word is the modern word processor for the system. Finally, Z/EVES provides all of the Z type checking and theorem proving. The Control Panel uses OLE Automation to communicate to Word. The Control Panel also uses Dynamic Data Exchange (DDE) to communicate to Z/EVES.

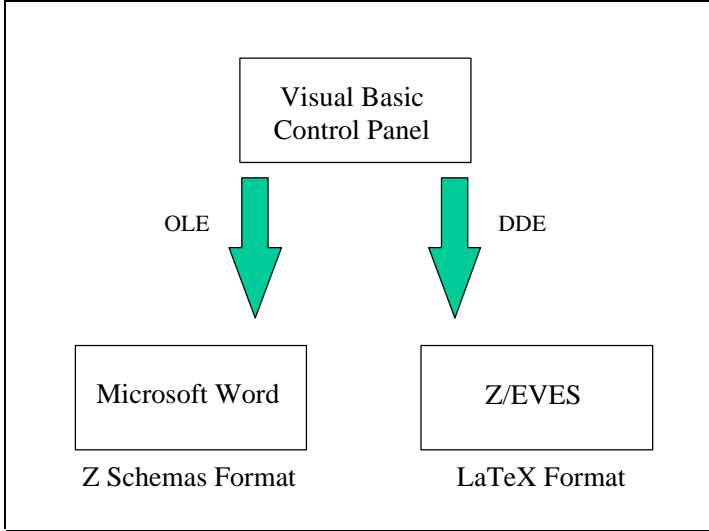


Figure 0-6: Specific Z Tool Architecture

3. Prototyping Methodology.....24

3.1. Determine Feasibility.....24

3.2. Rapid Application Development.....24

3.3. Modules24

3.4. Plan For Prototypes24

Prototyping Methodology

Determine Feasibility

As stated in Section 1.4, the rationale and scope of the project is to determine if it is feasible to build a Windows Z Specification tool using Visual Basic, Microsoft Word and Z/EVES. Once it is determined that it is feasible to build such a tool, another group of researchers and students will build the final product. A prototyping methodology was used to determine the feasibility of building the tool. The methodology was based on building simple prototypes that addressed the major risk areas of the software.

Most of the prototypes were small throw-away prototypes. The throw-away prototypes were built quickly and the developers didn't worry about code readability and efficiency. The sole purpose of each throw-away prototype was to address a single risk area and answer a specific feasibility question.

Rapid Application Development

The rapid application development (RAD) methodology was also used. Rapid application development is the quick development of program code and GUIs. This quick development allows the software developer to get instant feed back from the users about the GUI and code. Using Visual Basic, I could quickly design and build forms and conversion code for the tool. The software was immediately reviewed by Professor Knight and he gave me feedback on the functionality of the code and more specific requirements for the tool. I did not worry about the efficiency or readability of the code because that was not vital to the purpose of the project. Another facet of RAD is software reuse. Large-scale component software reuse was employed in this project by using Microsoft Word and Z/EVES.

Modules

The three main modules for the prototypes were the Visual Basic Control Panel, Microsoft Word, and Z/EVES. Most of the early prototypes dealt with the Microsoft Word module. The later prototypes dealt with the communication between all three modules.

Plan For Prototypes

The initial prototypes dealt with drawing the schemas in Word and extracting and converting the schemas to the LaTeX format needed by Z/EVES. If any of the simple schema prototypes could not be built, Microsoft Word would not be able to be used as the word processor. Once it was determined that Microsoft Word could handle the WYSIWYG schema development, the second major round of prototyping began. The next set of prototypes concentrated on the communication between the Visual Basic Control Panel and Z/EVES. ORA built various Z/EVES stubs to test the communication links.

4. Initial Prototypes	26
4.1. Risks Addressed	26
4.2. Use of OLE.....	26
4.3. Z Font Characteristics	26
4.4. Locate Z Text	26
4.5. Draw Schemas.....	27
4.6. Extract Z Schemas.....	29
4.7. Convert Schemas To LaTeX Format.....	30
Figure 4-1: Document With Text and Schemas	27
Figure 4-2: Schema Drawn Using One Style	28

Figure 4-3: Schema Drawn Using Two Styles.....	28
Figure 4-4: Schema Extraction and Conversion Process.....	30

Initial Prototypes

Risks Addressed

The initial set of prototypes addressed risks that involved drawing and extracting schemas from Word documents. The following is a list of risks that were addressed by the initial prototypes:

- How is the Z font stored in Word?
- How can a program locate Z text within a Word document?
- How can Z schemas be extracted from a Word document?
- Can the Z schemas be converted to the LaTeX format?
- Can all the lines for a Z schema be drawn in Word and can the schema expand as new text is added?

Use of OLE

Before any of the prototypes could be built, a thorough understanding of Word's OLE interface was needed. The Word OLE interface allows Visual Basic programmers to control Word through Visual Basic. Visual Basic code can be written to control almost every aspect of Word, including creating, manipulating, searching, and saving documents. Microsoft's *Word Developer's Kit* is an excellent reference manual for all of the OLE function calls for Word. It includes examples of function calls which were a valuable resource when creating the initial prototypes.

Z Font Characteristics

The first risk to address was to see if Word could handle the Z font and determine how it stored the font in the document. The Z font is a public font that contains all of the Z characters and symbols such as , , , and £. Professor Knight supplied the development team with the Z font that he had been using for specifications.

I was able to install the Z font through the Adobe Type Manager software. The Adobe Type Manager is a standard way to install and register new fonts in Windows. I also wrote a simple macro in Word that returned that ASCII value for the selected character. I tested the macro on several of the Z characters and found out that the characters had distinct ASCII values. This showed that the Z font was mapped to the ASCII character set. This mapping could be used to convert the schemas into the LaTeX format.

Locate Z Text

Another major risk to address was to determine how to locate Z text and Z schemas in a specification. A specification could have normal written text with Z schemas dispersed throughout the document. Therefore, the Z tool needed to be able to search a Word document and find the Z text.

Figure 4.1 shows a specification that has both normal written text and Z schemas and set definitions.

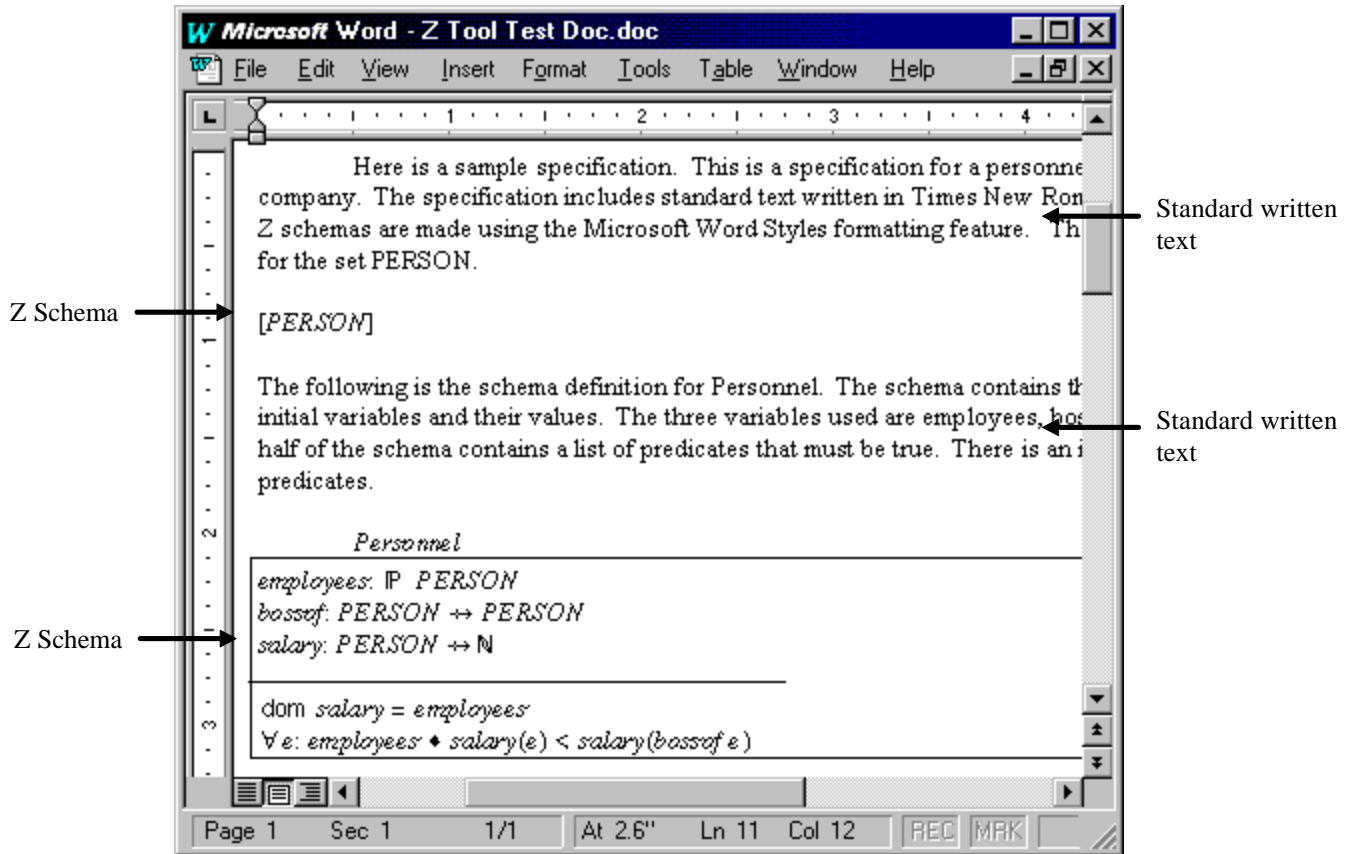


Figure 0-1: Document With Text and Schemas

One method of finding the Z text was to use the EditFindFont Word function. This function moves the cursor to the next location of a specified font. I wrote another small macro that would move the cursor to the position of the next occurrence of the Z font. This macro worked but was too basic. The schemas in the document would not just be simple Z text but would be objects with lines and other formatting details. Also, there are different types of Z boxes and the tool must be able to extract each type. Figure 2-4 is a diagram of the various types of Z boxes that the Z tool must handle.

