# Using Excel and PowerPoint to build a Reverse Engineering Tool

by

Fang Yang
B.Sc., Wuhan University, 1996

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Supervisor: Dr. Hausi A. Müller

# ABSTRACT

This thesis introduces a new reverse engineering tool development practice by presenting the development of PowerExcelRigi, a reverse engineering tool built by leveraging Rigi and two selected host tools, PowerPoint and Excel.

PowerPoint and Excel, both components of the Microsoft Office Suite, were selected as the host tools for this project because of their large user base, excellent end-user programmability and strong visualization capabilities. The original Rigi reverse engineering tool is used as the backend data engine to make use of its graph computing capabilities. Using PowerExcelRigi, users appreciate the familiar user interface of Excel and PowerPoint and at the same time benefit from the efficiency of Rigi.

A custom toolbar in Excel provides a means to perform several reverse engineering tasks. This toolbar follows the standard Office user interface design and seamlessly integrates reverse engineering tasks into the Office environment. Reverse engineering tasks implemented include reusing given program artifacts from Rigi format program fact files, analyzing the artifacts and visualization the analysis results by using Excel, and then reproducing Rigi graphs in PowerPoint. Some Rigi scripts demonstrating typical Rigi functionality have been executed entirely through the Office interface without noticeably using Rigi. Excel and Rigi use a loose, file-based data interchange method to interoperate with each other.

In comparison to a new tool with a dedicated user interface, PowerExcelRigi offers users the benefit of the cognitive support derived from their familiarity with the host tool, which decreases the learning barrier to using the new tool. This approach will

help solve the low adoption problem suffered by many reverse engineering tools. At the same time, development cost is significantly reduced by reusing Rigi, Excel and PowerPoint as existing components. We believe this to be a promising direction for the development of lower-cost, more adoptable low reverse engineering tools.

# Table of Contents

# List of Figures

**ACKNOWLEDGMENTS**

I would like to thank everyone who supported me and offered me guidance throughout this research. In particular, the advice by Dr. Hausi Müller, who encouraged me to research in a variety of directions and inspired me to find the topic of this thesis, was much appreciated. I would like to thank Anke Weber, who gave me a lot of guidance and support. Without her excellent work on Live Documents, most of my research work could not have been accomplished. I would also like to thank Eva van Emden and Piotr Kaminski for their help with editing this thesis.

I am also grateful for the financial support from the University of Victoria and Dr. Hausi A. Müller and his research projects.

Finally, I wish to thank my family and friends for all their encouragement and support.

# Chapter 1

# Introduction

## 1. 1 Problem

A very widely cited survey study by Lientz and Swanson in the late 1970, and replaced by others in different domains, exposed that on average, seventy percent of software costs are spent on maintenance [1, 2]. From the first day a piece of software is released, there is a need for maintenance. Not only are new bugs detected which need to be fixed, but changing requirements such as evolving tax laws, new business rules, or the need to adapt to new technologies such as e-commerce platforms or Web-based user interfaces. Over time the maintenance requirement may become critical. Software engineers face pressure to continue to evolve their products to keep them functionally correct and competitive. Reverse engineering has been a very promising technology to face these maintenance challenges [3].

However, the reverse engineering process is generally considered to be inefficient. The inefficiency of software maintenance has been related to the difficulty in comprehending software systems [4]. Working on a legacy system is different from designing a new system from scratch. Reengineers have to understand the legacy system first before they can start to redesign it. However, the gap between the information required and the information available increases over time for a number of reasons including developers leaving, documents being lost or becoming out of date, and greater complexity gradually increasing as code is added. At the same time, developers usually have insufficient time to finish their work; they need to meet deadlines set by management, by customers, or even by competitors. Thus, software comprehension has been considered a key bottleneck of software maintenance.

A classical way to reduce this inefficiency is to develop reverse engineering tools to support software engineers in the process of analyzing and understanding complex legacy systems, which may involves millions of lines of code [5]. Some tools have been implemented to extract information about relevant artifacts from source code and present them in a way that facilitates comprehension. For example, the Imagix 4D produced by Imagix corporation can help with providing accurate, up-to-date information by generating comprehensive documents automatically [6]. Rigi has the ability to produce the call diagrams, and ShriMP [7] is a visualization tool that eases browsing and searching of source code [8]. However, since none of these tools is essential to complete a task, in most cases software reverse engineers are not forced to use them. These tools, which facilitate software engineers' understanding of the subject system, suffer from low adoption in both academia and industry [5, 9]. Simple, widely available search tools such

as Unix *grep* are still the most widely used tools for program understanding and, thus, specialized reverse engineering tools are not widely used [10].

The adoption of a reverse engineering tool depends on many factors. Some are related to the tool itself, such as its functionality and usability; other factors, such as the previous working habits of software engineers have few relationships with the tool itself. The ACRE (Adoption-Centric Reverse Engineering) research project at the University of Victoria under the direction of Professor Hausi A. Müller focuses on how to improve reverse engineering tools to ease adoption [11].

For every reverse engineer, or we can safely say for most computer users, there are one or more existing tools that they use regularly during their daily work. If a reverse engineering tool is somehow related to the one that the potential user is already using and knows well, and at the same time provides most, if not all of the functionality of an existing reverse engineering tool, the user might be more initiative in using this new tool compared to the existing one. Normally when starting to use a new tool, the user feels that he or she knows little about the product and will need to spend a significant amount of time learning it, he may question whether it is worthy of the time and effort, and become hesitate to do so. However, if the new tool is similar to a familiar tool or even becomes part of it, then he or she already knows quite a bit about the new tool and does not have to spend a dramatic amount of time to learn it. If, in addition to the new tool's learning curve is low, it is proven that the tool has useful functionality and will improve his or her working efficiency accordingly, We are interested in building a tool this way and we hope the user will be more willing to adopt the new tool.

Thus, if the new reverse engineering tool is related to another tool that already exists on the reverse engineer's computers, users can benefit from the cognitive support derived from the familiarity and knowledge, and the tool is featured by its familiar user interface and low learning curve. Those reverse engineers may be less reluctant to use the new tool to assist in their reverse engineering tasks, compared to using tools with unfamiliar interfaces. At the same time, for tool developers this will be a new development approach worthy of research. The benefits of code reuse include cost saving during the initial development phase as existing functionality is reused. Also, maintenance costs may be decreased because of ongoing support from the vendor. However code reuse may also cause difficulties such as less control of the system and learning curves of the existing tools [12].

The assumption of the project now is that there is an existing reverse engineering tool, which helps with program understanding and can increase the work efficiency, such as Rigi [13]. The approach is to investigate the possibility of leveraging this existing reverse engineering tool with other suitable tools, which is referred as host tool. Host tools can be popular office tools or software engineering tools. The problem now is what makes a suitable host tool? Once the host tools, Excel and PowerPoint, are selected, is it possible to integrate these different purpose tools? If yes, how to integrate them? The project tries to resolve these problems and also tries to work out a feasible development process, which will be beneficial for other software engineering researchers. Evaluation of the project will be done by implementing a prototype of PowerExcelRigi, comparing PowerExcelRigi with Rigi, and comparing the ACRE approach with certain traditional

tool development methods. In addition, the potential of different host tools will be examined by analyzing their effects on the development processes.

## 1.2 Development Investigation

The development of an adoption-oriented tool can be roughly grouped into four stages: *tool selection, high-level design, tool development* and *evaluation.*

### 1.2.1 Tool Selection

The selected base tool is usually a successful research reverse engineering tool, which suffers of adoption problems. The base tool provides a functional example for the new tool and also become reference points for evaluating the learning curve and cognitive support. The host tool can be a popular office tool or a software engineering tool, which will be applied in the development of the new reverse engineering tool and should increase the final product's cognitive support, in comparison to the base tool.

The selection of the host tool is critically important for the final success of an ACRE product. To choose an appropriate host tool, we need to follow some basic rules. The size of the user base is one of the most important criteria because only the established users experience the facilitated cognitive support of the new tool. If a user knows nothing about the host tool, the new reverse engineering tool requires the same learning effort as the base tool, if not more. We target the potential users of the new adoption-oriented reverse engineering tool to be a subset of the host tool's existing user base. It is obvious that the bigger the user base the better.

The host tool's functionality is another main criterion. Basically, in order to handle reverse engineering tasks, the host tool needs data management and data visualization ability. When selecting the host tool, developers should also pay attention to native function boundaries which may restrict the potential of the new tool. It is obvious that no one will be able to use a calculator as an editor. Furthermore, if there are more analogous tasks between the host tool and the new tool we are designing, the less effort we may expect for the implementation. Existing menus like "Open" and "Save" may be able to reused with little effort.

The extensibility of the host tool is equally important. If the vendor does not allow the end users to program the host tool, no matter how powerful and popular the tool is, it is of no use of our purposes. The more control the developers have over the host tool, the easier the job will be. The cost to customize software without end user programmability may be overwhelming in most cases. At the same time, how the vendor provides access to the product will affect the design and implementation as well. For example, Rigi only allows the end users to customize its user interface, access the internal C/C++ functions, and automate operations through RCL (Rigi Command Language), as the end-user programming language. Thus, the only way to customize Rigi is to start Rigi, load RCL commands into Rigi, and let Rigi execute the commands. On the other hand, some products may use more complex technologies, such as component technologies, which imply that other developers are able to make use of the software by invoking one or more components without running it visibly. The way to invoke the components varies, but generally it follows some standards to be more accessible from different sources.

So a good host tool for our project should meet following criteria. It should be popular in the reverse engineering community, be end-user programmable, and have an efficient data representation and handling ability.

### 1.2.2 High Level Design

The architecture design is different from traditional development approaches. Designers have to learn how to build software around the host tool. One essential question is how to manage the relationship between the host tool and base tool. Should we abandon the base tool, use one host tool or a tool suite such as Microsoft Office or Sun's StarOffice, and extend it into a new reverse engineering tool that provides similar functionality as the base tool? Should we leverage both host tool and base tool instead of abandoning one? Should we even integrate several third party tools as host tools? All three options are feasible, and have strong points and weak points depending on the project requirements. Thus, the first step in the high level design is to investigate uses and relationships of both the host and the base tool, and relationships between them in the new tools.

In contrast to the classical software design process, where a tool is designed from scratch, the architecture and design not only rely on user requirements, but also depend heavily on the programming architecture of the host tool. Even having chosen the most suitable tool, with respect to both functionality and extensibility, developers may suffer from the inherent limitations brought on by the host tool. It is hard to escape the boundary set up by the tool vendor. For example, when we tried to customize PowerPoint, we went to some trouble to catch user's mouse selection of graphic items because there is not such an event exposed. Developers have to study the host tool

carefully and find out what they can do according to both the user's requirements and the tool's allowances. Thus, developers need to balance what needs to be done and what can be done. Once these options can be explored, we can start to design the architecture of the new tool.

Solving the above questions is still not enough; we need to gather more information before we can begin with the implementation. We need to make decisions about whether we should mask or change some user interface of the host tool. Designers may prefer either side for different reasons. Keeping the original user interface allows end users to enjoy the working environment they feel comfortable with and incur a minimal learning burden. It also benefits the users with more cognitive support and as a result higher tool adoption ability. On the other hand, if the decision is to allow user interface updates, developers have more options for design and development. Thus, there are more opportunities to make the new tool bug free and it is easier to implement the required functionality. If the final decision is to not touch any of the presented user interface, designers face more design challenges and implementation issues. If changes are permitted, there are more concerns, such as which parts need to be changed, whether they are able to be changed or not, if yes, how to change them. All these design decisions have a strong influence on the features of the new tool.

### 1.2.3 Tool Development

To enhance the extensibility, tool vendors provide end-user programmability in different ways. Sometimes it is a software dependent scripting language. For example, Sun's StarOffice features Star Basic; Microsoft Office uses Visual Basic for Applications and Visual Basic Script. End users customize the software by writing scripts in the host's

specific scripting language. Sometimes functions are exposed in a more programmable way. For example, Microsoft's COM/DCOM technology opens most of its Office functionality to third party products. Following a predefined procedure, programs can use different programming languages to use these binary components in their own products. Both ways require programmers to have good knowledge of the host tool's technology for efficient development.

Utilization of the host tool introduces another serious issue to developers: version control. Regardless whether the solution has a tight or loose relation to the host tool, developers rely on certain internal characteristics of the host tool. Since developers have no control over the host tool, whenever a new version or even a new patch is released, they have to test their new tool and they may need to re-implement parts or even the whole product.

### 1.2.4 Evaluation

The whole ACRE project introduces a new methodology for reverse engineering tool development, whose end result is expected to help with the adoption problems suffered by some reverse engineering tools. The ACRE project group assume that users prefer the tools with a less cognitive load of learning and higher cognitive support. Thus, there will be less reluctance for end users to adopt a new tool with such characteristics. In order to verify the idea and find an appropriate development process, the ACRE project decided on trying to use host tools to rebuild reverse engineering tool, Rigi. Office tools have been selected as host tools. The evaluation works include implementing the PowerRigiExcel by using PowerPoint and Excel, comparing the new tool with Rigi,

comparing Office with Lotus Notes as the host tool, and evaluates the development process as a whole.

## 1.3 Approach

Rigi is the existing reverse engineering tool featuring a Graphical User Interface (GUI), the ability of graph exploration and information extraction, and end-user programmability. Microsoft Office Suite is a widely used set of business tools. Each of its components addresses a general category of business tasks. For example, Word is widely used for document editing and Outlook is a multipurpose scheduler and email client. The Microsoft Office Suite has been chosen as the host tool suite, and PowerPoint and Excel have been exploited in the first instance as host tools. For the remainder of this thesis, the term Office is used to reference the Microsoft Office.

Three different approaches have been tried in this research. They are converting Rigi Standard Format (RSF) file, using the RigiOfficeHarness application and using the extended Office. Each approach is explained below in detail.

The first idea is to convert RSF files, which are pure text, into one of the Office native document formats. Thus, users can open the newly generated files with corresponding Office Application like normal Office files, such as Word documents and PowerPoint presentation files. We can also embed some macros (a sequence of commands in Office components) within these files, which are recognizable and executable by Office. When these files are opened, some operations provided by macros will be executed automatically or manually. Using this method, users are able to perform

some information analysis. Since users have full access to the data in the file and macro source code as well, this approach promises flexibility. Users can re-program and re-analyze the information freely. It is very similar to any other Office documents with macros.

The other approach is to build the RigiOfficeHarness application, a standalone program. This RigiOfficeHarness program accepts users input, sends requests to Rigi, demands answers from Rigi, transfers these answers into Office documents and then sends the documents to one of the Office Suite applications. In this design, Office functions as a pure input/output viewer. This approach builds a harness software of both the base tool and the host tool and these three programs cooperate closely. It promises powerful functionality by easing integration Office, Rigi and other possible third party technologies.

The extended Office solution is an add-in to Office. A user will have a customized tool bar or menu to open reverse engineering documents and assist the reverse engineering tasks. Most other functions of Office are kept the same and Rigi is running as the back-end server and doing some of the computing. The implementation of this approach will be more adoptable than RigiOfficeHarness by applying the original Office user interface, and provides more powerful functionality than what can be provided by the macros associated with the RSF file after converting.

Each approach has its advantages and disadvantages. My final implementation is based on the third approach, the extended Office.

## 1.4 Solution

Extending an existing Office tool with a large user base as a reverse engineering tool is a promising way to improve the new tool's cognitive support and lessen the learning curve. We believe the extended Office approach leverages the functionality and the adoptability. The only notable change in the user interface is a new toolbar for reversing engineering tasks. It can analyze RSF files and imitate some Rigi functions. It follows the Office working rules in exactly the same way as other toolbars, such as the graph tool bar and the form toolbar. It benefits from Rigi's features as well, that is, Rigi does some calculations. It not only avoids the efforts to re-implement existing Rigi functions, but also diminishes the efficiency problem that comes with Office. Concerning communication between Rigi and extended Office, a XML-style file based communication schema is implemented.

## 1.5 Outline of the thesis

This thesis is organized as follows. Chapter 2 provides background on reverse engineering tools, and COTS-based software development. Chapter 3 describes several related projects of tool development which is taking use of existing tools. Chapter 4 describes the reasons for selecting Microsoft Excel and PowerPoint as the host tools, and background of the Microsoft Office solution. Chapter 5 introduces three high level designs to integrate Rigi and Microsoft Office. Chapter 6 describes a case study about extending Microsoft Office into a reverse engineering tool. Chapter 7 compares PowerExcelRigi with Rigi, Microsoft office with Lotus Notes as host tool, and reports on

our development experience. Chapter 8 summarizes the contributions of this thesis and proposes future work.