Draw Schemas

As stated in the previous section, the schemas involve more than just text in the Z font. The schemas need to have specific horizontal lines and vertical lines that expand as more text is typed. I tried various methods to draw the lines. The first method was to automatically draw the necessary lines for the schema boxes. However, the schema could not expand as more text was added. I also tried using the rectangle drawing objects to draw the schema lines. However, the rectangle contained unnecessary lines and couldn't expand.

I also tried using the styles that come with Microsoft Word to draw the schemas. A style is a specific combination of character and paragraph formats in Word. When using styles, line borders can be added around paragraphs that expand as more text is added. The user can also specify which lines around a paragraph can be shown. I built a Z Schema Style that contained the outside lines of the schema box and I manually added a line across the middle. Figure 4-2 is a diagram of a schema box drawn using one style and one line drawn manually. As new text is entered, the middle and bottom horizontal lines are shifted down and the vertical line extends downward.

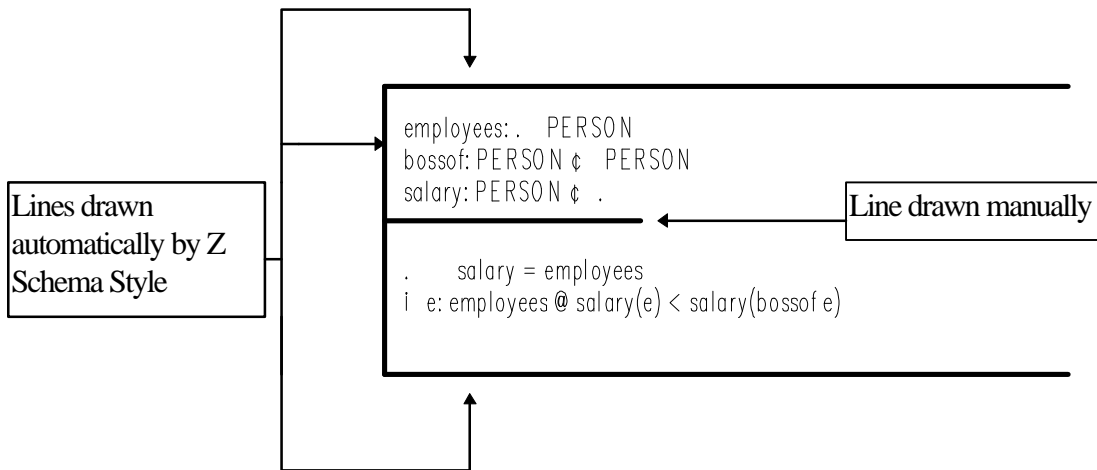


Figure 0-2: Schema Drawn Using One Style

However, this solution still had a problem. When extracting the schemas and converting them to the LaTeX format, the process needs to be able to differentiate between the upper signature half and the bottom predicate half of the schema. I tried building a schema using two styles: Z - Schema Signature and Z - Schema Predicate. As shown in Figure 4-3, the top half of the schema box is drawn by the Z - Schema Signature style and the second half is drawn by the Z - Schema Predicate style. The line in the middle is still drawn manually. With this new method, the conversion routine can differentiate between the upper and lower halves of the schema by checking the current style. This is discussed further in Section 4.7.

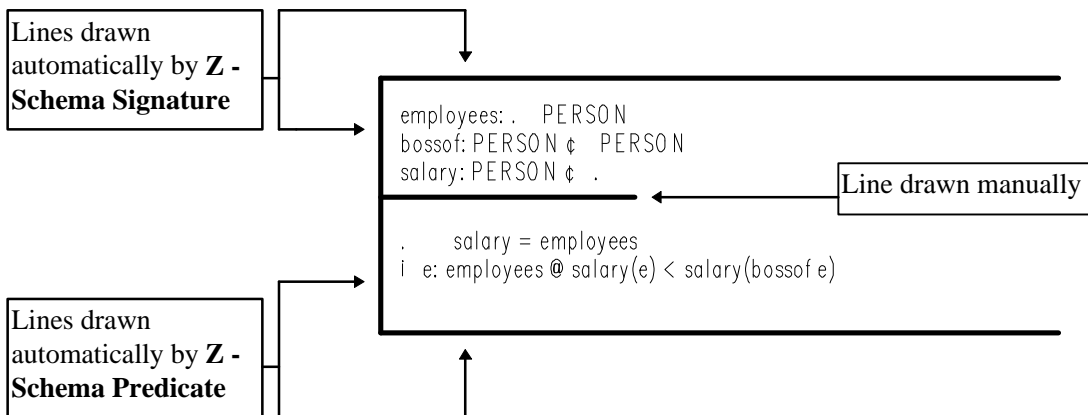


Figure 0-3: Schema Drawn Using Two Styles

Table 4-1 lists the styles that were added to Word to allow the schemas to be drawn. Each signature style must be followed by the predicate style of the same type.

Z - Set Definition
Z - Schema Signature
Z - Schema Predicate
Z - Axiomatic Signature
Z - Axiomatic Predicate
Z - Generic Signature
Z - Generic Predicate

Table 4-1: List of Schema Styles

Extract Z Schemas

Once it was determined that the schemas could be drawn correctly, the next step was to write the code to extract the schemas from the document. At the beginning of the project, the development team thought that it would be enough just to extract all of the schemas and convert them at once. However, after a conference call with ORA, the development team decided that the system should be able to extract all of the schemas or any schemas that were selected (highlighted in Word). The toolbar pictured in Figure 2-5 shows a “Convert All Schemas” button and a “Convert Selected Schemas” button. I designed a conversion process which could handle both of these conversions.

I built a Visual Basic class called `Converter`. This class performed all of the schema extraction and conversion functionality. The class contained the following member functions:

Function Name	Function Description
<code>ConvertSchemas</code>	Converts any extracted schemas to the LaTeX format by calling the appropriate <code>FormatZ</code> functions listed below
<code>ConvertString</code>	Converts a given string to the LaTeX format using the Z font mapping
<code>ExtractAllSchemas</code>	Extracts all schemas from the given specification document
<code>ExtractSchemas</code>	Actually extracts the schemas, called by <code>ExtractAllSchemas</code> and <code>ExtractSelectedSchemas</code>
<code>ExtractSelectedSchemas</code>	Extracts the highlighted schemas
<code>FormatZSchemaPredicate</code>	Converts the schema predicate by adding the necessary LaTeX keywords and converting each line
<code>FormatZSchemaSignature</code>	Converts the schema signature by adding the necessary LaTeX keywords and converting each line
<code>FormatZSetDefinition</code>	Converts the Z set definitions to the LaTeX format
<code>Setup</code>	Must be called before all other functions to initialize the class

Table 4-2: List of Converter Member Functions

As shown in Figure 4-4, the extraction of the schemas depends on whether the user tried to extract the highlighted schemas or all of the schemas. In the figure, each box is a temporary Word document used to store schema information. The name inside the box is the name of the variable in the `Converter` class that stores the document name. For example, the value of `is_DocumentDoc` will be the name of the specification document, such as “C:\SPECIFICATION.DOC”. The value of `is_ExtractedSchemasDoc` will be the name that Word assigns to the newly created temporary document, such as “Document 4”.

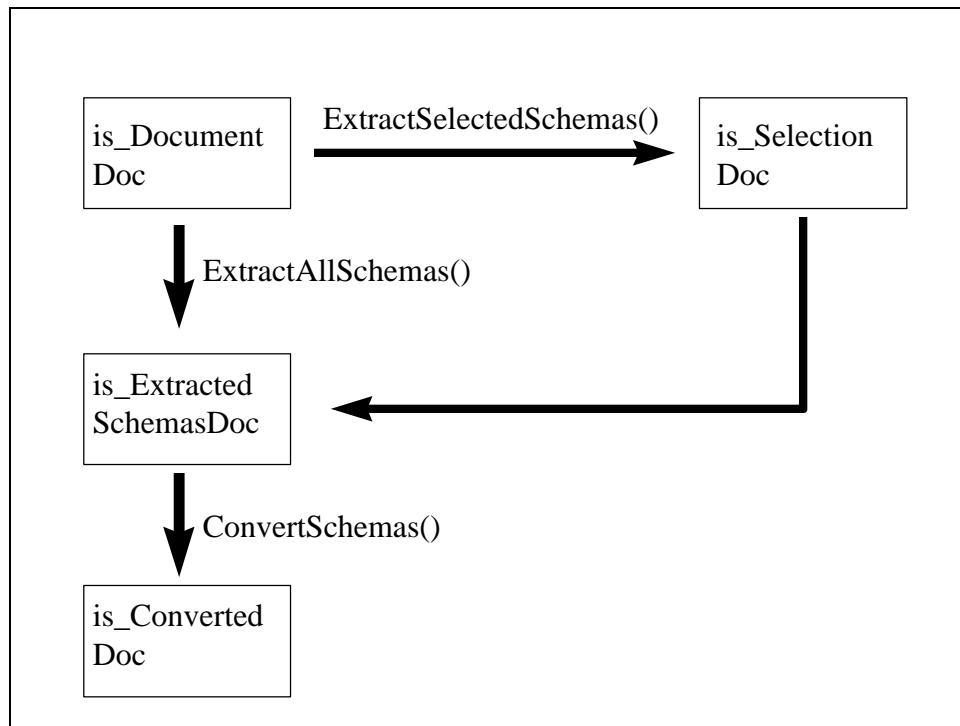


Figure 0-4: Schema Extraction and Conversion Process

If the user clicks on the “Convert All Schemas” button, the `ExtractAllSchemas()` function loops through the specification line by line and processes each schema encountered. The `ExtractAllSchemas()` function calls the `ExtractSchemas()` function on the specification document (`is_DocumentDoc`). The function knows a schema has been found when the style of the current line is one of the following:

- Z - Schema Signature
- Z - Axiomatic Signature
- Z - Generic Signature

When the function encounters the beginning of a schema, it copies the schema to the extracted schemas document. The `is_ExtractedSchemasDoc` document contains all of the extracted schemas without any of the regular text. For each style in a schema, a header and footer line is added to `is_ExtractedSchemasDoc`. This step is necessary in the conversion process described in the next section.

The extraction process for selected schemas is the same except that the highlighted schemas must be isolated first. The selected schemas are copied to the `is_SelectionDoc` document. The exact same extraction code, `ExtractSchemas()`, is then run on the `is_SelectionDoc` instead of `is_DocumentDoc`. After the extraction process, `is_ExtractedSchemasDoc` contains the isolated schemas with the appropriate header and footer lines.

Convert Schemas To LaTeX Format

As shown in Figure 4-4, the conversion process takes the schemas from `is_ExtractedSchemasDoc` and converts them to the LaTeX version in `is_ConvertedDoc`. The `ConvertSchemas()` function executes this conversion process. The `ConvertSchemas()` function loops through each line in `is_ExtractedSchemasDoc` and checks to see if a header for a schema is found. The function checks the type of schema and formats the code appropriately. For the prototype, only the conversion for the Z - Set Definition, Z - Schema Signature, Z - Schema Predicate

styles were coded. The conversion code for the other styles listed in Table 4-1 should be very similar to the existing code.

`ConvertSchemas()` calls `FormatZSetDefinition()`, `FormatZSchemaSignature()`, and `FormatZSchemaPredicate()` depending on what schema style was found. Each formatting function performs a different conversion routine. The `FormatZSetDefinition()` function just adds a specific header and footer line to the set name. The set definition PERSON would be converted as follows:

```
[PERSON]  →  \begin{zed}
             [PERSON]
             \end{zed}
```

The `FormatZSchemaSignature()` function converts each Z character to the corresponding LaTeX character sequence and also adds the characters “\” after each declaration. The `ConvertString()` function changes each Z character in a string to the corresponding LaTeX code. The function uses the `zfont1.dat` file to determine the LaTeX codes for each ASCII character. The schema signature for Personnel would be converted as follows:

<pre>Personnel employees: . PERSON bossof: PERSON ↯ PERSON salary: PERSON ↯.</pre>	<pre>→ \begin{schema}{Personnel} employees: \power PERSON \ bossof: PERSON \pfun PERSON \ salary: PERSON \pfun \nat</pre>
--	---

The `FormatZSchemaPredicate()` function also converts each Z character to the corresponding LaTeX character sequence but then adds “also” after each predicate statement. The schema predicate for Personnel would be converted as follows:

<pre>. salary = employees ie: employees @ salary(e) < salary(bossof e)</pre>	<pre>→ \where \dom salary = employees \also \forall e: employees @ salary(e) < salary(bossof~e) \end{schema}</pre>
---	---

When `ConvertSchemas()` is finished, `is_ConvertedDoc` is saved as a standard text file that can be read by Z/EVES. The “read” command can be sent to Z/EVES to read the text file and enter the schemas into the Z/EVES system.

5. Communication Prototypes32

- 5.1. Risks Addressed32
- 5.2. Control Panel32
- 5.3. Visual Basic Z/EVES Stub32
- 5.4. First Z/EVES Stub.....33
- 5.5. Second Z/EVES Stub.....34
- 5.6. Z/EVES With DDE Interface.....34

Figure 5-1: Visual Basic Z/EVES Stub33

Figure 5-2: Z/EVES With DDE Interface34

Communication Prototypes

Risks Addressed

The major risks addressed with the communication prototypes were the risks of how the Visual Basic Control Panel was going to communicate with Microsoft Word and with Z/EVES. The following is a specific list of risks addressed by the communication prototypes:

- How will the Visual Basic Control Panel send commands to Word?
- How will the Visual Basic Control Panel send commands to Z/EVES?
- How will the Visual Basic Control Panel receive error messages and structured output from Z/EVES?

Control Panel

The initial prototypes answered the question of how the Control Panel will send commands to Word. The Control Panel used OLE automation to manipulate word. The following is a listing of OLE automation code that creates two new Word documents named "Output.doc" and "Project Utah.doc".

```
Set WordObj = CreateObject("Word.Basic")
```

```
'set up output window  
WordObj.FileNewDefault Document.dot", NewTemplate:=0  
WordObj.Activate "Document1"  
WordObj.FileSaveAs Name:="Output.doc"
```

```
' Set up Writing Window  
WordObj.FileNew  
    Template:="Z:\ms\msoffice\Templates\ZedDocument.dot",  
    NewTemplate:=0  
WordObj.Activate "Document2"  
WordObj.FileSaveAs Name:="Project Utah.doc"
```

Other similar commands were used to perform all of the Microsoft Word tasks through Visual Basic.

The Control Panel communicates with Z/EVES through Dynamic Data Exchange (DDE). Dynamic Data Exchange is an established protocol for exchanging data through active links between applications that run under Microsoft Windows (Martinsen, 266). The development team held conference calls with ORA to setup a DDE interface for Z/EVES. Irwin Meisels at ORA wrote the DDE code for the simple Z/EVES stubs and for the Z/EVES version with the DDE interface. Irwin added certain DDE items to the Z/EVES programs which made information visible to the Visual Basic Control Panel.

Visual Basic Z/EVES Stub

Before ORA built the first Z/EVES stub, I built a Visual Basic stub that would simulate Z/EVES. This stub allowed us to continue to work with the Control Panel even though the ORA stub was not built. The stub posted simulated output and error messages to any client applications that were linked to it. Figure 5-1 is a picture of the Visual Basic Z/EVES stub.

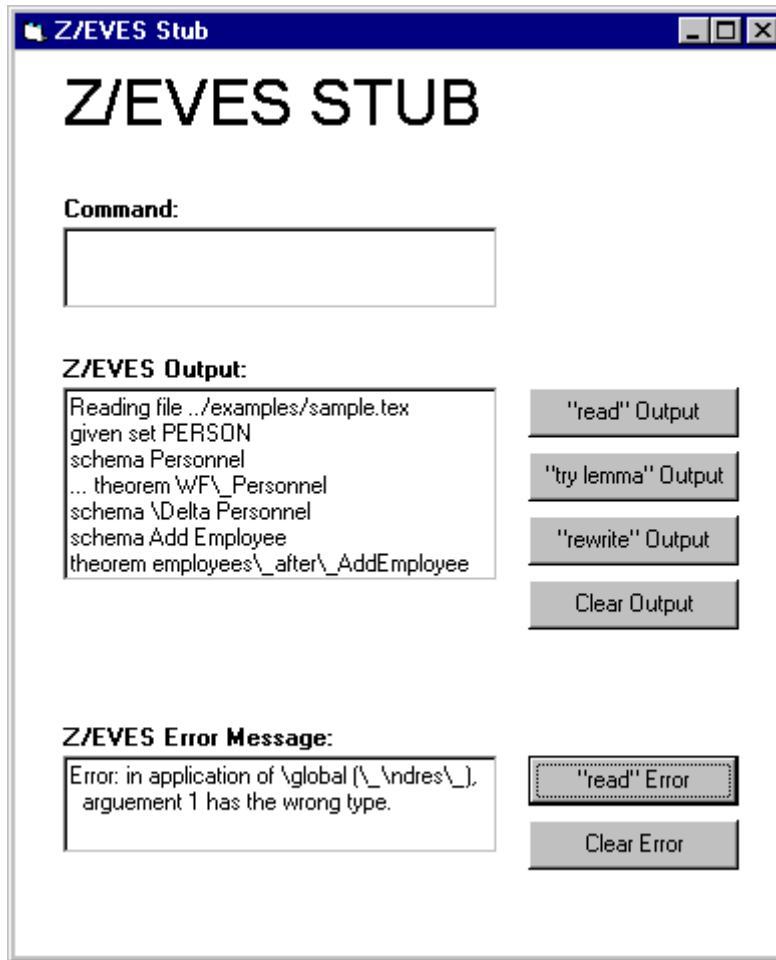


Figure 0-1: Visual Basic Z/EVES Stub

Table 5-1 is a list of the DDE items of the stub.