# Chapter 2

# Background

Software development is a fast growing and evolving field. New technologies are emerging all the time and user requirements are changing frequently as well. Lots of effort is devoted to developing new software to meet these changes and many software engineers are working hard to reverse engineer legacy systems and bring them up to date. At the same time, researchers are trying to develop theories and tools to assist these reverse engineering tasks. However, few of these tools have been widely adopted. One reason for the reluctance to adopt reverse engineering tools is the high cognitive load of learning. The ACRE project is trying to identify an efficient approach to develop a tool whose usage can be learned easily. The final implementation approach is using a host tool as the development base, which is technically similar to the COTS (Commercial Off-the-Shelf) based software development. Studying the advantages, disadvantages and risks of the COTS experience will be beneficial to the ACRE project.

## 2.1 Reverse engineering

Due to the development of computer-related technologies, such as Web services, object-oriented systems and component technologies, the need to move a legacy system

to a new platform or adopt a new technology is acute. Due to changing business rules, the requirement for software evolution is increasing. Due to software shifting and turnover of development team members, the need to re-document software systems is crucial. All of these requirements produce a need for methods, tools, technology and theory to evolve and exploit existing legacy systems efficiently and cost effectively. Reverse engineering becomes a very promising approach to fulfill these tasks. Reverse engineering is targeted at "analyzing the subject system to identify its current components and their dependencies, and to extract and create system abstractions and design information" [14]. One important goal of reverse engineering is program understanding. It has been estimated that fifty to ninety percent of evolution work is devoted to program comprehension or understanding [15, 16]. In order to modify the existing software, designers need to understand the current system using their computer science knowledge, domain knowledge, as well as some comprehension strategies. Only after they have a good comprehension of the target system, are they capable of working out how to update, how to evolve, and how to migrate the legacy system.

## 2.2 Overview of reverse engineering tools

To understand and analyze source code is difficult and time consuming, especially when reverse engineers are working with large and complex commercial software systems, which may be millions of lines of code long. In order to enhance program understanding, most reverse engineering tools automate some tasks and generate text documents or graphic diagrams from source code. They may be able to help engineers generate higher-level design diagrams as well. All of these capabilities can reduce time

spent on tedious work and help engineers focus on the tasks that have to be done manually or semi-manually, such as design recovery and architecture recovery.

### 2.2.1 The Rigi tool

Rigi is a reverse engineering tool developed over more than ten years by the Rigi group at the University of Victoria. It helps software engineers understand and re-document legacy systems by visualizing the artifacts extracted from the source code. These artifacts, representing components and relationship between different components, are stored in a text-based repository in RSF format. The Rigi environment consists of two parts. One part is language-specific parsers, which are used to extract major artifacts from the source code of the subject system. The core part is Rigiedit, a graph editor tool enhanced with additional functionality for reverse engineering tasks. It uses nodes to represent system components and directed arcs for the interrelationships between components. These nodes and arcs are visualized with different colours, as shown in Figure 2.1. Users can edit, manipulate, annotate, browse, and explore the nodes/arcs interactively. In short, using Rigi to re-document legacy systems generally requires two steps. First, parse the source code to generate its RSF file, and then use Rigiedit to analyze this RSF file to identify higher-level abstractions.

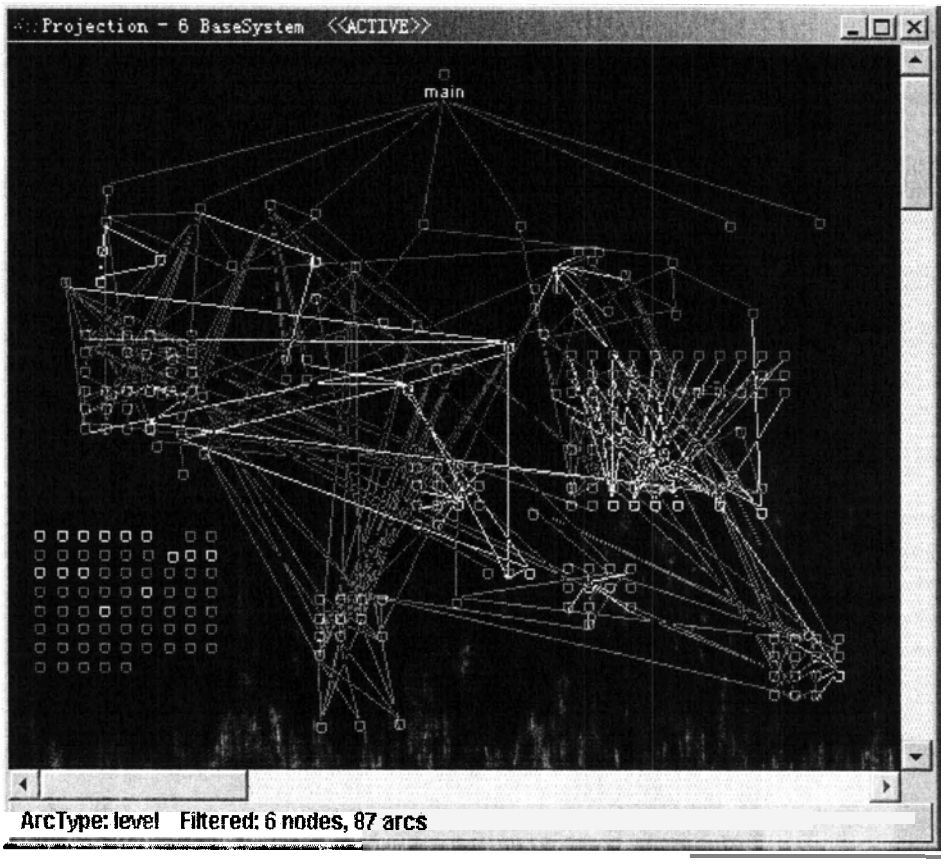



**Figure 2.1: Visualization of Ray Tracer in Rigi**

In comparison to some other reverse engineering tools, such as SNiFF+ [17] and Imagix [6], Rigi is distinguished by its flexible functionality, easy operation and end-user programmability. Rigi has a good reputation for its efficiency and wide applicability [18]. For example, as early as 1998, Rigi was used to semi-automate the migration of a compiler optimizer written in PL/IX, which consisted of approximately 300K LOC [19]. However, even though Rigi is freely available and has many advantages and successful applications, it has not evolved into a widely used tool.

### 2.2.2 Cognitive support

Our effort to re-implement Rigi on top of other widely used tools is based on the assumption that the new Rigi will be more adoptable due to the cognitive support derived from the host tool. There is no numerical scale to weigh a tool's adoptability. There is no agreed-upon definition of cognitive support either. Nor is there a complete theory about how cognitive support can affect a tool's adoptability. However, if there is a business or software engineering tool that has been extensively used by reverse engineers, we can safely say these engineers will be more likely to consider this particular tool or a similar one when they work on reverse engineering tasks. The ideal situation is that the new tool has exactly the same GUI as the tools they are accustomed to, or there are only minor differences.

This may be compared to using a cell phone as an email client. There are huge amount of people, who use a cell phone everyday may only know little about computer, and hesitate to learn a complex software system but need or want to use email, will not refuse to try his cell phone to send or receive email. First, it does provide some useful functions and works efficiently. Customers can access to their emails using this new cell phone solution. Furthermore, it can send and receive emails anytime and anywhere, freeing customers from carrying a heavier computer. Second, it is easy to use. Customers use a familiar tool without having to learn to work with a new device. Compared to traditional cell phones, there may be a couple of new buttons and options, but the learning curve is much less than learning how to use a brand new device. Third, this cell phone doesn't sacrifice any functionality available in the traditional cell phone for the new email features. Customers enjoy the basic phone calls, extended services such as

call display or call transfer in the same way. In a summary, even using cell phone for email may not be a good choice for those advanced computer users, but it will help others.

Iyad Zayour and Timothy Lethbridge [5] provide another approach to leveraging cognitive support of reverse engineering tools to improve adoption. He gives higher priority to reducing cognitive load than to other requirements such as functionality and efficiency in the tool design. He divides a task into processes, lists options for each process, quantitize them and selects the one that has the least cognitive load. Thus, he builds a model with minimal cognitive load, which eases the user's working burden.

## 2.3 COTS-based software development approach

COTS products are designed to be easily installed and to operate with existing system software. COTS-based software development means integrating COTS components as part of the system being developed. This development style is becoming increasingly commonplace in the software development community. According to David Carney, there are three types of COTS-based systems depending on the number of COTS used and their influence on the final system [20]. The first is the so-called Turnkey System, which is built around one commercial product, such as Microsoft Office or Lotus Notes. In this case, only one COTS is used, and customization does not change the nature of the original COTS. He also describes Intermediate Systems, which are built around one COTS, but integrate with other components, either commercial or in-house developed [20]. Finally, other systems are built by integrating several equally important COTS [20].

The development process of the ACRE project is similar to that of COTS-based system development to some degree. One approach to the ACRE project is to try to apply a widely used, open and commercial business office tool as the host tool. Another approach to the ACRE project is to use a platform independent, open technology as the main medium to deliver the reverse engineering information. Both approaches imply that the ACRE process will vary from the widely used top-down software development process and take on some features of COTS-based software development. Since COTS-based software development is a well-studied field, there is valuable experience that can be borrowed for the ACRE project.

Morisio and Seaman summarize the COTS software development process as follows: "COTS projects were obliged to follow a process quite different from traditional projects, with more effort put into requirements, test and integration, and less in design and code" [21]. This approach is well-suited to the ACRE project. For example, COTS development methodology tells us what is the potential effects COTS components may have on the final product's functionality design, comparing to traditional techniques. Although at the beginning requirements have to be gathered, analyzed and documented, in the same way as the traditional software development process, functions that can be implemented are those facilitated by the COTS components.

Requirements always need to be reviewed after the COTS components have been selected. The final implemented requirements of ACRE will partially depend on the host tool selected, since the possible host tools are limited and each tool has distinguished characteristics and end-user programming potential. Designers of the ACRE project spent time analyzing the functionalities of reverse engineering tools and trying to decide

which features were most important. Visualization the program artifacts and the ability to reorganize generated graphic items were two examples we tried to fit into various potential host tools. However, no matter which host tool is selected, its features may make it impossible to implement some of the well-liked functions and may make it easier to develop less popular functions. All of these conditions have to be considered. Furthermore, the requirements sometimes need to be given a second thought as well. Some features will be so difficult or even impossible to implement due to the limitation of the host tool, the host tool may be considered unsuitable and the features can not be supported. Some functions may not be provided by the host tool and be essential for a reverse engineering project, but will be a nice supplement if it is supported by the host tool. Thus, the requirements analysis is no longer only dominated by users; it depends on the host tool as well. This is a major difference compared to the traditional development process.

## 2.4 Summary

Reverse engineering tools, such as Rigi, have been developed and proved to be helpful in assisting program understanding. Concerning about adoption problems of these tools, we believe cognitive support leverage from the pre-selected host tools can be an incentive. Some experiences and lessons can be borrowed from the COTS-based software development to ease the adoption-centric tool development.

# Chapter 3

# Related work

Several existing projects, which attempt to develop tools on top of other existing software, are discussed below. In general, these projects apply some similar design and implementation techniques to the ones used in the ACRE project. The host software used in both cases may have some functionality comparable to the requirements of the new tool and at the same time, the host software must be open to end-user programming. Most developers find benefits such as accelerated development, lower cost and better quality arising from this methodology, but one of the most important benefits for the new tools is the ability to take advantage of the existing cognitive support for the host tool.

Some parallel work is currently under way at University of Victoria. There are feasibility studies of building reverse engineering tools on different host tools, such as StarOffice and Lotus Notes [22]. SVG (Scalable Vector Graphics) is an XML (eXtensible Markup Language) based image description standard [23]. In addition, there is work being done on how to utilize this cross-platform, text-based graphic information presentation technology to improve the adoption for the reverse engineering tool [24].

## 3.1 The ISI Visual Design Editor Generator

The ISI Visual Design Editor Generator is a design tool to produce visual development environments using a domain-specific language [25]. It uses PowerPoint's graph handling ability, applies PowerPoint as both graphic middleware and end-user GUI and finally extends PowerPoint into a design editor generator. This design editor generator is domain independent and can be used by experts from different domains.

The ISI Visual design editor generator is implemented as an extension of PowerPoint, programmed in Visual Basic. Technically, the extension is a COM server that receives "events" as the user modifies a design. The same module acts as a COM client of PowerPoint enabling it to navigate through a design and to paint analysis feedback directly onto the design.

The ISI Visual design editor generator provides some useful hints for the implementation of this project. For example, the COM server runs as an "in-process" component, which means it is incorporated into the PowerPoint process itself. Method calls are efficient when both client and server are part of a single operating system process. Another consideration is how to achieve more flexibility while trying to save the standard PowerPoint's GUI. Lots of effort is devoted to break the limitations imposed by this rule to get more flexibility. Event handling is one of the examples. In PowerPoint, there is a lack of "event" notification. Most of the design editor's activity must be triggered by the state change initiated from some GUI events. However, the interface of PowerPoint does not make many events available to programmers. This problem is encountered many times when working with PowerPoint. The ISI editor generator resolves this problem by listening to system level events and guesses whether

events are for ISI or not. If the answer is yes, the generator decides which object(s) of PowerPoint are selected and directs to call the corresponding event handler(s) [26].

The ISI Visual design editor generator adds a button to the standard format toolbar, which is highlighted by a circle in Figure 3.2. This button is used to turn the generator on and off. In the "design off" mode, PowerPoint acts the same as without the generator installed. When the generator is turned on, as shown in Figure 3.2, a group of menus and buttons are added to implement most design requirements. The generator customizes the event handler to enhance the functionality as well. It is worth noticing that the generated design editor is saved in the regular PowerPoint Presentation format. So if you load it using a normal PowerPoint operation, it will look and act as a normal presentation, as shown in Figure 3.3. If you load it into the generator, the generator will recognize the specially defined shapes and colours to produce semantic annotations as authors compose briefings, which can be transited to the Semantic Web[27].

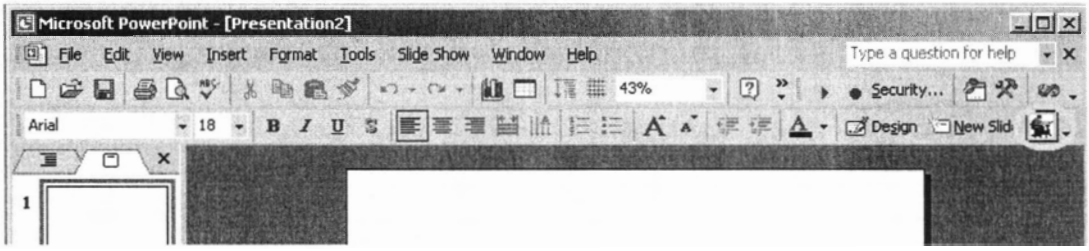



**Figure 3.1: ISI Visual design editor user interface when disabled**

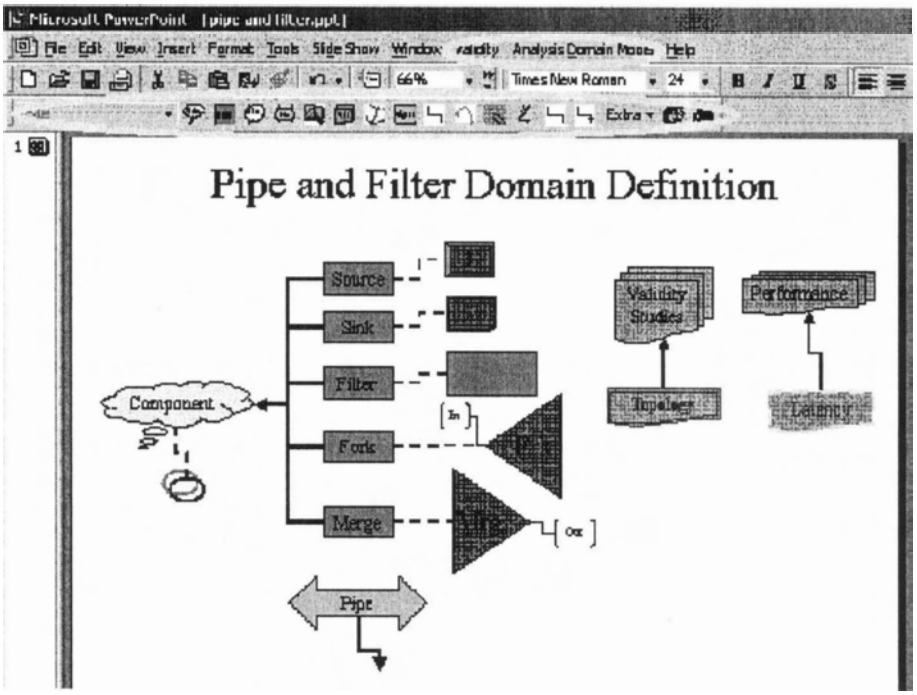



**Figure 3.2: ISI Visual design editor user interface when enabled**

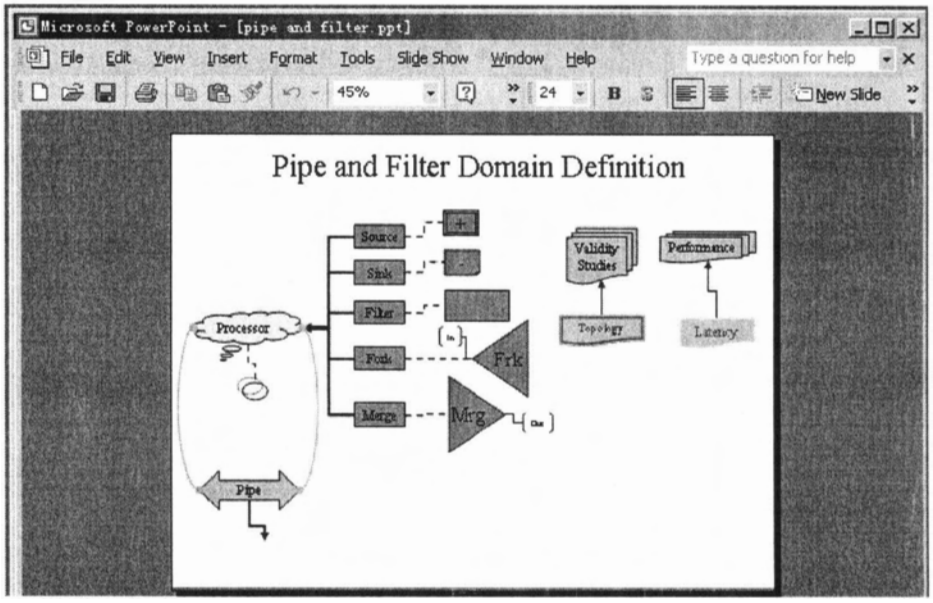



**Figure 3.3: Generated Design loaded in PowerPoint without ISI**

## 3.2 Desert Programming Environment

The Desert programming environment was developed at Brown University [20]. It is able to integrate different tools, and is targeted to be able to support all phases in software development. It uses FrameMaker as the central tool, embeds hypertext links that interconnect all aspects of software engineering, enables the programmer to view a software system as a single, dynamic document, and maintains an open environment for future tool integration [28].