DDE Link Topic	DDE Link Item	Description
SYSTEM	CONTROL	This item contained the command that the stub was to “execute”. Client applications can write a command string to this item.
SYSTEM	OUTPUT	This item contains the simulated output that Z/EVES would create when certain command strings are received. Client applications can look at this value.
SYSTEM	ERROR	This item contains the simulated errors that Z/EVES would create when certain command strings are received. Client applications can look at this value.

Table 5-1: DDE Items For VB Z/EVES Stub

The stub set up three DDE items. Each button on the stub wrote a message to the specific DDE item. The Control link item was setup to receive messages from clients. Client applications could use the Visual Basic function `LinkExecute()` to send data to the stub.

First Z/EVES Stub

The first Z/EVES stub written by ORA tested if simple DDE links could be established between Z/EVES and Visual Basic. This stub was written by Irwin Meisels at ORA. The stub does not actually

execute any Z/EVES commands but it just returns message strings. The list of the DDE items in the stub are listed in Table 5-2.

DDE Link Topic	DDE Link Item	Description
SYSTEM	SYSITEMS	Returns a list of all of the DDE link items that the stub supports ("TOPICS")
SYSTEM	TOPICS	Returns a list of the service topics ("SYSTEM", "EVAL")
EVAL	-	Clients can write a string to this topic and the stub will "execute" it
EVAL-RESULT	EVAL-RESULT	Returns the string passed into the EVAL topic

Table 5-2: DDE Items For the First Z/EVES Stub

The SYSTEM|SYSITEMS and SYSTEM|TOPICS links worked fine with Visual Basic. However, the EVAL topic would not work. The stub was set up so that client applications (like Visual Basic) had to dynamically set up a EVAL link item. This was causing an error in the Z/EVES stub. Since this didn't seem to be the standard protocol for sending DDE information, I suggested that a permanent COMMAND-STRING item be set up. Then, client applications could write to the item using the Visual Basic function `LinkPoke()`. The second Z/EVES stub used this new approach.

Second Z/EVES Stub

The second Z/EVES stub was also written by Irwin. Once again, the stub didn't actually execute any Z/EVES commands. It just returned message strings. Table 5-3 is a list of the DDE items set up by the stub.

DDE Link Topic	DDE Link Item	Description
SYSTEM	SYSITEMS	Returns a list of all of the DDE link items that the stub supports ("TOPICS")
SYSTEM	TOPICS	Returns a list of the service topics ("SYSTEM", "EVAL")
EVAL	COMMAND-STRING	Clients can write a string to this topic and the stub will "execute" it
EVAL-RESULT	EVAL-RESULT	Returns the string passed into COMMAND-STRING

Table 5-3: DDE Items For the Second Z/EVES Stub

All of the links for this stub worked. The Visual Basic program was able to poke a string message to the COMMAND-STRING item and the stub returned the message to the EVAL-RESULT item. This stub proved that it was feasible for the Visual Basic Control Panel to send commands to Z/EVES using DDE.

Z/EVES With DDE Interface

The next prototype written by ORA was a full working version of Z/EVES with a DDE interface. Instead of having the command-line editor interface as shown in Figure 2-2, this version of Z/EVES only had a DDE interface. This version of Z/EVES could only be run through commands sent over DDE links. Therefore, the prototype did not need to have a complex window that had a command menu. I requested that Irwin add a simple window so that I could tell when the prototype was running. The previous two ORA stubs did not have windows and I was not able to tell if the program was actually running.



Figure 0-2: Z/EVES With DDE Interface

Table 5-4 is a list of the DDE links for this version of

Z/EVES. Any strings sent to the COMMAND-STRING item were processed by Z/EVES. For example, if the string "read

‘c:\z-eves\examples\sample.tex’ was posted to COMMAND-STRING, Z/EVES would open the file and process the schemas in the file.

DDE Link Topic	DDE Link Item	Description
EVAL	COMMAND-STRING	Clients can write a Z/EVES command string to this item and Z/EVES will actually execute the string
EVAL	OUTPUT	Z/EVES posts any output resulting from executing the string sent to COMMAND-STRING
EVAL	ERROR	Z/EVES posts any errors resulting from executing the string sent to COMMAND-STRING

Table 5-4: DDE Items For the Second Z/EVES Stub

This version of Z/EVES proved that it is feasible to control Z/EVES with a Visual Basic program. The Control Panel can send commands to Z/EVES and it can also receive output and error messages back from Z/EVES.

6. Conclusion.....36

 6.1. Summary.....36

 6.2. Interpretation.....36

 6.3. Recommendations37

Figure 6-1: Final Z Tool Prototype37

Conclusion

Summary

The prototypes listed in Chapter 4 and Chapter 5 were all attempts to determine if it is feasible to build a Windows Z Specification tool. The tool could have a WYSIWYG interface by using the Z font. All of the specific Z characters and Z symbols can be viewed in Word when the Z font is used. A WYSIWYG environment is also achieved by drawing Z schemas with styles. Schemas built by using styles have automatically drawn lines and also expand as more text is entered.

Through the use of OLE automation, a Visual Basic Control Panel can control Microsoft Word. The Control Panel can search for Z schemas by using OLE functions to loop through the document and check the style of each line. The Control Panel can also extract the schemas out of the specification document using OLE automation and temporary Word documents. Once the schemas have been extracted, the `Converter` class can convert the schemas to the LaTeX format which can then be read by Z/EVES.

Also, it was proven that a Visual Basic program can control Z/EVES. The prototypes listed in Chapter 5 demonstrate how information can be passed between two Windows programs using DDE. Finally, it was proven that the Control Panel could control Z/EVES. The last version of Z/EVES accepted standard Z/EVES commands from the Visual Basic Control Panel and processed the commands.

Interpretation

Based on the facts summarized in Section 6.1, I conclude that it is feasible to build a Windows Z Specification tool. The tool can be built using a Visual Basic Control Panel which controls Microsoft Word and Z/EVES. The two most important parts of the tool are the WYSIWYG specification environment and the interaction with Z/EVES. The WYSIWYG environment is supplied by Microsoft Word. The interaction between Visual Basic and Z/EVES is achieved through DDE. Figure 6-1 is a picture of the final prototype including the three main modules of the system.

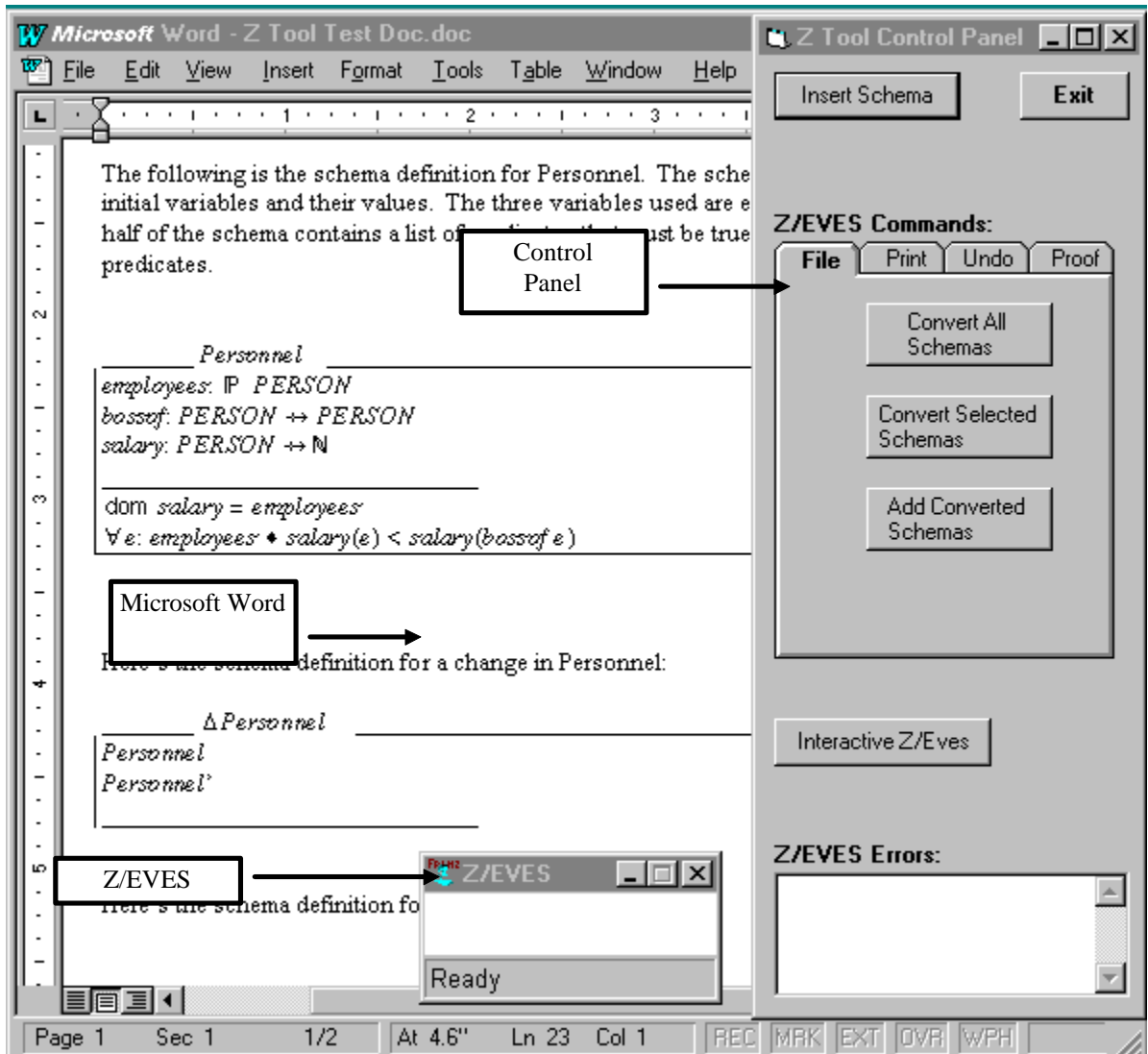


Figure 0-1: Final Z Tool Prototype

Recommendations

The purpose of this research was to determine if building a Windows Z Specification tool is feasible. The next stage of the overall project is to actually build the complete tool. The prototypes built during the project are not completely fit for distribution. The following is a list of possible changes to the tool and possible future work:

- Perform usability tests on the Control Panel and determine if real users can easily build and test specifications using the system.
- Rewrite the conversion code to completely follow the fuzz specifications for LaTeX style Z specifications
- Add the functionality to highlight schema lines with errors directly in the specification document
- Add the functionality to Z/EVES to incrementally develop and type-check schemas

Once the following changes are made, the final Z tool can be packaged and distributed to users.

7. Appendix.....	39
A. Annotated Bibliography.....	39
B. Z Character Set.....	41
C. ZFONT1.DAT Contents	43
D. Control Panel	45
E. Splash Form.....	50
F. Interactive Z/EVES Form.....	51
G. Converter Class	52
H. Module File	65

Appendix

Annotated Bibliography

Z Specification

- 1996 Green, S. Sa Jin, A. Vince, and J. Webb. "An investigation into the effect of students' use of the CADiZ tool for checking Z formal specifications." International Conference Software Engineering: Education and Practice, Dunedin, New Zealand, 24 Jan 1996.

This paper describes an experiment in which students were taught Z using the CADiZ tool. It describes the experiment process and shows how the students had an easier time learning Z while using the tool.

Ince, Darrel. [An introduction to discrete mathematics, formal system specification, and Z.](#) New York, NY: Oxford University Press: 1992.

This book provided a good introduction into the basics of formal proofs and Z. It was used for basic research into the Z specification technique.

Turner, J. H. "Issues in teaching Z." Proceedings of Software Engineering in Higher Education SEHE 94, Southampton, UK, 23 Nov 1994.

This paper describes the problems in teaching Z to students. The paper addresses the issues of the students' background and education and their understanding of proofs.

Z FAQ. <http://www.cis.ohio-state.edu/hypertext/faq/usenet/z-faq/faq-doc-1.html>.

This site is the official Frequently Asked Questions section of the comp.specification.z news group. The site contains information on Z and the current tools available.

Z Soup. http://www.scranton.com/~bschnell/zed_soup.html.

This site is maintained by the Formal Specifications Team at the University of Scranton. The set of pages contains information on the history of Z, the specification technique and lists the major tools and publications involving Z. The site also offers an online presentation of the Z specification of the Tower of Hanoi problem.

Current Z Tools

CADiZ. ftp://ftp.cs.york.ac.uk/hise_reports/cadiz

The York Software Engineering ftp site contains documentation on the CADiZ tool for both UNIX and Windows. The Web site also contains some screen shots of the CADiZ tool.

fuzz. <ftp://comlab.ox.ac.uk/pub/Zforum/fuzz>

This WWW site contains information on the fuzz package of tools. The sites contains information on the capabilities of fuzz and also has some free software that can be downloaded.

Z/EVES. <http://www.ora.on.ca/z-eves/welcome.html>

This site contains information on the various versions of Z/EVES. The site describes the capabilities of Z/EVES and describes how the software is still in development.

Meisels, Irwin and Mark Saaltink. The Z/EVES Reference Manual DRAFT. Ottawa, Ontario: ORA Canada, 1995.

This document is the reference manual for the Z/EVES tool. It describes how to use the tool to analyze specifications. It also details the LaTeX format for input.

Z FAQ. <http://www.cis.ohio-state.edu/hypertext/faq/usenet/z-faq/faq-doc-6.html>

This is the “What tools are available” section of the Z FAQ. It contains a comprehensive list of the current tools. The site describes the functionality of each tool and provides links to most of the software.

Application Development

Martinsen, Ronald R., Taylor Maxwell, Mike McKelvy, Michael Regelski, and Jeff Webb. Using Visual Basic 4. Indianapolis, IN: Que Corporation, 1995.

Using Visual Basic 4 is a programmers guide. The book contains two sections that deal with OLE automation and integrating other Microsoft applications with Visual Basic.

Microsoft Corporation. Microsoft Visual Basic Language Reference. Redmond, WA: Microsoft Corporation, 1995.

The *Language Reference* manual is Microsoft’s official Visual Basic programmer’s guide. It likes the standard Microsoft coding conventions and provides a detailed description of each Visual Basic function. Most function descriptions include an example.

Microsoft Press. Microsoft Word Developer’s Kit. Redmond, WA: Microsoft Press, 1995.

The Word Developer’s Kit lists all of the Word Basics functions. The book is a programmers reference manual which groups the methods alphabetically and by function. The book details each function with descriptions of the function, and lists of function arguments and return values. Most function descriptions include an example.

Z Character Set

Alt-Key - ASCII Value	Character	Name
033	!	exclamation
036	\$	subset
037	%	in
038	&	subseteq
064	@	spot
094	^	cat
095	-	neq
0131	.	delta
0132	.	cup
0133	.	integers
0134	.	dagger
0135	.	circ
0136	.	lor
0137	.	dom
0139	.	uplus
0140	.	nat
0145	‘	theta
0147	.	ran
0148	.	seq
0149	.	notin
0150	.	power
0151	.	real
0152	.	cup
0153	.	cross
0155	.	rbag
0156	.	omega
0161	¡	forall
0162	¢	pfun
0163	£	mapsto
0164	¤	lbag
0166	¦	ffun
0167	§	end of text
0168	¨	iff
0169	©	rangle
0170	ª	comp
0174	®	langle
0175	·	xi
0176	*	empty
0180	´	oplus
0182	¶	paragraph
0183	·	lnot
0184	·	filter
0186	≤	leq
0187	»	mu
0189	½	upto
0190	¾	defs

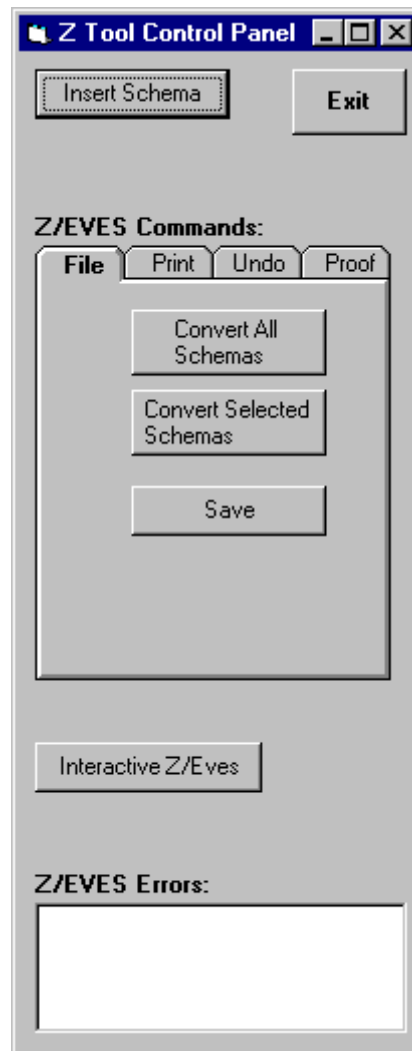
0191	$\dot{\iota}$	rel
0198	\dot{E}	implies
0208	\dot{D}	bigcap
0216	\emptyset	subbag
0221	\dot{Y}	exists
0222	\dot{b}	bigcup
0223	β	fun
0224	\dot{a}	surj
0225	\dot{o}	bij
0226	$\dot{\alpha}$	pinj
0227	$\dot{\alpha}$	psurj
0228	\ddot{a}	inj
0229	$\dot{\alpha}$	finj
0230	\ae	geq
0232	\dot{e}	ndres
0233	\dot{e}	nrres
0234	\hat{e}	rres
0235	\ddot{e}	dres
0247	\div	land
0248	\emptyset	subbag proper
0253	\dot{y}	finset
0254	\dot{b}	lambda