The Desert environment applies three different integration mechanisms. The first is the standard message-based control integration. It introduces FIELD as a message server and uses Sun's ToolTalk as the message bus. The second is called fragment integration, which is a simplified form of data integration. Finally, it has a common editor framework, FrameMaker, which supports hyperlinks, display, and the editing of a wide variety of software artifacts [28].

It is interesting to analyze why developers of the Desert Environment selected FrameMaker as the development base for the common editor, how it has been extended and how the system is kept open for future integration. The main reasons for the choice of FrameMaker are its cross-platform availability and end-user programmability. FrameMaker is available under Windows, Mac OS, and UNIX. It is end-user programmable through FDK (Frame Developer's Kit). Using the libraries and header files in FDK, developers can directly access all the objects in a Frame session or document. Using FDK, the Desert environment integrates tools, such as FRED (FRameMaker-based program EDiting), FINS (FrameMaker INSets for graphical software engineering tools), FLIP (FrameMaker Support for Literate Programming) and

FOOD (FrameMaker Tools for Object-Oriented Design) as extensions to FrameMaker. FRED interacts with other tools, such as the SAND and SAGE packages by sending and receiving messages. All of these tools can be invoked from FrameMaker. Attempting to keep the environment open, designers left the existing data structure untouched. There are no changes to any data sources of any integrated tool.

Desert developers face the challenges of the limitation of FrameMaker. For example, to use the Desert Environment, a user selects a source file, either C++, C or Java, converts this file into a FrameMake file, and then views or edits this file using FrameMaker. There is a dialog box, as shown in Figure 3.4 for user input. Due to a bug in FrameMaker, this dialog will always preselect "C program" regardless of whether the file was a source or header file and regardless of the actual source language [29].

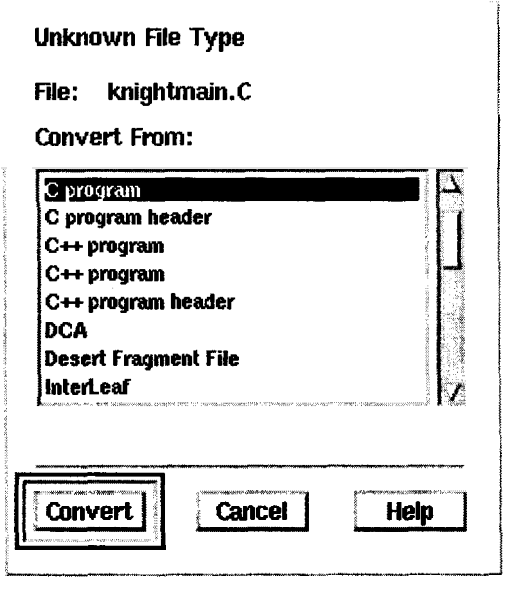



**Figure 3.4: Desert conversion page**

## 3.3 Other related projects

There are several projects focusing on making use of widely available technologies and standards to assist tool development. BOX is one of the most impressive examples [21]. It makes use of standard off-the-shelf browser technology, for example Internet Explorer, to view software engineering documents, specifically UML (Unified Modeling Language) models. Basically, BOX will transfer UML model information in XMI (XML Metadata Interchange) format into VML (Vector Graphic Markup Language). VML is vector graphics rendering technology, which is a predecessor of SVG and refers to SVG used in Web pages [30]. VML can be accessed and viewed directly from later versions of several Internet browsers. The benefits are obvious: group members are not required to install the costly CASE tool and learn how to use the tool. It also takes advantage of the Internet and intranet for data communication and this makes cooperation with group members much easier. It is interesting that the BOX project opted for a new stand-alone tool rather than a plug-in for Rational Rose, the leading UML modeling tool, to transfer the XMI into the HTML and VML form for web publishing, while still using Rational Rose to generate XMI. BOX designers claim that the plug-in is specific to Rational Rose and that the tool independence that BOX gains from using the open XMI standard would be lost [21].

Software Bookshelf is another good example [31]. Software Bookshelf is a software migration assistant environment, which captures, organizes, manages, and delivers comprehensive information about a software system, and provides an integrated suite of code analysis and visualization capabilities intended for software reengineering and migration [31]. Software Bookshelf applies several Web technologies, including

HTTP (Hyper Text Transfer Protocol) for communication, URL (Uniform resource locators) for resource identity and locating, MIME (Multipurpose Internet Mail Extensions) for data and requests/responses association, HTML (Hyper Text Markup Language) for document references, and CGI (Common Gateway Interface) for a Web server to launch and convey requests to arbitrary external programs. Netscape Navigator is chosen as the default Web browser for the client-side user interface of the bookshelf environment. High usability and immediately familiarity to end users is one major concern to make the selection. This off-the-shelf Web browser not only accelerates the tool development process but also lowers the startup cost and training effort of the user in adopting the populated bookshelf environment [31].

## 3.4 Summary

Reusing existing tools, especially some well-developed commercial tools, for tool development is becoming a well-received research practice. Some research projects try to extend an existing tool to another domain, some try to integrate one into a new environment, some try to make use of components. These different tool development approaches provide valuable experience to the ACRE project in decision making, system architecture design and development process analysis and evaluation, e.g. what can good host tools, what can be done with host tools, what benefits we can expect by using it.

# Chapter 4

# Using Excel and PowerPoint as host tool

As discussed in Chapter 1, one of the basic requirements of the host tools is their visualization ability and the ability to browse and edit graphics. Furthermore, it is desirable for the host tool to have a big user base and be end-user programmable. Excel and PowerPoint fulfill both of these requirements, although due to the complexity and level of specialization of these Microsoft applications, further development requires a good understanding of the technologies.

## 4.1 Why select Excel and PowerPoint?

Microsoft Office Suite (or Office for short) is the office productivity software developed by Microsoft Corporation. Office XP is the latest version in the Office family, and all of the discussion here refers to it. Office XP has four different editions, which are Standard, Professional, Standard Educational, and Developer. Word, Excel, Outlook, and PowerPoint are the four basic and tightly related components that come with all four editions. They are used for word processing, spreadsheet manipulation, creating presentation, and email and personal information management.

Reverse engineering is about understanding the legacy systems. In order to "extract and create system abstractions and design information" [14] effectively, visualization of the target system to communicate its complex information becomes the key objective of reverse engineering tools. Both PowerPoint and Excel provide robust support for graphs, and therefore seem to be a competitive candidate for reverse engineering visualization tasks. There are 32 basic shapes, 6 lines, 9 connectors, 32 block arrows and 32 flow charts available in these applications. Users can define different shadow and 3-D style for these graphic artifacts, which can be used to generate complex diagrams. Both RGB and HSL colour models are used in these two tools. These shape and colour options will fulfill the visualization requirements of most reverse engineering tools. Furthermore, Excel has hundreds of embedded mathematical and analytical functions and some data handling ability, such as filtering and validating which can be used to implement certain reverse engineering tasks.

Office is a leading business tool with a large user base. According to Michael Silver, Vice President and Research Director at Gartner Group in Stamford, Connecticut, Microsoft's share of the business market for office software is estimated at just over 90 percent [32]. By June 2001, there were already more than 250 million people who regularly use Office to get their work done [33]. We assume that most of those users are able to work efficiently with at least one of the Microsoft Office Applications. There is also a great chance that they are familiar with other Microsoft Office Applications, since all applications share a similar user interface and similar operation methods for common tasks, such as file operations. Thus, we argue that if Rigi is re-implemented using Office and preserving the Office user interface where possible, it will be easier for the existing

users to learn how to use it. However, Office applications have some major weaknesses, such as inefficiency and perhaps generic functionality, and it is difficult to know exactly how and to what degree the users will tolerate these inconveniences. However, we still believe that those Office users will be more willing to adopt a tool based on Office if it provides comparable functionality as a non-Microsoft-Office based tool because of their previous experience using Office.

Another reason to choose Office is its strong end-user programmability compared to similar productivity tools such as StarOffice, popular business productivity software by Sun, which sets to challenge Office. Each Office Application exposes most functions through programmable objects and also supports the ability to integrate with other applications by using Automation (formerly known as OLE Automation). This object model is well documented in the MSDN (Microsoft Developer Network) library [29] or under "Help with the tool." Also all Microsoft Office Applications use the same development language, Visual Basic for Applications (VBA), which comes with an integrated development environment, the Visual Basic Editor.

Fast development is an important issue as well. There is overlap between reverse engineering tasks and the existing Office functions, which can be reused. Sharing the same programming language (VBA) and the same development environment (VB Editor) among all Microsoft Office Applications, and the widespread use of COM technology also improves the development speed. What you learn while programming one application applies when working with another application [34]. Another benefit is that it is easy to migrate an office-hosted application to a standalone application, due to the architecture and grammar similarity between VBA and VB.

## 4.2 Technology background for Office solutions

An Office solution is an application, which takes advantage of the Office tools and technologies to create customized and integrated solutions on top of the Office suite. A typical Office solution is likely to fall into one of the following broad categories: data-management application, document templates, add-ins, and Web application – either with or without a data-management component [35]. However, there are possibilities to target the Office application for other domains. Furthermore, each of the office applications is better suited to some applications than to others. For example, in order to assist program understanding, developers can customize Office and allow users to use Word to view the source code, PowerPoint to show the program architecture graphically, and Outlook to cooperate with other team members efficiently.

To develop an Office solution can be as simple as writing a VBA (Visual Basic for Application) procedure or as complex as writing a sophisticated banking system. Regardless of what the target application is, it requires a good understanding of the Office application itself and good technology background of Office applications in general. Then the developer can choose which technology he or she will use. Some concepts in these technologies can be confusing; however, these technologies constitute the foundation of Microsoft Windows/Office application development. Without deep knowledge of Office as an expandable development platform, it will be difficult to make use of it effectively and achieve the desired custom solutions. Some terms and

technologies related to the Microsoft Office/Windows development are explained in the following paragraph.

### 4.2.1 COM-based technologies

COM (Component Object Model) is designed as a platform independent, distributed, and an object-oriented system for creating binary software components that can interact [36]. It is the foundation technology for technologies such as DCOM, COM+, Microsoft's OLE, and ActiveX. Since COM underlies many of the software systems developed for the Windows operating system, and is the key technology that makes individual Office applications programmable and makes creating an integrated Office solution possible, good knowledge of how to use COM components and create new ones is necessary for Office programming. For example, a COM add-in for Microsoft Office Applications can be either an ActiveX EXE file or a DLL file that is specially registered for loading by Office applications. Unlike application-specific add-ins, COM add-ins are available to Word, Excel, Access, PowerPoint, Outlook, FrontPage and other Office family applications. The process of creating a COM add-in can be broken down into three general steps: 1. Create a new DLL/EXE COM add-in project; 2. Implement the IDTExtensibility2 interface; and 3. Add settings to the Microsoft Windows registry consisting of a subkey to indicate which applications can host the COM add-in [37].

COM+ is an extension of COM, which adds to COM a new set of system services for application components while they are running, such as notifying them of significant events or ensuring they are authorized to run. In comparison to COM, which provides a set of interfaces allowing clients and servers to communicate within the same computer,

COM+ also works on a network within an enterprise or on other networks besides the public Internet. OLE (Object Linking and Embedding) is a technology that enables an application to create compound documents which contain information from a number of different sources [38]. An ActiveX control is a type of component that uses COM technologies to provide interoperability with other types of COM components and services. It offers enhancements specifically designed to facilitate distribution of components over high-latency networks and to provide integration of controls into Web browsers. An ActiveX control is essentially a simple OLE object that supports the IUnknown interface [39].

Automation (formally known as OLE Automation) is a COM-based technology that enables developers to create and control software objects exposed by any application, DLL, or ActiveX control that supports the appropriate programmatic interface [40]. In particular to Office, developers use this technology to customize applications to automate frequent tasks, and Office uses it to expose almost all functions to the developers through different object models.

### 4.2.2 Microsoft Office Object Model

Object Model consists of objects and becomes the main working environment. Appendix A shows the Shared Components (Microsoft Office XP Object Model, Microsoft Graph Object Model, Microsoft Forms Object Model and Visual Basic Editor 6.0 Object Model), Appendix B shows the Excel Object Model, and Appendix C shows the PowerPoint Object Model [41]

In object models, objects are organized in a tree structure. For example, PowerPoint, as most other Office components, has Application Object as its top-level

object. Developers start automating by using it and go through the right path to reach the desired object. For example, from the Application Object, we can open an existing Presentation Object or create a new presentation. Each Presentation Object contains one or more Slide Objects and each Slide Object can contain Shape Objects that represent text, graphics, tables, and other items found on a slide. Developers work with these objects by using its properties and methods. Some objects, so-called Office Objects, are shared by all Office components. For example, a CommandBars Object can contain a CommandBarControl Object, which provides control of menus, floating bars, etc. Using them, developers can customize pre-built menus and add new menus. In Office XP, there are 23 different CommandBarControl types to choose from.

### 4.2.3 The development languages

Any languages that support automation could be used for Office Application development, including VB (Visual Basic) and VBA (Visual Basic for Applications), Visual C++, etc. VB and VBA are two development languages from Microsoft. VBA is the default language for the Microsoft Office Developer. Since VBA offers a set of programming tools based on the Microsoft VB development system, and a complete integrated development environment that features the same elements familiar to developers using Microsoft VB. These elements include a Project Window, a Properties Window, and debugging tools.

New VB/VBA programmers or Office application developers may get confused by these two languages. Some of the differences between these two approaches are listed below. They are designed for different purposes. VB works either as a stand-alone product or as part of the Visual Studio suite of tools. It is designed as a Rapid

Application Development (RAD) tool to create components and applications. VBA is Microsoft's development technology for customizing rich-client desktop packaged applications and integrating them with existing systems. VBA enables customers to buy COTS components and customize them to meet their specific business processes, rather than build solutions from scratch. For example, all Office suite components are VBA-enabled and provide opportunities for application customization and integration by developers. The main feature that differentiates VBA from VB is the fact that VBA functions as a macro language. In contrast to that VB is designed to create applications and program components; VBA can not create compiled or executable code like VB. VBA codes can only run within their host application. For example, if a VBA project is created to work with Excel (its host application), then it can only be executed within the Excel application. Thus, VBA is basically used to enhance or extend its host application in some way, such as adding new functions to Excel or changing PowerPoint's menu structure by adding new options or removing the ones you do not want to see. So in order to be fluent in using VBA, one must be familiar with the various objects and classes implemented in the host application or applications for which one wants to create macros. For VB, you need to understand the VB's objects to use it effectively. Finally, VB and VBA are development environments that are both similar and different. Most VBA functionalities can be re-implemented by using VB, and some VB functionalities can be accomplished with VBA. However, creating a standalone executable program can not be done with VBA.