ZFONT1.DAT Contents

36=\subset
37=\in
38=\subseteq
64=@
94=\cat
95=\neq
123=\{
125=\}
131=\delta
132=\cap
133=\integers
134=\dagger
135=\circ
136=\lor
137=\dom
139=\uplus
140=\nat
145=\theta
147=\ran
148=\seq
149=\notin
150=\power
151=\real
152=\cup
153=\cross
155=\rbag
156=\omega
161=\forall
162=\pfun
163=\mapsto
164=\lbag
166=\ffun
167=\end of text
168=\iff
169=\rangle
170=\comp
174=\langle
175=\Xi
176=\empty
180=\oplus
182=\paragraph
183=\inot
184=\filter
186=\leq
187=\mu
189=\upto
190=\defs
191=\rel
198=\implies
208=\bigcap
216=\subbag
221=\exists

222=\bigcup
223=\fun
224=\surj
225=\bij
226=\pinj
227=\psurj
228=\inj
229=\finj
230=\geq
232=\ndres
233=\nrres
234=\rres
235=\dres
247=\land
248=\subbag proper
253=\finset
254=\lambda

Control Panel



Control: **Form1**

Code:

```
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\Zed 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)

ZEVESOutput.LinkMode = vbLinkAutomatic
ZEVESOutput.LinkMode = vbLinkAutomatic
Unload WaitForm
Unload SplashForm

End Sub

Private Sub Form_Click()

  ' Force a DDE request, needed for cold links on Error and Output
  ZEVESOutput.LinkRequest
  ZEVESOutput.LinkRequest
  ZEVESOutput_Change
  ZEVESOutput_Change

End Sub

Private Sub ZEVESOutput_Change()

  If ZEVESOutput.Text = "" Then Exit Sub

  MsgBox "New output received from ZEVES"

  WordObj.Activate "Output.doc"
  WordObj.Insert "*** " & Now
  WordObj.InsertPara

```

```

WordObj.Insert ZEVESOutput.Text
WordObj.InsertPara
WordObj.InsertPara
WordObj.InsertPara

End Sub

Private Sub ZEVESError_Change()

    MsgBox "New Error found!"

End Sub

```

Control: Generic

Insert Schema

Code:

```

Private Sub Generic_Click()

WordObj.Style "Z Text"
WordObj.Font "Times New Roman"
WordObj.Insert "_____"
WordObj.Font "Zed"
''WordObj.Overtyp e 1

WordObj.Insert "          "
WordObj.Font "Times New Roman"
WordObj.Insert
"_____ "
WordObj.Insert "_____"
WordObj.InsertPara
''WordObj.Overtyp e 0

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

WordObj.Style "Z - Schema Predicate"
WordObj.InsertPara
WordObj.InsertPara
WordObj.Style "Z Text"
WordObj.LineUp 1
WordObj.LineDown 1
WordObj.LineUp 4

End Sub

```

Control: Command5

Exit

Code:

```
Private Sub Command5_Click()  
  
    ZEVESCommand.Text = "quit;"  
    ZEVESCommand.LinkMode = vbLinkManual  
    ZEVESCommand.LinkPoke  
  
    End  
  
End Sub
```

Control: Command1

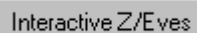
Convert All
Schemas

Code:

```
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
```

Control: Command19**Code:**

```
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
```

Control: Command4**Code:**

```
Private Sub Command4_Click()  
  
Form2.Visible = True  
  
success% = SetWindowPos(Form2.hWnd, HWND_TOPMOST, 0, 0, 0, 0, FLAGS)  
  
End Sub
```

Splash Form

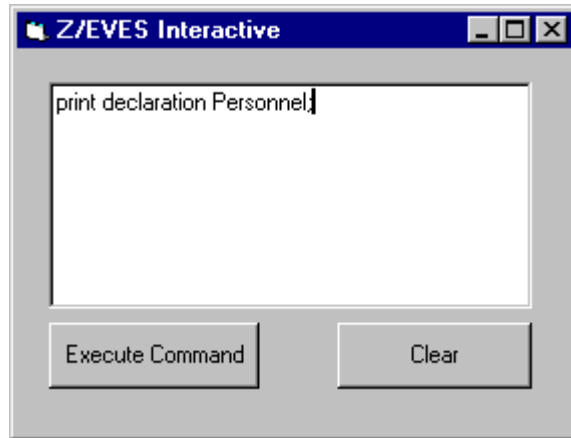


Specification Tool

Project Utah

**Developed by the Software
Engineering Group at the
University of Virginia**

Interactive Z/EVES Form



Control: Command1

Execute Command

Code:

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

Control: Command2

Clear

Code:

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

Converter Class

```
*****
' Converter Class
' Ver 3.0   02/05/97
*****
Option Explicit

' Names of the three Word windows used
Private is_DocumentDoc As String
Private is_ExtractedSchemasDoc As String
Private is_SelectionDoc As String
Private is_ConvertedDoc As String

Private is_DestinationFilename As String
Private is_LookupArray(0 To 256)
Private ii_OutputType As Integer
Private ii_Font As Integer
Private io_Word As Object

' Constants used for determining conversion type
Const CONV_LATEX = 1
Const CONV_Z_FONT_1 = 1
Const CONV_Z_FONT_1_FILENAME = "zfont1.dat"
Const CONV_CARRIGE_RETURN = 13
Const CONV_LINE_FEED = 10
Const CONV_SCHEMA_DEC_END_LINE = " \\"
Const CONV_WORD_TEXT_FILENAME = "new_z.tex"

' Constants for parsing text files
Dim CONV_DOCUMENT_HEADER As String
Dim CONV_DOCUMENT_FOOTER As String

Const CONV_Z_SET_DEFINITION = "Z - Set Definition"
Const CONV_Z_SCHEMA_SIGNATURE = "Z - Schema Signature"
Const CONV_Z_SCHEMA_PREDICATE = "Z - Schema Predicate"
Const CONV_Z_AXIOMATIC_SIGNATURE = "Z - Axiomatic Signature"
Const CONV_Z_AXIOMATIC_PREDICATE = "Z - Axiomatic Predicate"
Const CONV_Z_GENERIC_SIGNATURE = "Z - Generic Signature"
Const CONV_Z_GENERIC_PREDICATE = "Z - Generic Predicate"

Const CONV_HEADER_BEGIN_POSITION = 7
Const CONV_HEADER_BEGIN = "<BEGIN "
Const CONV_HEADER_END = "<END "
```

```

Function ConvertSchemas() As Integer
' *****
' * Function: ConvertSchemas
' *
' * Description: Converts the temp Word document (stored as a text file)
into the correct Z format
' *
' * Input Parameters: none
' *
' * Output Parameters: none
' *
' * Return Value: integer - 0 = Error occurred
' *                  1 = No errors
' *
' * Modification Log:
' * Date      Developer      Change
' * 10/14/96   S.Ziegler      Initial Development
' * 11/07/96   S.Ziegler      Changed name of functions to
Convert...Signature
' *
' * 02/05/97   S.Ziegler      and Convert...Predicate
Changed name of function to
ConvertSchemas
' *
' *                  and conversion occurs on the Windows
side now
' *
' *****

    Dim lc_CurrentChar
    Dim li_AsciiValue As Integer
    Dim li_Counter As Integer
    Dim li_TypeOfText As Integer
    Dim li_CharIndex As Integer
    Dim li_SchemaPortion As Integer
    Dim ls_Buffer As String
    Dim ls_Write As String
    Dim ls_StyleName As String

    io_Word.ScreenUpdating 1
    io_Word.FileNew
    is_ConvertedDoc = io_Word.WindowName$()
    io_Word.Activate (is_ExtractedSchemasDoc)
    io_Word.StartOfDocument

    ' Parse input file and do conversions
    Do While Not io_Word.AtEndOfDocument()
        ls_Buffer = GetCurrentLine()

        If Left(ls_Buffer, CONV_HEADER_BEGIN_POSITION) =
CONV_HEADER_BEGIN Then
            ls_StyleName = Mid(ls_Buffer, 8, Len(ls_Buffer))
            ls_StyleName = Left(ls_StyleName, Len(ls_StyleName) - 1) '
remove ">"

            Select Case ls_StyleName

                Case CONV_Z_SET_DEFINITION

```

```

        If FormatZSetDefinition() = 0 Then
            MsgBox ("Error")
            ConvertSchemas = 0
            Exit Function
        End If

        Case CONV_Z_SCHEMA_SIGNATURE

            If FormatZSchemaSignature() = 0 Then
                MsgBox ("Error")
                ConvertSchemas = 0
                Exit Function
            End If

        Case CONV_Z_SCHEMA_PREDICATE

            If FormatZSchemaPredicate() = 0 Then
                MsgBox ("Error")
                ConvertSchemas = 0
                Exit Function
            End If

    End Select
End If
Loop

io_Word.Activate (is_ExtractedSchemasDoc)
io_Word.FileClose 2
io_Word.Activate (is_ConvertedDoc)
'io_Word.FileSaveAs Name:=is_DestinationFilename, Format:=2
io_Word.FileSaveAs Name:="c:\z-eves\z_latex.tex", Format:=2

io_Word.ScreenUpdating 1

End Function

Function ConvertString(as_Line As String) As String
' *****
' * Function: ConvertString
' *
' * Description: Converts the given string to change Z symbols to the
correct codes
' *
' * Input Parameters:
' *   as_Line as String           -   String to be converted
' *
' * Output Parameters: none
' *
' * Return Value: integer - ""           = Error occurred
' *                   non-empty string   = No errors
' *
' * Modification Log:
' *   Date       Developer       Change
' * 10/14/96    S.Ziegler       Initial Development
' *
' *
' *****