## 4.3 Summary

PowerPoint and Excel, two components of Office, are selected as the host tools for their popularity. They have hundreds of millions of users. Another reason is their robust visualization and data management abilities, which are essential reverse engineering requirements. Furthermore, strong `end-user programmability makes it possible to build a reverse engineering tool from them. In addition, good understanding of the COM-related technology, Office Object Model and the development languages becomes a key reason of the successful design and implementation of the new reverse engineering tool.

# Chapter 5

# Potential solutions – high level design

After analyzing requirements, selecting a host tool, and understanding the technology background, the next step is to sketch out the initial design for the application – the technologies to be used, the user interface, and the organization of the system. For the ACRE project, the focus is on how to reuse Rigi, how to make use of Office, and how to make subsystems interoperate with each other. The final design will affect the tool's functionality, development efficiency, deployment options and adoptability.

## 5.1 Preliminaries

Rigi is a successful reverse engineering tool, which has been proven effective in assisting program understanding [42]. However, its GUI's look and feel is perceived as rather crude compared to the state of art [11]. Furthermore, learning the usage of Rigi takes time and effort. Lastly, images cannot be exported (except for taking screenshots) [24]. These weaknesses affect the adoption of Rigi in both academia and industry. If a new tool could be produced with similar functionality to Rigi with a user interface based on Office user interface features, it will be easier for users to learn how to use it and possibly wider adoption could be achieved.

Implementing a reverse engineering tool from scratch is costly in our case. Reusing the existing Rigi implementation as a base can save a large amount of programming, allowing the development effort to be devoted to the higher-level user interface aspects of the application. The first valuable thing we can use from Rigi is its RSF (Rigi Standard Format) output. The Rigi parsers extract the artifacts from the source code required for program understanding and saving those artifacts into an unstructured RSF file. For example, from C code, information about variables, functions, function calls, definitions, etc. can be extracted. Rigiedit loads this information, performs layout operations, and saves the result as structured RSF, which includes spatial information [43]. RSF has also been reused in another reverse engineering tools, Shrimp [25]. The new tool will use structured RSF as input, and then display it and allow further interactive manipulation by the user. Rigiedit, the Rigi graph editor for RSF files includes important functionality, such as its efficient dealing with large amounts of data, its automatic layout algorithms, and its end-user programmable scripting language. These features make it attractive for reusing the existing Rigi implementation – at least for the moment. So the problem is how to leverage Office and Rigi into a new Rigi in a seamless way. The following sections discuss three potential architectures to resolve this problem.

## 5.2 Convert RSF file

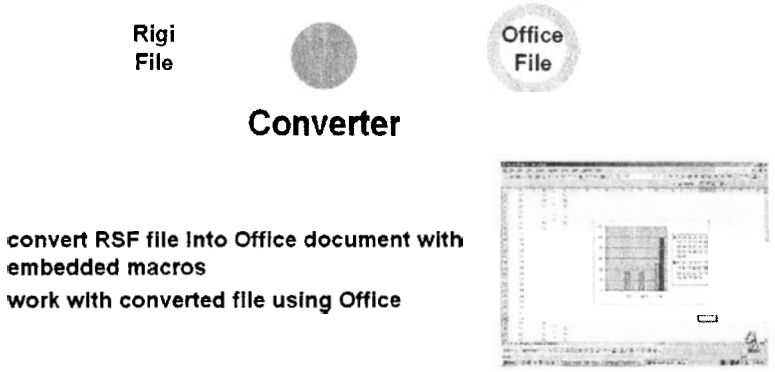



**Figure 5.1: Coverting RSF files into Excel**

As shown in Figure 5.1, the idea of converting the RSF file is to build a separate converter. Rigiedit is not included in the new system and PowerPoint/Excel is only used as the output viewer. The converter will accept a structured RSF file as input and generate a native Office document, which is embedded with some executable macros. These macros can do some non-complex tasks such as compiling statistics on the types appearing in the system. This newly generated document will be viewed by an Office application and the embeded macros will be executed either automatically or manually. To the user, it will appear as opening any other Office supported document.

There are two possible ways to develop the new converter. The first option makes no use of Office COM. An RSF file is translated into an Office binary application file name. The converter will run in an environment without Office installed and, thus, can be cross platform. Developers are able to work on their platform of preference, even UNIX or Linux. They can switch to the Windows platform with Office installed, such as when meeting customers. One problem of this method is the difficulty of getting the file

format specification. Even though both PowerPoint and Excel have not changed the format since Office 97, the format for recording new features has not been published yet. Therefore, unfortunately, this does not seem to be a very promising direction for the development of Office-based application. However, this can be a good architecture candidate if applying another host tool, such as StarOffice. StarOffice uses XML file format and publishes the DTD (Document Type Definition) online [12].

Another option is to have the converter use Office objects to create files through OLE technology. This is the recommended way for Office-based application development by Microsoft. Since it is well documented, less time and effort are needed for development and maintenance. For example, in PowerPoint, saving files in different file formats can be accomplished by setting corresponding parameters using the following command:

PowerPoint.Presentation.SaveAs (Filename As String, [FileFormat], [Password], [WriteResPassword], [ReadOnlyRecommended], [CreateBackup], [AddToMru], [TextCodepage], [TextVisualLayout], [Local]).

This approach has its own drawbacks. The converter will not be platform independent any more, which can be a problem for Office application. Another underlying problem is the security concerns coming with the use of macros. By default when Office is installed, macro security levels of Excel and PowerPoint are set to High, which means disables all unsigned macros automatically. Only after the security setting is changed to Medium or Low, macros have chance to be executed [44].

There is little cognitive load to learn how to use these generated files. If users use Office, they know how to use them naturally. However, in general only fairly simple

functions, such as generation of statistics, which does not require many interactions, can be done by embedded macros. We can develop custom COMs, and invoke these COMs from the macros to get more functions. However, this will raise another problem, which is that components need to be copied to and registered with the MS Office. Comparing the two approaches, described in Section 5.3 and 5.4, it provides less functionality and usability. Moreover, since the output is in binary format, not text based, it is very difficult to extract the info and use for any other purpose. It will be the final working result except for manual changes or the operations done by macros, which are limited.

## 5.3 The application of RigiOfficeHarness

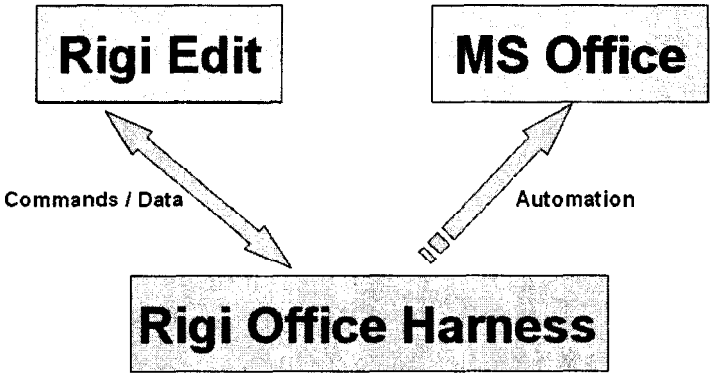



**Figure 5.2: Architecture of the Application of RigiOfficeHarness**

As shown in Figure 5.2, the new tool will be a separate RigiOfficeHarness application and serve as a working center of Rigi and Office. It accepts users' queries and passes them on as Rigi commands, accepts data from Rigi, transforms it and sends the results to Office. Rigi works as a data server, computing reverse engineering tasks and sending results back to RigiOfficeHarness. Comparably, RigiOfficeHarness

translates these results into an Office compatible format and transfers them to Office. Concerning the role of Office, it functions similarly to the converter system described in Section 5.1; Office will not be changed at all and works as a pure viewer.

Compared to the approach described in Section 5.2, the emphasis is on a wide range of functionality possibilities. With this architecture we are freed from the restrictions set by the Office development environment and will have much more freedom to make a powerful new tool. RigiOfficeHarness is able to use any technologies, both from Microsoft or a third party to meet the requirements of a reverse engineering tool. However, some usability is sacrificed for the increased functionality. As you can see, users have heavier cognitive load, since they need to know how to use both Office, and RigiOfficeHarness.

Implementing the two-way communication between Office and RigiOfficeHarness, as well as one-way talk from RigiOfficeHarness to Office is a challenging problem. Since original Rigi design and implementation does not use COM architecture, updating Rigi today to make it support direct data exchange with other COM-based software would be time-consuming and overwhelming task for this project. Rigi's existing data interchange approach has been used. Rigi is designed as a data server based on a loose file relationship. In other words, Rigi can listen to the outside world by checking a special file, which contains RCL commands. So, every second, Rigi reads input file and executes any commands found inside, which are expressed in the Rigi scripting language. For outputting, one method is to write the result of a computation to a file. The commands to write the output file should be part of the input file. Since TCL supports direct COM/DCOM calls, this loose interoperation may be changed after RCL

adopts TCL 8.0 and later [45]. Talking to Office can be done more directly. A reference to the running application will allow the program to exchange data with the Office application through its object model. In VB, "Getobject("Excel.Application")" can get a reference to Excel if it is running. If Excel is not already running, both createObject("Excel.Application") and "New Excel.Application" will start it and return a reference to it.

In summary, the RigiOfficeHarness application is a more open structure than converting RSF files. Since it has fewer design restrictions, it can be implemented using different components from different vendors. Moreover, it takes as much advantage as possible of existing functionality in both Rigi and Office. However, the complex working environment, RigiOfficeHarness plus Office, leads to a more complicated tool, which brings high learning curve and could be less likely to be adopted.

## 5.4 Extended Office

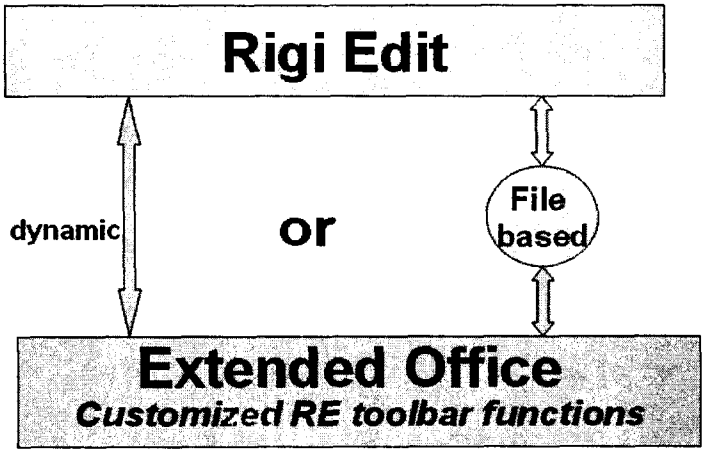



**Figure 5.3: Architecture of Extended Office**

The third possible approach is to extend Office into a reverse engineering tool. As shown in Figure 5.3, only Rigiedit and Office are involved in the new system. The job allocation between Rigi and Office is clear. Rigi works at the back end as the data server, executing most graph related computation. Office works at the front end as the interface for both input and output, and does other computations whose results support program understanding. Office will accept user input, analyze it, and then either manipulate the input itself or transfer it to Rigi. Rigi executes those received commands and sends results back to Office. Office reads the returned information, and then presents it to the users visually.

Here Rigi and Office are still in a loose file-based relationship. Office writes the RCL commands to a file specified by Rigi. Rigi will direct output to files specified by Office. Rigi's output file is text based and includes graph layout information. Text based information promises future reusability and the layout information eases PowerPoint's visualization tasks. Rigi and Office watch these files and execute accordingly when they monitor any changes.

When compared to converting RSF files, the Extended Office will have more flexibility for multi-functionality. It permits enough opportunities to integrate third party products, to implement its own functions, even though it is not the most flexible architecture among the three. When compared to the RigiOfficeHarness application, the Extended Office provides a unified Office user interface to the end users. A Rigi menu or toolbar will be added to the user interface for reverse engineering purposes; all the original functionality of Office will be kept.

## 5.5 Summary

Because of the familiarity with the user interface, importing Office into reverse engineering tool development will lower the learning cognitive load of the new tool for the reverse engineers who are already using Office. Using the existing Rigi implementation will accelerate the development rate and ensure software quality. The high level design determines how to leverage both Rigi and Office, outlines the architecture as well as the relationship between Office and Rigi. The architecture also affects the ease of implementation, potential functionality, deployment options and the adoptability of the new tool. The three architectures discussed here have different strengths. In conclusion, converting RSF files has high portability but limited functionality; the RigiOfficeHarness application has the broadest development space but likely lowest adoptability, and the Extended-Office best leverages both functionality and cognitive support. The Extended Office architecture has been chosen for the final implementation.

# Chapter 6

# Extending Office to support Reverse Engineering Tasks: A case study

Extending Office to support reverse-engineering tasks is based on the assumption that working with Office, a familiar working environment, can lessen the cognitive load of learning a new tool and, thus, will enhance adoption accordingly. From the users' point of view, their working habits in one domain, their business productive work with Office, will be migrated into another domain, namely reverse engineering. The developers therefore need to implement the reverse engineering functionality inside Office, a well developed commercial product, in order to benefit from the cognitive support already acquired. In this project, PowerPoint is used to implement a visualization tool similar to Rigiedit and Excel is used to provide a user interface and statistical analysis tool for this visualization system. In the scope of this thesis, only a prototype was implemented and, at the time of writing, there are some interesting observations that might be useful for future developers of adoption-centric reverse engineering tools.

## 6.1 Rigi Interface in Excel

PowerPoint and Excel were compared to decide which should be used as the major user interface for the new Rigi tool. Technically the two are similar. Both of them use VBA as the end-user programming language. VBA includes most functions available in Visual Basic and can automate all operations and functions of Office as macros. Furthermore, both PowerPoint and Excel provide a rich object model that allows users to programmatically access almost all information. It is also possible for the two applications to interact by accessing one another's methods. However, within Excel, XML support is extensive and powerful while PowerPoint does not support XML well. Although the graphic data interchange between different parts of the ACRE project is accomplished with RVG (Rigi Vector Graph), a non-XML format [18]. In the future an XML format, such as GXL (Graph eXchange Language) or SVG, might be used. Therefore Excel, with its superior XML support, was chosen as the main host and user interface implementation as shown in Figure 6.1.

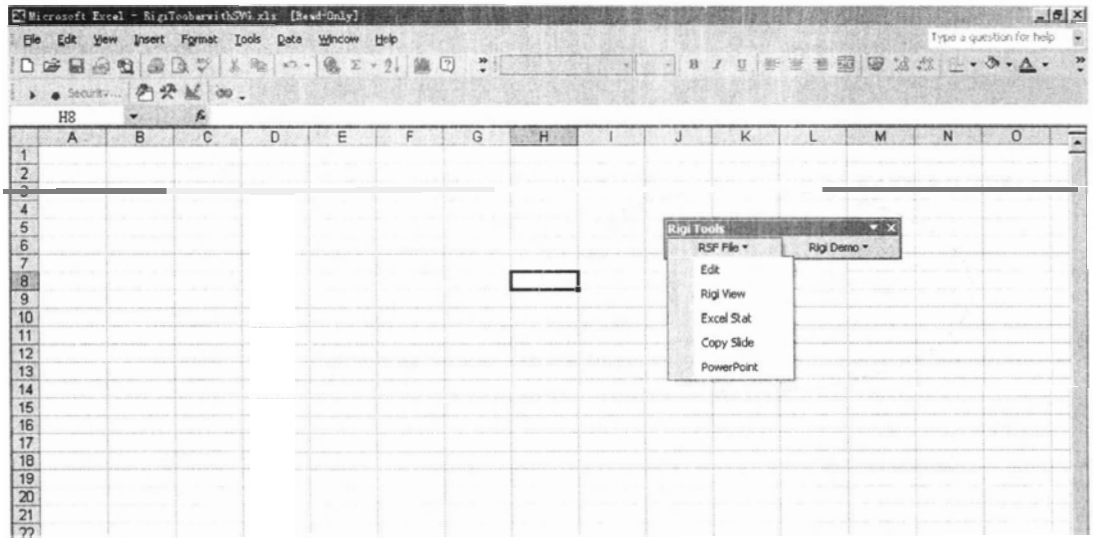



**Figure 6.1: PowerExcelRigi Toolbar in Excel**

All Microsoft Office applications use the same style command bars, and provide a similar method for command bar customization, using Application.Commandbars.Add([Name], [Position], [MenuBar], [Temporary]) As CommandBar. Three different CommandBar objects, toolbars, menu bars and pop-up menus, can be created by setting the [Position] parameter to one of the following values: msoBarFloating, msoBarMenuBar, and msoBarPopup. Pop-up menus may be displayed in any of the three different ways: as menus that drop down from menu bars, as submenus that cascade off menu commands, and as shortcut menus. Shortcut menus (also called "right-click menus") are menus that appear when the user right-clicks something.