```

```

Dim ls_NewString As String
Dim ls_CurrentChar As String
Dim li_AsciiValue As Integer
Dim i As Integer

For i = 1 To Len(as_Line)
    ls_CurrentChar = Mid(as_Line, i, 1)
    li_AsciiValue = Asc(ls_CurrentChar)
    If is_LookupArray(li_AsciiValue) <> "" Then
        ls_NewString = ls_NewString & is_LookupArray(li_AsciiValue)
    Else
        ls_NewString = ls_NewString & ls_CurrentChar
    End If
Next

ConvertString = ls_NewString
End Function

Function ExtractAllSchemas() As Integer
' *****
' * Function: ExtractAllSchemas
' *
' * Description: Loops through the current open document in Word and
copies all Z styles
' *           to a temp document and then saves the temp doc as a
text file
' *
' * Input Parameters: none
' *
' * Output Parameters: none
' *
' * Return Value: integer - 0 = Error occured
' *                   1 = No errors
' *
' * Modification Log:
' * Date           Developer           Change
' * 10/14/96       S.Ziegler           Initial Development
' * 11/05/96       S.Ziegler           Added code to close active document when
done
' * 02/05/97       S.Zielger           Changed function name to
ExtractAllSchemas
' *
' *****

    io_Word.AppShow
    io_Word.ScreenUpdating 0

    is_DocumentDoc = io_Word.WindowName$()
    io_Word.StartOfDocument
    io_Word.FileNew
    is_ExtractedSchemasDoc = io_Word.WindowName$()
    io_Word.Activate (is_DocumentDoc)

    ExtractAllSchemas = ExtractSchemas(is_DocumentDoc)

End Function

```

```
Function ExtractSchemas(SourceWindowName As String)
```

```
Dim ls_StyleName As String  
Dim li_SelBegin, li_SelEnd As Integer
```

```
io_Word.ScreenUpdating 0
```

```
io_Word.Activate (SourceWindowName)  
While Not io_Word.AtEndOfDocument()  
    ls_StyleName = io_Word.StyleName$()  
    Select Case ls_StyleName
```

```
        Case ""  
            MsgBox ("Error.  StyleName$() returned empty string.")
```

```
        Case CONV_Z_SET_DEFINITION
```

```
            io_Word.Activate (is_ExtractedSchemasDoc)  
            io_Word.Insert CONV_HEADER_BEGIN & ls_StyleName & ">"  
            io_Word.InsertPara
```

```
            io_Word.Activate (SourceWindowName)  
            io_Word.SelectCurSentence  
            io_Word.EditCopy  
            io_Word.Activate (is_ExtractedSchemasDoc)  
            io_Word.EditPaste
```

```
            io_Word.InsertPara  
            io_Word.Insert CONV_HEADER_END & ls_StyleName & ">"  
            io_Word.InsertPara  
            io_Word.InsertPara  
            io_Word.Activate (SourceWindowName)
```

```
        Case CONV_Z_SCHEMA_SIGNATURE, CONV_Z_AXIOMATIC_SIGNATURE,  
CONV_Z_GENERIC_SIGNATURE
```

```
            io_Word.Activate (is_ExtractedSchemasDoc)  
            io_Word.Insert CONV_HEADER_BEGIN & ls_StyleName & ">"  
            io_Word.InsertPara
```

```
            ' Get Schema name  
            io_Word.Activate (SourceWindowName)  
            io_Word.LineUp  
            io_Word.StartOfLine  
            li_SelBegin = io_Word.GetSelStartPos()  
            io_Word.EndOfLine  
            li_SelEnd = io_Word.GetSelStartPos()  
            io_Word.SetSelRange li_SelBegin, li_SelEnd  
            io_Word.EditCopy  
            io_Word.Activate (is_ExtractedSchemasDoc)  
            io_Word.EditPaste  
            io_Word.InsertPara
```

```
            io_Word.Activate (SourceWindowName)
```

```

io_Word.LineDown

While io_Word.StyleName$() = ls_StyleName
  ' Select current line and copy it to
is_ExtractedSchemasDoc
  io_Word.StartOfLine
  li_SelBegin = io_Word.GetSelStartPos()
  io_Word.EndOfLine
  li_SelEnd = io_Word.GetSelStartPos()
  io_Word.SetSelRange li_SelBegin, li_SelEnd

  If li_SelEnd - li_SelBegin > 1 Then
    io_Word.EditCopy
    io_Word.Activate (is_ExtractedSchemasDoc)
    io_Word.EditPaste
    io_Word.InsertPara
    ' try to remove blank lines in schema
    'io_Word.InsertPara
  End If
  io_Word.Activate (SourceWindowName)
  io_Word.LineDown

Wend

io_Word.Activate (is_ExtractedSchemasDoc)
io_Word.Insert CONV_HEADER_END & ls_StyleName & ">"
io_Word.InsertPara
'io_Word.InsertPara
'io_Word.InsertPara
io_Word.Activate (SourceWindowName)
io_Word.LineUp

Case CONV_Z_SCHEMA_PREDICATE, CONV_Z_AXIOMATIC_PREDICATE,
CONV_Z_GENERIC_PREDICATE

  io_Word.Activate (is_ExtractedSchemasDoc)
  io_Word.Insert CONV_HEADER_BEGIN & ls_StyleName & ">"
  io_Word.InsertPara

  io_Word.Activate (SourceWindowName)

  While io_Word.StyleName$() = ls_StyleName
    ' Select current line and copy it to
is_ExtractedSchemasDoc
    io_Word.StartOfLine
    li_SelBegin = io_Word.GetSelStartPos()
    io_Word.EndOfLine
    li_SelEnd = io_Word.GetSelStartPos()
    io_Word.SetSelRange li_SelBegin, li_SelEnd

    If li_SelEnd - li_SelBegin > 1 Then
      io_Word.EditCopy
      io_Word.Activate (is_ExtractedSchemasDoc)
      io_Word.EditPaste
      io_Word.InsertPara
    End If
    io_Word.Activate (SourceWindowName)
    io_Word.LineDown

  Wend

```

```

        io_Word.Activate (is_ExtractedSchemasDoc)
        io_Word.Insert CONV_HEADER_END & ls_StyleName & ">"
        io_Word.InsertPara
        'io_Word.InsertPara
        'io_Word.InsertPara
        io_Word.Activate (SourceWindowName)
        io_Word.LineUp
    End Select

    io_Word.LineDown
    io_Word.StartOfLine
Wend

    io_Word.Activate (is_ExtractedSchemasDoc)
    io_Word.ScreenUpdating 1
    ExtractSchemas = 1

```

End Function

```

Function ExtractSelectedSchemas() As Integer
' *****
' * Function: ExtractSelectedSchemas
' *
' * Description: Extracts the Z Schemas in the current selection
' *
' * Input Parameters: none
' *
' * Output Parameters: none
' *
' * Return Value: integer - 0 = Error occurred
' *                  1 = No errors
' *
' * Modification Log:
' * Date           Developer           Change
' * 02/05/97      S.Ziegler           Initial Development
' *
' *****

```

```

    io_Word.AppShow
    io_Word.ScreenUpdating 0

    is_DocumentDoc = io_Word.WindowName$()
    io_Word.FileNew
    is_SelectionDoc = io_Word.WindowName$()
    io_Word.FileNew
    is_ExtractedSchemasDoc = io_Word.WindowName$()

    io_Word.Activate (is_DocumentDoc)
    io_Word.EditCopy
    io_Word.Activate (is_SelectionDoc)
    io_Word.EditPaste
    io_Word.StartOfDocument

    ExtractSelectedSchemas = ExtractSchemas(is_SelectionDoc)

    io_Word.Activate (is_SelectionDoc)

```

```
io_Word.FileClose 2
```

```
End Function
```

```
Function FormatZSchemaPredicate() As Integer
'
*****
' * Function: FormatZSchemaPredicate()
' *
' * Description: Converts the Schema Predicate style to the Latex format
' *
' * Input Parameters: none
' *
' * Output Parameters: none
' *
' * Return Value: integer - 0 = Error occured
' *                   1 = No errors
' *
' * Modification Log:
' * Date       Developer      Change
' * 10/18/96   S.Ziegler      Initial Development
' * 02/08/96   S.Ziegler      Changed to do conversion on the Windows
side
' *
' *
' *
'
*****
***
    Dim ls_Buffer As String
    Dim ls_Write As String
    Dim li_Continue As Integer

    ' Add beginning section
    ls_Write = Chr$(13) & Chr$(10) & "\where" & Chr$(13) & Chr$(10)
    WriteToConvertedDoc (ls_Write)

    ' convert rest of schema section
    ls_Write = ""
    li_Continue = True
    ls_Buffer = GetCurrentLine()
    While li_Continue

        If ls_Buffer <> "" Then
            ls_Write = ls_Write & ConvertString(ls_Buffer) &
Chr$(CONV_CARRIGE_RETURN) & Chr$(CONV_LINE_FEED)
            ls_Write = ls_Write & "\also" & Chr$(CONV_CARRIGE_RETURN) &
Chr$(CONV_LINE_FEED)
        End If
        ls_Buffer = GetCurrentLine()
        If ls_Buffer <> CONV_HEADER_END & CONV_Z_SCHEMA_PREDICATE & ">"
Then
            li_Continue = True