All Microsoft Office applications use toolbars to group buttons, which are designed for related tasks, such as Forms, charts, etc. Toolbars can be shown and hidden as needed. A visible toolbar can be docked to a fixed position or be floating on the desktop. In order to be consistent with this user interface style, a floating PowerExcelRigi Toolbar was created for the final implementation rather than the original Rigi submenu, as shown in Figure 6.2. Appendix E lists the sample code for the implementation of the PowerExcelRigi toolbar. This code can be used to achieve a similar effect in other Microsoft Office applications, such as PowerPoint or Word, with few changes.

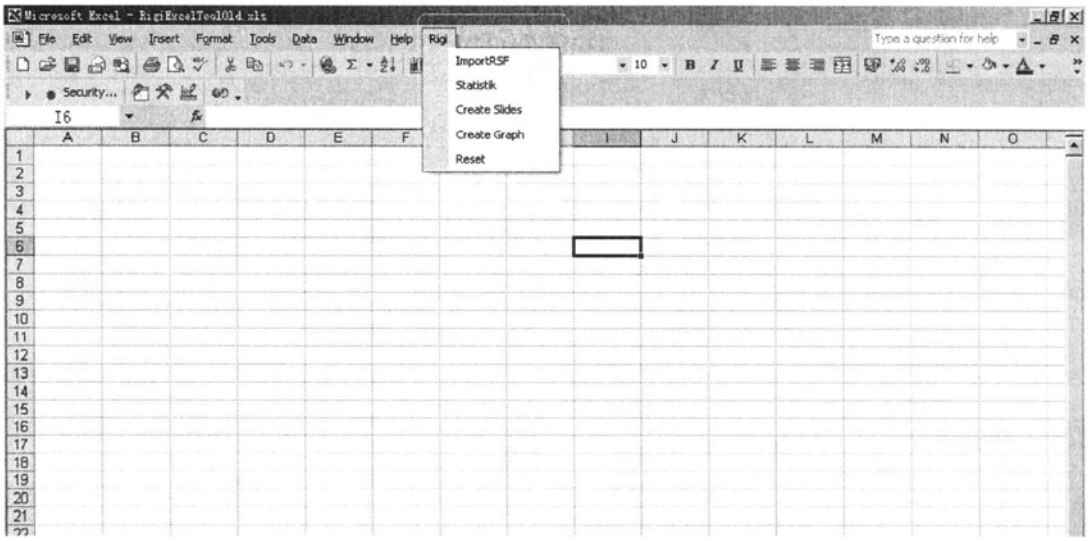



**Figure 6.2: Original Design of Rigi Menu**

Each Microsoft Office application stores its customized Command Bar information in its specified location. Concerning the specialties of a custom toolbar in Excel, it may be saved in two different places. One is the workbook that the toolbar is attached to; the other is the default Excel workspace. The default workspace CommandBars are saved in a file called *Excel.xlb*, which is stored in the *C:\Windows\Profiles\UserName\Application Data\Microsoft\Excel* subfolder if user profiles have been set up for multiple users or in the *C:\windows\Application Data\Microsoft\Excel* subfolder otherwise. When a workbook is opened, and if this workbook contains custom toolbars, which do not already exist in the workspace, the toolbars will be copied to the workspace. After you close the workbook with the toolbar, the custom toolbar still appears on the screen as part of the Excel workspace and is not deleted automatically.

Note that during development, even after you modify the workbook (the working on document) copy, the later version will not be copied to and does not overwrite the old workspace copy. Furthermore, the workspace copy is the default one and the only one that will be displayed when there is more than one copy of a given custom toolbar with the same name. Also when trying to add a toolbar programmatically, a run-time exception has to be caught in case that the toolbar's name has already been used. Thus, in order to update a toolbar, it is necessary to delete the workspace copy and then add the newer workbook version again.

## 6.2 Tool interoperation

The PowerExcelRigi integrates Office components, Excel and PowerPoint, and Rigi together, so as to provide users with some selected reverse engineering operations. The PowerExcelRigi ToolBar is a floating toolbar which includes several buttons, each providing a unique function. By using these functional buttons, users are able to invoke methods in Excel, PowerPoint and Rigi. For example, selecting the *Statistics...* menu item invokes statistical analysis routine of the preloaded structured RSF file and shows the result with a column diagram. This is implemented using an embedded Excel formula. The generated column diagram can be copied to a PowerPoint slide easily using a reference to PowerPoint. Appendix D contains sample code for getting a reference to Excel. Integrating Rigi was done by starting a new instance of Rigi from the PowerExcelRigi toolbar and communicating with it through a loose file-based relationship. Sample code for starting a non-office application is included in Appendix F.

Rigi and Office are two finished products designed for different purposes. Rigi is a visualization tool to facilitate program understanding whereas Office is for more general business tasks. We decided to use a loose file-based information exchange method which can be implemented with little difficulties. According to Rigi's ring architecture [13], as depicted in Figure 6.3, the Rigi kernel works as an RCL command interpreter to extract, select, measure, organize, and represent the graphic artifacts in response to RCL commands. This architecture gives Rigi its powerful end-user programmability. If a user runs Rigi with the flags of "-i startup.rcl -poll -nowb", Rigi will execute startup.rcl [Appendix G], hide the workbench window and only leave the graph window open, with a polling process active in the background. The polling process checks the "/tmp/rigi/input" file every second and executes any commands found, which must follow the RCL syntax. PowerExcelRigi starts the Rigi Ray Tracer Demo simply by writing "menu_demo_db ray c" into the input file.

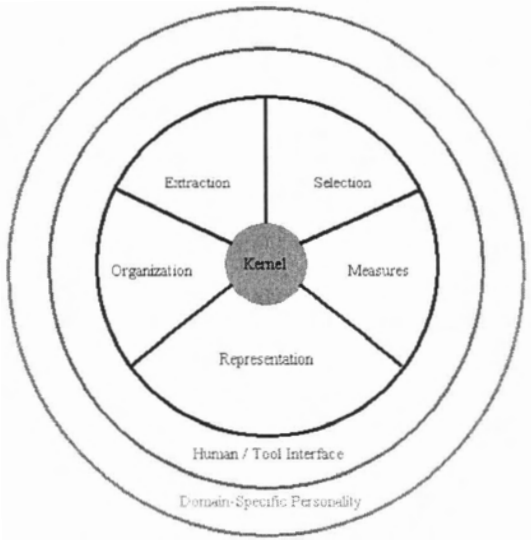



**Figure 6.3: The programmable Rigiedit's ring architecture**

Rigi's regular output is a graph diagram window containing coloured nodes and arcs. In order to transmit the output efficiently to PowerExcelRigi, the RVG format is used. RVG is a text-based format defined by Holger Kienle, University of Victoria [24]. RVG includes information about Node Type (id, name, colour, and filter), Arc Type (id, name, colour, and filter), Node (id, name, position, file name, line number, and connected lines), and Arc (id, source node, destination node, and type). Appendix H shows an RVG file for the tree diagram shown in Figure 6.4.

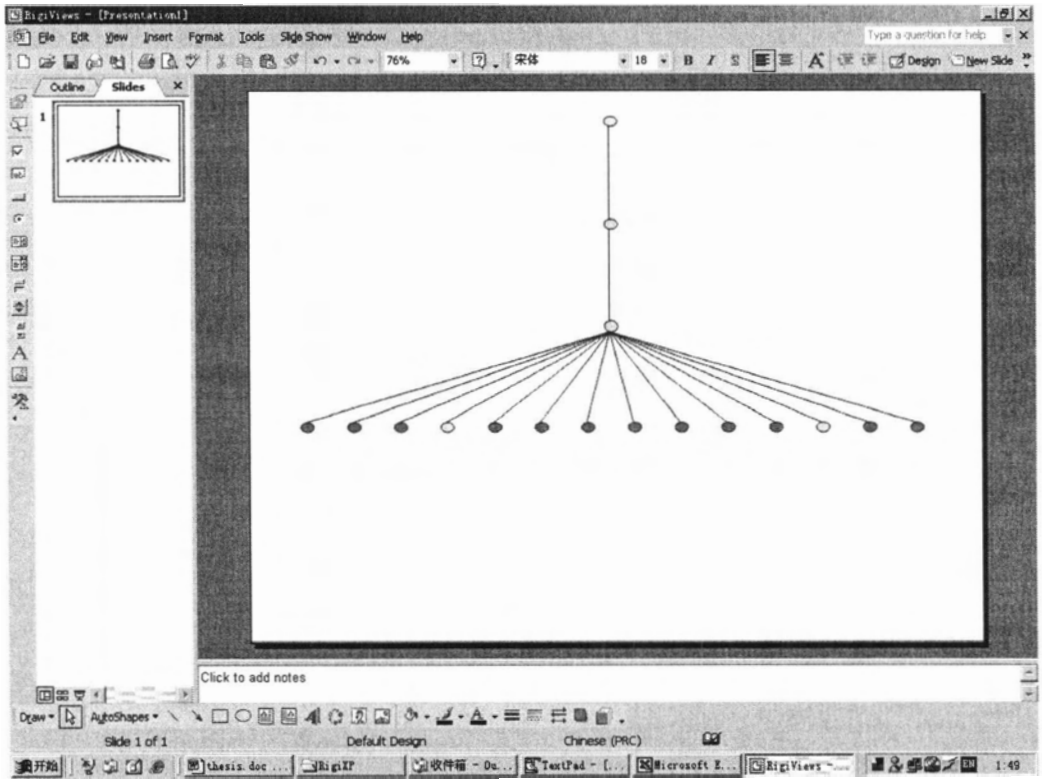



**Figure 6.4: PowerPoint view of RVG**

To summarize the process, PowerExcelRigi translates the user queries into the RCL commands and sends them to Rigi where they are processed and a graph view is

produced. Rigi then sends a description of this graph back in the form of an RVG file, which is visualized by PowerExcelRigi and displayed as a PowerPoint slide.

## 6.3 Implementation issues

Task allocation is required for a task that can be done with either Rigi or Office. For example, Rigi can scale the graph by fitting the current window, by selection or by certain factor. PowerPoint is also able to scale a graph by grouping selected objects and resizing the group region. PowerPoint is selected for this job efficiency, easy coding and maintenance reasons.

Rather then creating new CommandBars, overriding the built-in menus or commands can be a better solution in some cases. For example, there is a LoadRSF button in the PowerExcelRigi Toolbar which, when clicked, pops up a standard Open File dialog window, allowing the user to select an RSF file (a file with extension .RSF). The selected RSF file will be loaded into the current Excel worksheet, giving the user a chance to interact with the data directly and get a general idea about the size of the system. However Excel's standard File/Open menu could also be changed to support this function. Thus, users can load an RSF file in the same way as they open any other Excel file. To the user, it is the most simple and natural way when using Office. To implement it, the built-in menu command needs to be overridden, which can be done by changing the menu's action. When the menu or button is clicked, a new routine, instead of the built-in one, is called.

One way to override menu commands is to create a subroutine with the same name as the original one. Unfortunately, Microsoft does not disclose the subroutine

names. Word is the only application in Office suite that provides users with a direct way to get the subroutine names for menus, toolbars and keyboard commands [46]. For other Office application, sometimes developers may be able to work around this problem. For example, in Excel, subroutine names are often a combination of the menu and submenu command names. For example, to write a custom font dialog box instead of the built-in Font dialog box, we can create a subroutine called FormatFont (Format + Font) and place the custom dialog box inside the subroutine. If this fails, a custom subroutine can be created, using any name. However, this subroutine will not yet be executed from the menu. Another subroutine will need to be created to specify the menu command to be overridden. In the second subroutine, use the onAction property of the object of CommandBarControl and specify the name of the first subroutine created.

Familiarity with Excel and PowerPoint can sometimes dramatically simplify the implementation of new features. For example, in Rigi the graphic model, directed arcs are used to represent the relations between different nodes. When a node is moved, both outgoing and incoming arcs should move accordingly. In PowerPoint, if an arrow line is used to represent those incoming and outgoing arcs, maintaining the connection between nodes and arcs will be complex, probably involving complex data structures, algorithms and lots of computing. However, if a directed connector is used instead of an arrow line, this association is readily supported, and the only change in code is to update ".AddLine" with ".AddConnector". In this way, time and effort spent in learning the host tool will pay off in the long run.

## 6.4 Distribution of Office applications

To distribute this PowerExcelRigi tool or any other Office application project, we have two options. The first is to send out the workbook with the project directly to the users, which means that users have the access to the source code. Users are then able to reuse the code or personalize the solution. However the security may be a concern if the end users have full control of the source code. The second option is distributing an Excel add-in, which is a standard Windows DLL that can provide user-defined functions to Excel. It is very similar to a PowerPoint add-in or a Word global template and is able to implement and export specific methods to Excel, where these methods will operate just like those that are built into the product. There are two ways to load those user-defined methods. A user can load an add-in file with extension .xla or .xll, into Excel manually by selecting the "add-in" item from the menu "Tools" then browsing the directory structure to locate the add-in file. A user can also program to load an add-in by the Add method of the collection of "AddIns" and setting the Installed property of the corresponding AddIn object to True. Another way is to save the add-in file to the C:\Windows\Application Data\Microsoft\Excel\XLStart subfolder, or if the system is using user profiles, to the C:\Documents and Settings\UserName\Application Data\Microsoft\Excel\XLStart subfolder. This add-in is then loaded automatically whenever Excel starts. The RigiToolbar is implemented using macros saved in XLL format. Both methods have been implemented for experimenting purpose.

## 6.5 Development effort

The main developer the PowerExcelRigi is a computer science graduate student, who has about 3 years programming experience. All of the following time is estimated based on her working load. As PowerExcelRigi is part of the ACRE project, lots of help from the teammates is not accounted because of the uncertainty.

Once the requirement analysis is done, tool selection needs brainstorm the potentials, study official documentations about those potential tools' functionality and extensibility, learn from related projects, and then analyze the feasibility. About 10 working days have been spent on it. The architecture design and the analysis of these designs took around 20 working days. The implementation took about three months. The actually programming time would vary dramatically according to developer's familiarity of the host tool and related technology. A much faster development speed can be expected as knowledge of the host tools is accumulated.

## 6.6 Summary

This chapter described some implementation details about the PowerExcelRigi tool. Excel is selected as the main user interface because of its better XML support. The PowerExcelRigi floating bar is added for reverse engineering tasks. To interoperate, the Rigi polling process is turned on to detect the input file and Excel checks Rigi's output file regularly. RCL is used to execute Rigi commands and RVG format is used to record visualization results. Issues about task allocation, overriding built-in commands and Office application distribution have been discussed as well.

# Chapter 7

# Evaluation

In this chapter, PowerExcelRigi is compared to Rigi, with respect to both functional and non-functional issues. Lotus Notes, another selected host tool of the ACRE project, is also used for comparison purposes. Benefits and lessons learned from the development process are also discussed.

## 7.1 Comparison with Rigi

As depicted in Figure 7.1, Rigi provides a menu-based graphic user interface, which includes three parts. The first part includes a customizable menu, buttons, the RCL input textbox and a textbox for running results. The other two parts are floating windows, which look similar to a standard Windows command window. One is for the output of Rigi's system information, such as configuration file and error messages. All other windows are a group of graph diagram windows, which represent the main output of Rigi – visualized program information. After almost 10 years, this user interface is out of date and does not compete with modern user interfaces such as Office. The multiple output windows are not sorted or grouped in any way, and titles of windows on the windows toolbar may be half hidden which makes them unreadable. Thus it is hard to

locate a wanted window, which may be a problem when the number of windows increases.

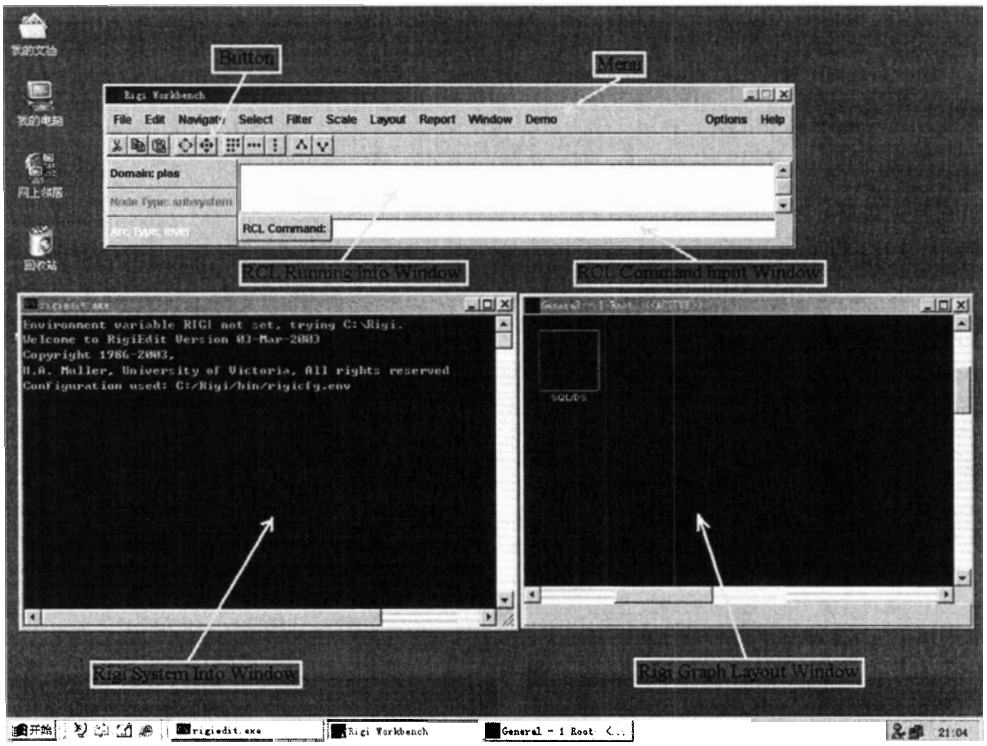



**Figure 7.1: Rigi's Default User Interface**

Compared to Rigi, PowerExcelRigi has a more elegant and modern user interface. It is seamlessly integrated with PowerPoint and Excel using a typical floating toolbar, as shown in Figure 6.1. Furthermore, the output uses normal Office documents, including PowerPoint Presentation and Excel Workbook. Multiple graph output is kept in separate slides, which can be easily located through a preview window, or the slide's name. This promises to be a friendlier interface, especially to experienced Office users.

Rigi has better performance then PowerExcelRigi for many operations. For example, Rigi is more efficient in loading graphs. Figure 7.2 is a software structure

graph which contains 932 nodes and 2395 arcs. It takes PowerPoint about three minutes to load this graph while Rigi only takes seconds on a PIII 733 platform. When working with large programs which may have multi-million lines of code, the waiting time becomes unbearable. Scaling is faster in Rigi as well. Thus, PowerPoint works better with little data as in high-level architecture diagrams.

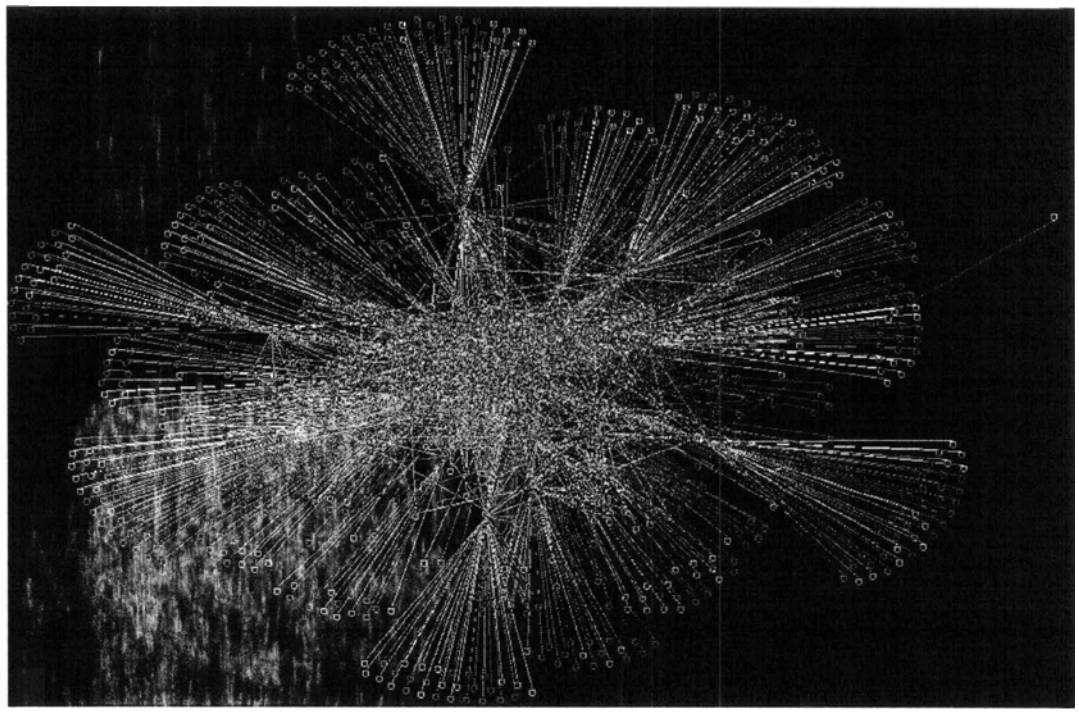


**Figure 7.2: Software structure graph**

## 7.2 Comparison with Lotus Notes as a host tool

Lotus Notes/Domino is one of several optional host tools. It is a popular groupware product that is used by many organizations [47]. It is designed for team members to share information with their team mates. RENotes is built on top of it as a

reverse engineering tool [22]. The implemented functions of RENotes and PowerExcelRigi so far are similar. Lotus Notes as a host tool has its own pros and cons.

Comparable to Office, Lotus Notes is also customizable. It uses Domino Object Model to expose its internal states, which has been implemented for several languages including Lotus Script, VB, Java Script and Java. Thus, Lotus Notes has strong end-user programmability, which makes it possible to become the host application for tens of customized applications [22]. One benefit of Lotus Notes is that all of the information is represented as documents and saved in a database by default. RENotes follows this rule and saves all nodes and arcs of a diagram as database records, which makes exporting information easier and supporting interoperability better. Another strength of Lotus Notes is its client/server working model. Domino acts as the server, providing database access; Lotus Notes hosts the client application that accesses the database. Reverse engineering is a team work: its process involves other team members' devotion, and its output needs to be shared within group or even among different groups. Lotus Notes client/server model eases the implementation of this information sharing requirement.

Unlike Office, Lotus Notes has a much smaller user base. Lotus Notes users congregate at big companies, including IBM, Statoil, etc. Since the target user of RENotes is subset of Lotus Notes' existing users, it has fewer potential users. Another drawback of Lotus Notes as a reverse engineering tool is its lack of embedded graph editing application. Since one key task for a reverse engineering tool is to visualize code, lack of a pre-built graph editor requires much more implementation efforts. RENotes provides a visual representation of the nodes and all relationships between them by using a custom built graph browser, as shown in Figure 7.3 [22]. However, cognitive support

from the familiar UI of the host may be sacrificed when reverse engineers work intensively with the graph browser.

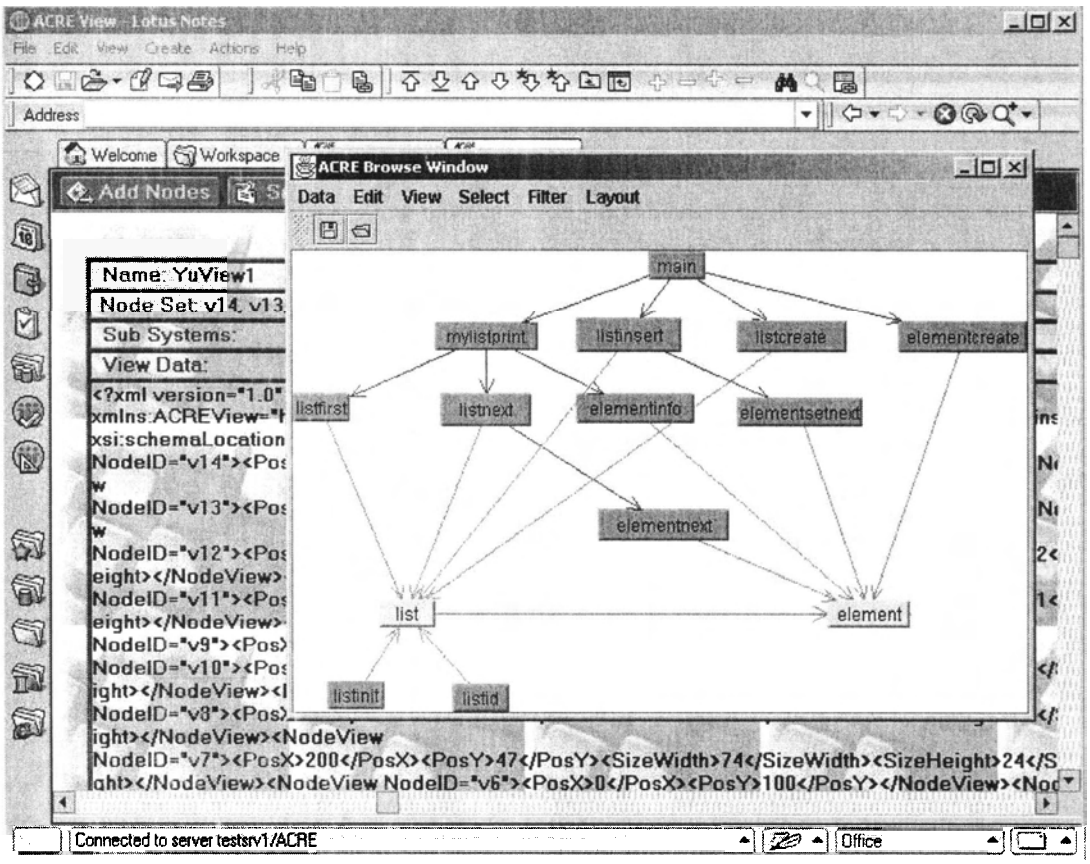



**Figure 7.3: Visualization of an RENotes database**

## 7.3 Architecture reuse

The previous designed architecture, Extended Office, has been reused in the following project, REVisio. REVisio is a reverse engineering tool built on top of Microsoft Visio [48]. Visio is a diagramming program that can help you create business and technical diagrams [49]. It is a qualified host tool competitive that meets the selection criteria. Since Visio shares the similar development features with Office as a

host tool, REVisio applies the same architecture as Extended Office, by customizing the Visio and taking use of Rigi. At the same time, developers benefit from the development experience of PowerExcelRigi to get started and the development time has been reduced by about 50%,

## 7.4 Development experience

Compared to traditional tool development activity, Extended Office benefits from accelerated developing rate. Although as stated in Section 6.5, at the initial stage, there is a steep learning curve with respect to the host tool technology, Microsoft technology and Microsoft Office developer, effort and time can be saved by reusing existing components. There are several examples: a typical one is the CommonDialog of *.ShowOpen;* one line of code provides a standard dialog to open a file, which may need much more code in some languages. Another example is using a connector other then arrow line in PowerPoint to represent relationships between different nodes. Keeping two nodes connected while one of them is selected and dragged around requires complex data structures and algorithms, which brings significant difficulties to both development and maintenance. Furthermore, developers are able to focus on functional features rather then on tedious GUI design.

Extended Office is a cost saving approach, not only with respect to the initial development but also the ongoing support from the Office vendor, Microsoft. Better quality is also expected, not only because of the reused components have been developed and tested by field experts, but also because most bugs and design efficiency have been uncovered by previous users [12].

One main risk is that maintenance and evolution of PowerExcelRigi is not under the full control of its developers, since they have little or no effect on Office's development. When a new version of Microsoft Office is released, PowerExcelRigi has too be updated according to the updated object model or other related changes. In some extreme cases, the whole project may need to be re-implemented or even re-designed to adapt to changes. Another drawback is the lack of informatoin. Even though there is a comprehensive MSDN library and on content help, sometimes it is hard to find information needed or it is just not exposed to the end user at all. For example, the built-in menu commands are not published, which makes override the embedded behavior of menus and buttons difficult. Sometimes the high learning curve of the required skills can be overwhelming. In the worse case, this time and effort consuming process may only prove the selected host tool is not a compatible one.

## 7.5 Summary

Compared to Rigi, PowerExcelRigi's user interface appears to be more friendly and elegant. However, it is less efficient in some graphic operations, which determines that it is better to handle small amount of data and high level design recovery. As a host tool, compared to Office, the Lotus Notes database-based data management scheme makes importing and exporting data easier, and the client-server based working environment helps to share results among group members. However, it is mainly used in big organizations to improve team work efficiency [47] and has relatively small user base, this limits its potential adoption among reverse engineers. Furthermore, its lack of built-in graphic component sacrifices its usability as well. Compared to the traditional

tool development process, extending Office benefits include accelerated development speed, cost saving and high quality. However, it suffers from higher maintenance and evolution risks.

# Chapter 8

# Conclusions

## 8.1 Summary

Major themes and goals of this thesis include:

- designing architecture to integrate existing reverse engineering tools with Microsoft Office;
- analyzing host tool selection criteria by emphasizing the host tool's functionality, user base, end-user programmability and function overlaps; and
- helping Office Application developers get started by discussing implementation issues.

Adoption difficulty is a key problem for software engineering tools. PowerExcelRigi prototypes a tool development approach to resolve this problem by decreasing the cognitive load to learn the new tool. PowerExcelRigi integrates host tools, Excel and PowerPoint, and Rigi uses Excel as its main working environment, which provides Office users a familiar user interface. Problems and solutions, including host tool selection criteria, architecture design, and implementation issues, such as user interface decision and task allocation, have been fully discussed in this thesis. The PowerExcelRigi tool is compared with Rigi; Office as a host tool is compared to Lotus

Notes; and some general development issues are compared with the traditional tool development approach.

## 8.2 Contribution

In this project, we investigated and implemented a novel approach to build reverse engineering tools by integrating the existing reverse engineering system, Rigi, with Microsoft Office. The ACRE project's background (Chapter 2) on reverse engineering tools, cognitive support, COTS-based software development, and related tool development work (Chapter 3) discussed in this thesis should provide a good resource and starting point for people to work on ACRE-related projects.

The concept of a host tool and the criteria to select a host tool (Chapter 4), which are a host tool's functionality, user base, end-user programmability and overlaps between a host tool's functions and a new tool's requirement, should decrease future development risks. The summary of the Office application development technology hopefully clarifies some development confusion as well.

Designers are able to learn from the three potential architecture designs of integrating Rigi and Office (Chapter 5). The advantages and disadvantages of each design help to blueprint good high level designs for other similar projects. The ad-hoc file-based loose cooperation can be a model for integration of unrelated final products.

Some thoughts on the implementation issues of the PowerExcelRigi prototype (Chapter 6), such as detailed user interface design decision and task allocation between Office and Rigi, are also valuable for future development work.

## 8.3 Future work

### 8.3.1 Improve reverse engineering functionality

We need to extend the visualization ability of PowerExcelRigi. PowerExcelRigi is only a prototype now with limited reverse engineering capability. Since, eventually, Rigiedit will be abandoned, its function needs to be re-implemented in PowerExcelRigi step by step. These functions include some simple ones, such as to select forward tree, as well as complex operations, such as to generate Sugiyama layout. Furthermore, design revision needs to be done. After we get experience with Office Application Development, some original design choices may turn out to be unreasonable and need be changed. Some features can also be added; for example, a simple version control system can be implemented by customizing the "Revisions Pane," which keeps a record of the change history of slides and presentations.

We only apply PowerPoint and Excel in PowerExcelRigi so far. This is not enough. The other two basic Office components, Word and Access, could also be used to address practical issues underlying the reverse engineering tool. Word will be able to used for the source code and note browser. So far the node and arc information are not saved except as binary PowerPoint presentation or Excel workbook after PowerExcelRigi accepts the RSF/RVG data. Access can be used to save node/arc information for future use.

### 8.3.2 REOffice: an integrated reverse engineering environment

We can take advantage of the tight interoperability among Office applications to build an integrated reverse engineering environment: ReOffice. A state can be defined to show whether an application is working as part of the reverse engineering environment or not. Special copy/paste behavior will be one important feature of this environment. If an application is under the reverse engineering state, the data pasted to it will be represented using its own visualization strategy. For example, copy .rsf source from Excel and paste it into PowerPoint will end up with nodes and arcs in PowerPoint. Data sensitive will be another characteristic feature. Data changes in one application will invoke related data updates in another application. For example, collapsing nodes in PowerPoint will insert the information about the newly collapsed node into Excel. Concerning the user interface, rather than selecting one application as the system main entry, such as Excel in ExcelRigi, each Office application will have its own reverse engineering toolbar which will be visible only when the application is running in the reverse engineering state. This integrated reverse engineering environment will be able to fulfill most reverse engineering documentation requirements.

### 8.3.3 SVG visualization

SVG is the description of an image as an application of XML. Except for separate SVG viewers, software including PowerPoint, Internet Explorer, and Adobe Illustrator can display the image using the information provided in SVG format. SVG as a visualization standard has the following advantages. It can be visualized by different products, is an

open-source implementation, and is capable of cross-platform usage. Its XML-based format makes it easy to be integrated and hyperlinked with other XML documents. Furthermore, SVG documents can be made interactive with embedded scripting (e.g., ECMAScript). Thus, SVG documents can be sensitive to environment changes, which make them become live documents [50]. Thus, transmitting a PowerPoint presentation into SVG will broaden the application range of the reverse engineer's working results.

## 8.4 Summary

Some investigation has been done in the ACRE project by choosing appropriate host tool to build a reverse engineering tool. Especial focus has been put on integrating Excel and PowerPoint with Rigi. There are still quite a few of work left for the future development, such as using more host tools to implement other reverse engineering functionality and SVG visualization.