```

```

        Else
            li_Continue = False
        End If

    Wend
    ls_Write = Left(ls_Write, Len(ls_Write) - 8) ' remove "\also" from
final line
    WriteToConvertedDoc (ls_Write)

    ' write ending section
    ls_Write = "\end{schema}" & Chr$(CONV_CARRIGE_RETURN) &
Chr$(CONV_LINE_FEED)
    WriteToConvertedDoc (ls_Write)

    FormatZSchemaPredicate = 1

End Function

Function FormatZSchemaSignature() As Integer
' *****
' * Function: FormatZSchemaSignature()
' *
' * Description: Converts the Schema Signature style to the Latex format
' *
' * Input Parameters: none
' *
' * Output Parameters: none
' *
' * Return Value: integer - 0 = Error occured
' *                  1 = No errors
' *
' * Modification Log:
' * Date      Developer      Change
' * 10/17/96   S.Ziegler      Initial Development
' * 11/07/96   S.Ziegler      Trimmed ls_SchemaName
' * 02/06/97   S.Ziegler      Changed code to perform conversion on
Word Side
' *
' *****

    Dim ls_SchemaName As String
    Dim ls_Buffer As String
    Dim ls_Write As String
    Dim li_Continue As Integer

    ' Add beginning section
    ls_SchemaName = GetCurrentLine()
    ls_SchemaName = TrimAndRemoveTabs(ls_SchemaName)

    ls_Write = Chr$(13) & Chr$(10) & "\begin{schema}{" &
ConvertString(ls_SchemaName) & "}" & Chr$(13) & Chr$(10)
    WriteToConvertedDoc (ls_Write)

    ' convert rest of schema section
    ls_Write = ""
    li_Continue = True

```

```

ls_Buffer = GetCurrentLine()
While li_Continue

    If ls_Buffer <> "" Then
        ls_Write = ls_Write & ConvertString(ls_Buffer) &
CONV_SCHEMA_DEC_END_LINE & Chr$(CONV_CARRIGE_RETURN) &
Chr$(CONV_LINE_FEED)
    End If
    ls_Buffer = GetCurrentLine()
    If ls_Buffer <> CONV_HEADER_END & CONV_Z_SCHEMA_SIGNATURE & ">"
Then
        li_Continue = True
    Else
        li_Continue = False
    End If

Wend

ls_Write = Left(ls_Write, Len(ls_Write) - 4) ' remove "\\" and
final line
WriteToConvertedDoc (ls_Write)

FormatZSchemaSignature = 1

End Function

```

```

Function FormatZSetDefinition() As Integer
' *****
' * Function: FormatZSetDefinition()
' *
' * Description: Converts the Set Definition style to the Latex format
' *
' * Input Parameters: none
' *
' * Output Parameters: none
' *
' * Return Value: integer - 0 = Error occurred
' *                  1 = No errors
' *
' * Modification Log:
' * Date           Developer           Change
' * 10/17/96      S.Ziegler           Initial Development
' *
' *
' *
' *****
Dim ls_SetName As String
Dim ls_Write As String

ls_SetName = GetCurrentLine

ls_Write = "\begin{zed}" & Chr$(13) & Chr$(10)
ls_Write = ls_Write & ls_SetName & Chr$(13) & Chr$(10)
ls_Write = ls_Write & "\end{zed}" & Chr$(13) & Chr$(10)

WriteToConvertedDoc (ls_Write)

```

```
FormatZSetDefinition = 1
```

```
End Function
```

```
Function GetCurrentLine() As String
    Dim StartPos, EndPos As Integer

    io_Word.StartOfLine
    StartPos = io_Word.GetSelStartPos()
    io_Word.EndOfLine
    EndPos = io_Word.GetSelStartPos()

    io_Word.SetSelRange StartPos, EndPos
    GetCurrentLine = io_Word.Selection$()
    io_Word.LineDown
```

```
End Function
```

```
Function Setup(DestinationFile As String, OutputType As Integer, Font As
Integer, WordObj As Object) As Integer
' *****
' * Function: Setup()
' *
' * Description: Sets the instance variables and reads the lookup array
' *
' * Input Parameters:
' *     DestinationFile As String - Path and name of text file to be
created
' *     OutputType As Integer - Type of ASCII file to create 1 =
Latex
' *     Font As Integer - Type of Font being used
' *     WordObj As Object - OLE object that must have been
created before function call
' *
' * Output Parameters: none
' *
' * Return Value: integer - 0 = Error occured
' *                   1 = No errors
' *
' * Modification Log:
' * Date      Developer      Change
' * 10/14/96   S.Ziegler      Initial Development
' *
' *
' *
' *****
    Dim ls_Buffer, ls_Code, ls_LookupFilename As String
    Dim li_EqualSign, li_Index, li_LineNumber As Integer
```

```

    CONV_DOCUMENT_HEADER = "\documentstyle[zed,z-eves]{article}" &
Chr$(10) & Chr$(10) & "\begin{document}"
    CONV_DOCUMENT_FOOTER = "\end{document}"

If DestinationFile = "" Or OutputType = 0 Or Font = 0 Then
    Setup = 0
End If

'*****
' Initialize instance variables
'*****
is_DestinationFilename = DestinationFile
ii_OutputType = OutputType
ii_Font = Font

Set io_Word = WordObj

'*****
' Initialize LookupArray
'*****
For li_Index = 0 To 256
    is_LookupArray(li_Index) = ""
Next li_Index

' Get correct data file for font
Select Case ii_Font
    Case CONV_Z_FONT_1
        ls_LookupFilename = App.Path & "\" & CONV_Z_FONT_1_FILENAME
End Select

' Read data file and fill in lookup array
Open ls_LookupFilename For Input As #1
Do While Not EOF(1)
    Line Input #1, ls_Buffer
    li_EqualSign = InStr(ls_Buffer, "=")
    If li_EqualSign = 0 Then
        Close #1
        MsgBox ("Line:" & ls_Buffer & " doesn't have an equal
sign.")
        Setup = 0
        Return
    End If

    li_Index = CInt(Left(ls_Buffer, li_EqualSign - 1))
    ls_Code = Mid(ls_Buffer, li_EqualSign + 1, Len(ls_Buffer))
    is_LookupArray(li_Index) = ls_Code

Loop
Close #1

Setup = 1

End Function

```

```

Function TrimAndRemoveTabs(as_Input As String) As String
' *****
' * Function: TrimAndRemoveTabs
' *
' * Description:   Searches a given string for Tab characters and
removes
' *               them and also removes leading and trailing spaces
' *
' *
' * Input Parameters: as_Input      String      String to remove tabs
from
' *
' * Output Parameters: none
' *
' * Return Value: string - "" = Error occured
' *                   otherwise returns string without tabs and
spaces
' *
' * Modification Log:
' * Date      Developer      Change
' * 11/07/96   S.Ziegler      Initial Development
' *
' *****

    Dim ls_NewString As String
    Dim i As Integer
    Dim li_TAB As Integer

    li_TAB = 9

    For i = 1 To Len(as_Input)
        If Mid(as_Input, i, 1) <> Chr$(li_TAB) Then
            ls_NewString = ls_NewString & Mid(as_Input, i, 1)
        End If
    Next

    ls_NewString = Trim(ls_NewString)
    TrimAndRemoveTabs = ls_NewString

End Function

Function WriteToConvertedDoc(as_Line As String)

    io_Word.Activate (is_ConvertedDoc)
    io_Word.Insert as_Line
    io_Word.Activate (is_ExtractedSchemasDoc)

End Function

```

Module File

```
Global Const SWP_NOMOVE = 2
Global Const SWP_NOSIZE = 1
Global Const FLAGS = SWP_NOMOVE Or SWP_NOSIZE
Global Const HWND_TOPMOST = -1
Global Const HWND_NOTOPMOST = -2

Declare Function SetWindowPos Lib "User32" (ByVal hWnd As Integer, ByVal
hWndInsertAfter As Integer, ByVal x As Integer, ByVal y As Integer,
ByVal cx As Integer, ByVal cy As Integer, ByVal wFlags As Integer) As
Integer
Global WordObj As Object
```

```
Sub SetWaitForm(Caption As String, Increment As Integer, Total As
Integer)
```

```
    WaitForm.ProgressBar1.Value = (Increment / Total) * 100
    WaitForm.CaptionLabel = Caption
    DoEvents
```

```
End Sub
```

```
Sub ShowWaitForm()
```

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

```
End Sub
```