# References

[1] K.H. Bennett, V. T. Rajlich. "Software maintenance and evolution: a roadmap," In Proceedings of the 22nd International Conference on Software Engineering, pages 3-22, Limerick, Ireland, June 2000. ACM Press.

[2] E.B. Swanson, and M. Cynthia, and C.M. Beath. "Maintaining information systems in organizations," 1989. 255 pages. John Wiley & Sons.

[3] H.A. Müller, J.H. Jahnke, D.B. Smith, M.-A. Storey, and S.R. Tilley, K. Wong. "Reverse engineering: a roadmap," In Proceedings of the 22nd International Conference on Software Engineering, pages 47-60, Limerick, Ireland, June 2000. ACM Press.

[4] S. Rugaber. "Program understanding," A. Kent and J. G. Williams, editors, Encyclopedia of Computer Science and Technology, pages 341-368, Georgia Institute of Technology, 1995.

[5] I. Zayour and T. C. Lethbridge. "Adoption of Reverse Engineering Tools: a Cognitive Perspective and Methodology," In Proceedings of 9th International Workshop on Program Comprehension, pages.245-255, Toronto, Ontario, Canada, May 2001. IEEE Computer Society Press.

[6] Imagix 4D. Online at: http://www.imagix.com/index.html

[7] M.-A. Storey; and H.A. Müller. "Manipulating and Documenting Software Structures using SHriMP Views," In Proceedings of IEEE International Conference on Software Maintenance, pages 275-284, Opio, France, October 1995. IEEE Computer Society Press.

[8] SHriMP Views. Online at: http://shrimp.cs.uvic.ca/

[9] S. Tilley, S. Huang and T. Payne. "On the Challenges of Adopting ROTS Software," In Special Report (CMU/SEI-2003-SR-004) of the 3rd International Workshop on Adoption-Centric Software Engineering, 25th International Conference on Software Engineering, pages 30-35, Portland, Oregon, USA, May 2003. Online at: http://www.sei.cmu.edu/pub/documents/03.reports/pdf/03sr004.pdf

[10] J. Singer, and T.C. Lethbridge. "Studying Work Practices to Assist Tool Design in Software Engineering," In Proceedings of International Workshop on Program Comprehension, pages 173-179, Ischia, Italy, June 1998. IEEE Computer Society Press.

[11] H.A. Müller, M.-A. Storey and K. Wong. "Leveraging Cognitive Support and Modern Platforms for Adoption-Centric Reverse Engineering (ACRE)," CSER Research Proposal, Nov. 2001. Online at: www.acse.cs.uvic.ca

[12] W. M. Gentleman. "Effective use of COTS (commercial-of-the-shelf) software components in long lived systems (tutorial)," In Proceedings of the 19th international conference on software engineering, pages 636-636, Boston, Massachusetts, USA, May 1997. ACM Press

[13] S.R. Tilley, K. Wong, M.-A.D. Storey, and H.A. Müller. "Programmable Reverse Engineering," International Journal of Software Engineering and Knowledge Engineering, Volume 4, Issue 4, pages 501-520, December 1994. World Scientific Publishing Company.

[14] E.J. Chikofsky, and J.H. II. Cross. "Reverse engineering and design recovery: taxonomy," IEEE Software, Volume 7, Issue 1, pages 13 -17, January 1990. IEEE Computer Society Press.

[15] H.A. Müller, K. Wong, and S.R. Tilley. "Understanding Software Systems Using Reverse Engineering Technology," V.S. Alagar and R. Missaoui, editors, "Object-Oriented Technology for Database and Software Systems," pages 240-252, 1995. World Scientific Publishing Company.

[16] T. A. Standish. An essay on software reuse. IEEE Transactions on Software Engineering, Volume SE-10, Issue 5, pages 494-497, Sep 1984. IEEE Computer Society Press.

[17] SNiFF+ product. Online at:
http://www.windriver.com/products/sniff_plus/index.html

[18] D. R. Raymond, H. J. Fawcett. "Playing detective with full text searching software," In Proceedings of the 8th annual International Conference on System Documentation, pages 157-166, Little Rock, Arkansas, USA, October/November 1990. ACM Press.

[19] D. Carney. "Assembling Large Systems from COTS Components: Opportunities, Cautions, and Complexities," SEI Monographs on Use of Commercial Software in Government Systems, Carnegie Mellon University Software Engineering Institute, Pittsburgh, USA, June 1997.

[20] Desert Programming Environment Homepage,
http://www.cs.brown.edu/software/desert/

[21] C. Nentwich, W. Emmerich, A. Finkelstein, and A. Zisman. "BOX: Browsing objects in XML," Software Practice and Experience, Volume 30, Issue. 15, pages 1661-1667, 2000. John Wiley & Sons.

[22] J. Ma, H.M. Kienle, P. Kaminski, A. Weber, and M. Litoiu. "Customizing Lotus Notes to Build Software Engineering Tools," In proceedings of CAS Conference (CASCON-2003), pages 276-287, Markham, Ontario, Canada, October 2003. IBM Press.

[23] Scalable Vector Graphics (SVG). Online at: http://www.w3.org/Graphics/SVG/Overview.htm8

[24] H.M. Kienle, A. Weber, and H.A. Müller. "Leveraging SVG in the Rigi Reverse Engineering Tool," In Proceedings of the SVG Open Developers Conference (on CD Rom and Web), Zurich, Switzerland, July 2002

[25] Visual Design Editor Generator, Online at: http://mr.teknowledge.com/daml/default.htm

[26] N. Goldman, and R. Balzer. "The ISI Visual Design Editor Generator," In Proceedings of the 1999 IEEE Symposium on Visual Languages, pages 20-27, Tokyo, Japan, September 1999. IEEE Computer Society Press.

[27] M. Tallis, N.M. Goldman, and R.M. Balzer. "The Briefing Associate: Easing Authors into the Semantic Web," IEEE Intelligent Systems, Volume 17, Issue. 1 (January 2002), pages 26-32. IEEE Educational Activities Department.

[28] S. Reiss. "Simplifying data integration: the design of the Desert software development environment," In Proceedings of the 18th International Conference on Software Engineering, pages 398-407, Berlin, Germany, May 1996. IEEE Computer Society Press.

[29] Getting Started with Desert. Online at: http://www.cs.brown.edu/software/desert/starting.html

[30] B. Mathews, D. Lee, B. Dister, J. Bowler, H. Cooperstein, A. Jindal, T. Nguyen, P. Wu, and T. Sandal. Vector Markup Language (VML) World Wide Web Consortium Note 13-May-1998. Online at: http://www.w3.org/TR/NOTE-VML

[31] P. Finnigan, R. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. Müller, J. Mylopoulos, S. Perelgut, M. Stanley, and K. Wong. "The Software Bookshelf," IBM Systems Journal, Volume 36, Issue 4, pages 564-593, November 1997. IBM Press.

[32] T. Krazit, IDG News Service. "StarOffice Set to Challenge Microsoft's Office," PC World, May 16, 2002. Online at: http://www.pcworld.com/news/article/0,aid,99643,00.asp

[33] Microsoft Press. Online at: http://www.microsoft.com/china/press/2001/0619.asp

[34] D. Shank, M. Roberts, T. Myers. "Microsoft Office 2000/Visual Basic Programmer's Guide," May 1999. 800 pages. Microsoft Press.

[35] MSDN Library. Online at: http://www.msdn.microsoft.com/library/default.asp

[36] Microsoft COM Technologies. Online at: http://www.microsoft.com/com/tech/com.asp

[37] F. C. Rice. "Building a COM Add-in for Microsoft Office XP Using Microsoft Visual Basic 6.0," 41 printed pages, CDROM and Web, June 2002. Online at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnoxpta/html/odc_comaddinvb6.asp

[38] Microsoft COM+ Technologies. Online at: http://www.microsoft.com/com/tech/COMPlus.asp

[39] Microsoft ActiveX Controls. Online at: http://www.microsoft.com/com/tech/ActiveX.asp

[40] L. Turner. "Automating Microsoft Office 97 and Microsoft Office 2000," CDROM and Web, 123 printed pages, March 2000. Online at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dno2kta/html/offaut.asp

[41] "Microsoft Office XP Developer Object Model Guide," CDROM and Web, 2001. Online at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/modcore/html/deovrobjectmodelguide.asp

[42] K. Wong, S. R. Tilley, H. A. Müller, and M.-A. Storey. "Structural Redocumentation: A Case Study," IEEE Software, Volume 11, Issue 6, pages 46-54, January 1995. IEEE Computer Society Press.

[43] K. Wong. Rigi User's Manual, June 1998. Version 5.4.4, 168 pages.

[44] Office Macro Security Setting, Office XP Resource Kit, April, 2001. Online at: http://www.microsoft.com/office/ork/xp/two/admc04.htm

[45] TCL Developer Xchange. Online at: http://www.tcl.tk

[46] L. Wollin. "Overriding Built-in Menus and Commands in Microsoft Word," MSDN Library (CDROM and Web), March 2002. Online at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnword2k2/html/odc_wdoverride.asp

[47] Office XP Developer, "What technology should you use?", CDROM and web, 2001

Online at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/modcore/html/deovrwhichtechnologiesshouldyouuse.asp

[48] Q. Zhu, Y. Chen, P. Kaminski, A. Weber, H.M. Kienle, H. A. Müller. "Leveraging Visio for Adoption-Centric Reverse Engineering Tools," In Proceedings of the 10th Working Conference on Reverse Engineering, pages 207-274, Victoria, British Columbia, Canada, November 2003. IEEE Computer Society Press.

[49] Microsoft Visio. Online at: http://www.microsoft.com/office/preview/visio/default.asp

# Appendix A: Microsoft Office shared component

## 1. Microsoft Office XP Object Model


Source (Type Library)

The Microsoft Office XP object model is provided by MSO.DLL, which is included when you install Office [41].

Notes:

The plus sign (+) indicates that the CommandBarControls collection contains CommandBarControl objects [41].

## 2. Microsoft Graph Object Model


Source (Type Library):

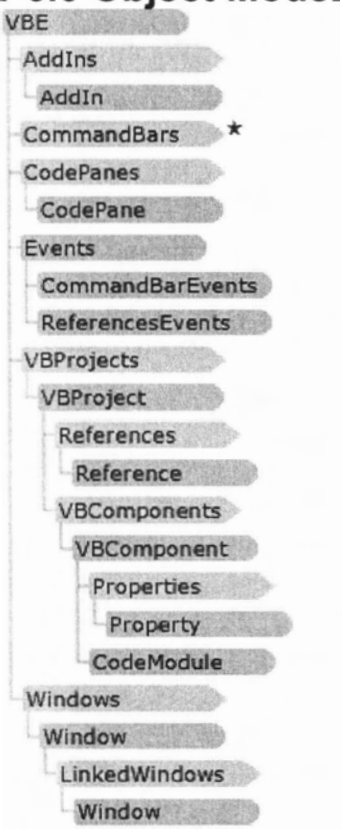
The Microsoft Graph object model is provided by GRAPH.EXE, which is included when you install MS Office XP [41].

# 3. Microsoft Forms Object Model


Source (Type Library):

The Microsoft Forms object model is provided by FM20.DLL, which is included when you install Microsoft Office XP [41].

# 4. Visual Basic Editor 6.0 Object Model

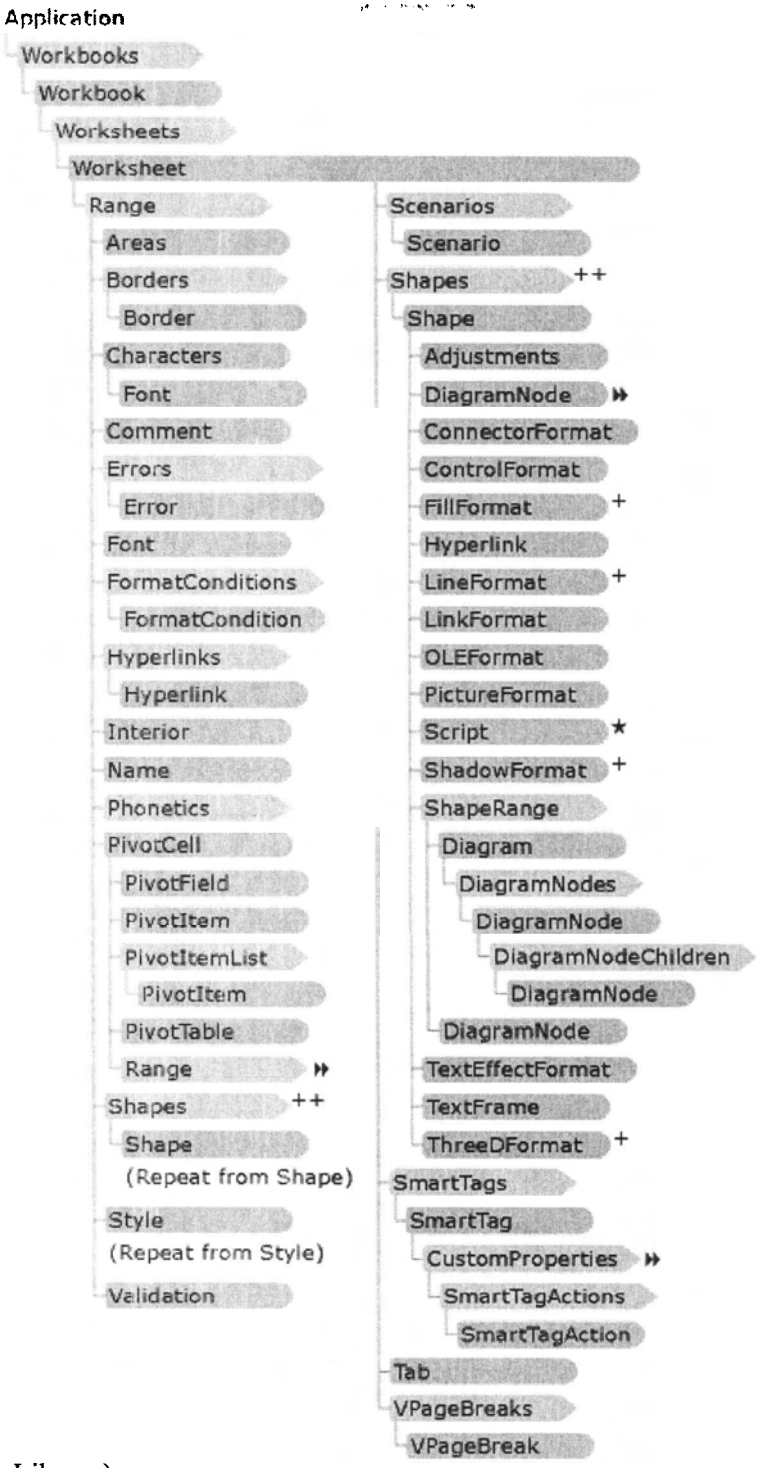



Source (Type Library):

The Microsoft Visual Basic Editor Object model is provided by VBE6EXT.OLB, which is included when you install Microsoft Office XP [41].

# Appendix B: Microsoft Excel Object Model


Application
Workbooks
Workbook
Charts
Chart
PlotArea
Scripts *
SeriesCollections
(Repeat from Series)
Shapes ++
(Repeat from Shape)
Walls
CustomViews
CustomView
DocumentProperties *
HTMLProject *
Names
Name
PivotCaches
PivotCache
PublishObjects
PublishObject
RoutingSlip
Styles
Style
Borders
Border
Font
Interior
VBProject **
Windows
Window
Panes
Pane
WebOptions
Worksheets
Worksheet
AutoFilter
Filters
Filter
Comments
Comment
ChartObjects
ChartObject
Chart
(Repeat from Chart)
PivotLayout
CustomProperties
CustomProperty
HPageBreaks
HPageBreak
Hyperlinks
Hyperlink
Names
Name
OLEObjects
OLEObject
Outline
PageSetup »
PivotTables
CalculateMembers
CalculateMember
PivotTable
CubeFields
CubeField
PivotCache
PivotFields
PivotField
PivotItems
PivotItem
PivotFormulas
PivotFormula
Protection
AllowEditRanges
AllowEditRange
Range
UserAccessList
UserAccess
QueryTables
QueryTable
Parameters
Parameter



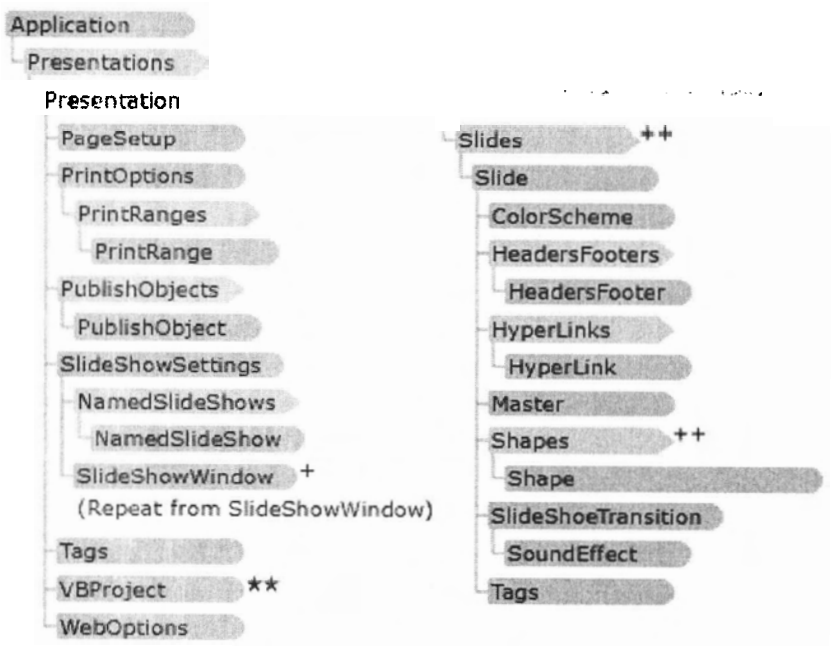

Source (Type Library):

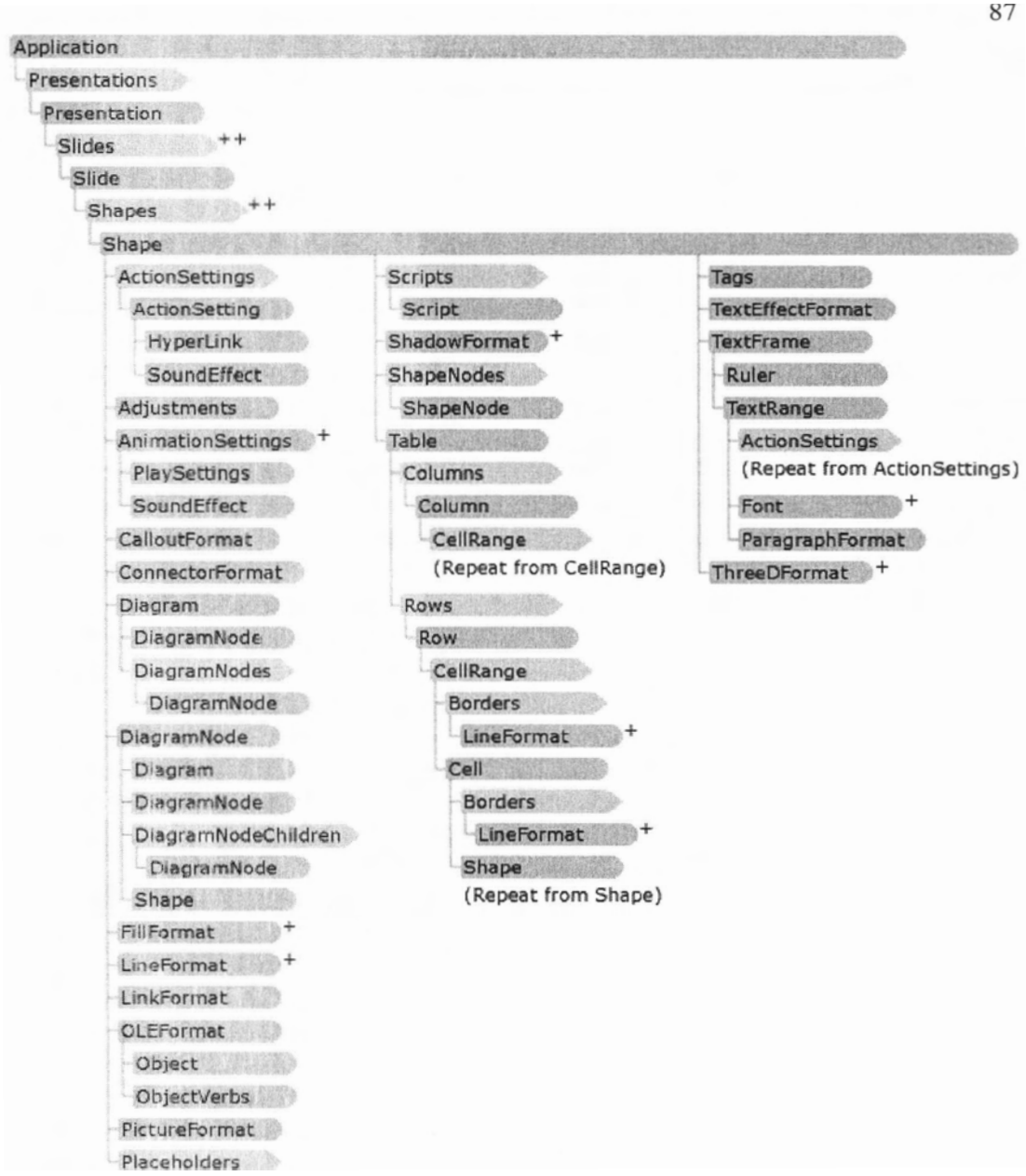
Microsoft Excel object model is provided by EXCEL.EXE [41].

Legend:

- A single plus sign (+) designates objects with accessors to the ColorFormat object.
- A double plus sign (++) indicates that the ShapeRange objects have been omitted from this diagram.
- A double arrow (>>) designates that you can reference the children of this object in another location within this object model [41].

# Appendix C: Microsoft PowerPoint Object Model


Application
Presentations
Presentation
PageSetup
PrintOptions
PrintRanges
PrintRange
PublishObjects
PublishObject
SlideShowSettings
NamedSlideShows
NamedSlideShow
SlideShowWindow
+
(Repeat from SlideShowWindow)
Tags
VBProject
**
WebOptions
Slides
++
Slide
ColorScheme
HeadersFooters
HeadersFooter
HyperLinks
HyperLink
Master
Shapes
++
Shape
SlideShoeTransition
SoundEffect
Tags




Source (Type Library):

The Microsoft PowerPoint object model is provided by MSPPT.OLB, which is included when you install PowerPoint [41].

# Appendix D: VBA code to get a reference of MS Excel

```
Sub GetExcel()
   Dim MyXL As Object   ' Variable to hold reference
                        ' to Microsoft Excel.
   Dim ExcelWasNotRunning As Boolean   ' Flag for final release.

' Test to see if there is a copy of Microsoft Excel already running.
   On Error Resume Next   ' Defer error trapping.
' Getobject function called without the first argument returns a
' reference to an instance of the application. If the application isn't
' running, an error occurs.
   Set MyXL = Getobject(, "Excel.Application")
   If Err.Number <> 0 Then ExcelWasNotRunning = True
   Err.Clear   ' Clear Err object in case error occurred.

' Check for Microsoft Excel. If Microsoft Excel is running,
' enter it into the Running Object table.
   DetectExcel


   If ExcelWasNotRunning = True Then
     MyXL.Application.Quit
   End IF

   Set MyXL = Nothing   ' Release reference to the
                        ' application and spreadsheet.
End Sub
```

# Appendix E: VBA code sample to create a toolbar

```
Public Sub RigiToolBar()
   Dim RigiToolBar As CommandBar
   Dim aPopup As CommandBarPopup
   Dim aButton As CommandBarButton

   'Create a floating toobar, Rigi Tools, through CommandBars collection
   Set RigiToolBar = CommandBars.Add(Name:="Rigi Tools",
Position:=msoBarFloating, _
      MenuBar:=False, Temporary:=False)

   'Create a Popup menu, RSF File, and add it to toolbar "Rigi Tools"
   Set aPopup = RigiToolBar.Controls.Add(Type:=msoControlPopup)
   With aPopup
      .Caption = "RSF File"
      .Width = 110
      'Create a button, Edit, and add it to menu "RSF File"
      Set aButton = .Controls.Add(Type:=msoControlButton)
      With aButton
         .Caption = "Edit"
         .Width = 80
         .OnAction = "actOpenRSF"
      End With
      'Create a button, Rigi View" and add it to menu "RSF File"
      Set aButton = .Controls.Add(Type:=msoControlButton)
      With aButton
         .Caption = "Rigi View"
         .Width = 80
         .OnAction = "actRigi"
      End With
   End With
```

# Appendix F: VBA code sample to run a non-Office application

```
Option Explicit
Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hWnd
As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As
String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long
Declare Sub BringWindowToTop Lib "user" (ByVal hWnd As Integer)
Declare Function SendMessage Lib "user" (ByVal hWnd As Integer, ByVal wMsg As
Integer, ByVal wParam As Integer, lParam As Any) As Long
Declare Sub SetWindowPos Lib "user" (ByVal hWnd%, ByVal hwndAfter%, ByVal x%,
ByVal y%, ByVal cx%, ByVal cy%, ByVal swp%)
Global Const SW_SHOW = 5
Global Const WM_CLOSE = &H10
Global Const HWND_TOP = 0
Global Const HWND_BOTTOM = 1
Global Const HWND_TOPMOST = -1
Global Const HWND_NOTOPMOST = -2
Global Const GWL_ID = (-12)
Global Const GW_HWNDNEXT = 2
Global Const GW_CHILD = 5
Global Const FWP_STARTSWITH = 0
Global Const FWP_CONTAINS = 1
Function Execute_Program(ByVal strFilePath As String, ByVal strParms As String,
ByVal strDir As String) As Integer
   'run program
   Dim hwndProgram As Integer
   hwndProgram = ShellExecute(0, "Open", strFilePath, strParms, strDir, SW_SHOW)
   'evaluate errors
   Select Case (hwndProgram)
      Case 0
         MsgBox "Insufficent system memory or corrupt program file.", 0, "error running
" & strFilePath
         Execute_Program = False
         Exit Function
      Case 2
         MsgBox "File not found.", 0, "error running " & strFilePath
         Execute_Program = False
         Exit Function
      Case 3
         MsgBox "Invalid path.", 0, "error running " & strFilePath
         Execute_Program = False
         Exit Function
      Case 5
         MsgBox "Sharing or Protection error.", 0, "error running " & strFilePath
```

```
        Execute_Program = False
        Exit Function
    Case 6
        MsgBox "Separate data segments are required for each task.", 0, "error running "
& strFilePath
        Execute_Program = False
        Exit Function
    Case 8
        MsgBox "Insufficient memory to run the program.", 0, "error running " &
strFilePath
        Execute_Program = False
        Exit Function
    Case 10
        MsgBox "Incorrect Windows version.", 0, "error running " & strFilePath
        Execute_Program = False
        Exit Function
    Case 11
        MsgBox "Invalid program file.", 0, "error running " & strFilePath
        Execute_Program = False
        Exit Function
    Case 12
        MsgBox "Program file requires a different operating system.", 0, "error running "
& strFilePath
        Execute_Program = False
        Exit Function
    Case 13
        MsgBox "Program requires MS-DOS 4.0.", 0, "error running " & strFilePath
        Execute_Program = False
        Exit Function
    Case 14
        MsgBox "Unknown program file type.", 0, "error running " & strFilePath
        Execute_Program = False
        Exit Function
    Case 15
        MsgBox "Windows program does not support protected memory mode.", 0, "error
running " & strFilePath
        Execute_Program = False
        Exit Function
    Case 16
        MsgBox "Invalid use of data segments when loading a second instance of a
program.", 0, "error running " & strFilePath
        Execute_Program = False
        Exit Function
    Case 19
        MsgBox "Attempt to run a compressed program file.", 0, "error running " &
strFilePath
```

```
        Execute_Program = False
        Exit Function
      Case 20
        MsgBox "Invalid dynamic link library.", 0, "error running " & strFilePath
        Execute_Program = False
        Exit Function
      Case 21
        MsgBox "Program requires Windows 32-bit extensions.", 0, "error running " &
strFilePath
        Execute_Program = False
        Exit Function
    End Select
    Execute_Program = True
End Function
```

# Appendix G: Modified Startup.rcl

```
#====== Begin file startup.rcl ======
# enable/disable getting of requests
proc poll_request_on {} {
   global parms
   set parms(go) 1
}

proc poll_request_off {} {
   global parms
   set parms(go) 0
}

# ----------
# main polling routine ...
proc poll_get_request {} {
   global parms

   # temporary dir and lockfile for this instance of the server;
   # quick-and-dirty poor man's "locking" ...
   if {$parms(go) == 0} return

      set lockfile lock

   if {[file exists $lockfile]} return

   set inputfile "c:/tmp/rigi/input"
   set inputfilew "c:\\tmp\\rigi\\input"
   if {[file size $inputfile]} {
      source $inputfile
      exec c:/WINNT/system32/command.com /c del $inputfilew
   }
}

# ----------
# get things going ...
poll_request_off

proc rcl_poll_proc {} {
   poll_get_request
}

poll_request_on
```

* This file is based on the startup.rcl provided by Kenny Wong.

# Appendix H: Sample RVG file of a tree diagram

nodetype=0 typename=Collapse color=255 179 96 filter=0
nodetype=1 typename=System color=80 255 255 filter=0
nodetype=2 typename=Release color=255 255 0 filter=0
nodetype=3 typename=Procedure color=255 0 0 filter=0
nodetype=4 typename=Module color=76 201 255 filter=0
nodetype=5 typename=Revision color=255 255 255 filter=0
nodetype=6 typename=Definition color=255 46 190 filter=0
nodetype=7 typename=Implementation color=0 190 255 filter=0
nodetype=8 typename=Generic color=255 255 29 filter=0
nodetype=9 typename=Alternative color=179 255 210 filter=0
nodetype=10 typename=Document color=179 255 210 filter=0
nodetype=11 typename=Data color=255 255 0 filter=0
nodetype=12 typename=Picture color=108 255 210 filter=0
nodetype=13 typename=Accessory color=255 255 210 filter=0
nodetype=14 typename=Syntactic color=255 232 178 filter=0
nodetype=15 typename=Executable color=255 255 255 filter=0
nodetype=16 typename=Directory color=171 58 136 filter=0
nodetype=17 typename=File color=255 255 255 filter=0
nodetype=18 typename=Scope color=0 255 0 filter=0
nodetype=19 typename=Function color=0 195 107 filter=0
nodetype=20 typename=Block color=0 255 255 filter=0
arctype=0 typename=call color=255 0 255 filter=0
arctype=1 typename=data color=255 255 0 filter=0
arctype=2 typename=structure color=0 255 250 filter=0
arctype=3 typename=syntactic color=255 147 0 filter=0
arctype=4 typename=block color=147 0 89 filter=0
arctype=5 typename=include color=255 166 228 filter=0
arctype=6 typename=composite color=99 248 177 filter=0
arctype=7 typename=refractive color=0 46 190 filter=0
arctype=8 typename=level color=160 255 14 filter=0
nodeid=247 name=listfirst type=19 x=750 y=300 file=list.c lineno=45 arcs=1048609
nodeid=303 name=listinit type=19 x=650 y=300 file=list.c lineno=29 arcs=1048608
nodeid=151 name=element type=11 x=550 y=300 file=element.c lineno=7 arcs=1048610
nodeid=227 name=elementsetnext type=19 x=450 y=300 file=element.c lineno=42
arcs=1048604
nodeid=295 name=elementnext type=19 x=350 y=300 file=element.c lineno=35
arcs=1048605
nodeid=199 name=elementinfo type=19 x=250 y=300 file=element.c lineno=28
arcs=1048611
nodeid=179 name=elementcreate type=19 x=150 y=300 file=element.c lineno=13
arcs=1048600
nodeid=11 name=listcreate type=19 x=50 y=300 file=list.c lineno=14 arcs=1048599
nodeid=273 name=Base type=1 x=100 y=100 file=* lineno=* arcs=1048597,1048596

nodeid=275 name=mylistprint type=19 x=-50 y=300 file=listtest.c lineno=4
arcs=1048601
nodeid=255 name=listid type=19 x=-150 y=300 file=list.c lineno=38 arcs=1048607
nodeid=225 name=src type=1 x=100 y=200 file=* lineno=*
arcs=1048599,1048601,1048607,1048609,1048608,1048603,1048598,1048602,1048606,
1048600,1048611,1048605,1048604,1048610,1048597
nodeid=116 name=Rigi type=14 x=100 y=0 file=* lineno=* arcs=1048596
nodeid=207 name=list type=11 x=-250 y=300 file=list.h lineno=3 arcs=1048606
nodeid=163 name=listnext type=19 x=-350 y=300 file=list.c lineno=63 arcs=1048602
nodeid=243 name=listinsert type=19 x=-450 y=300 file=list.c lineno=53 arcs=1048598
nodeid=291 name=main type=19 x=-550 y=300 file=listtest.c lineno=18 arcs=1048603
arcid=1048597 src=273 dst=225 type=8
arcid=1048609 src=225 dst=247 type=8
arcid=1048608 src=225 dst=303 type=8
arcid=1048610 src=225 dst=151 type=8
arcid=1048604 src=225 dst=227 type=8
arcid=1048605 src=225 dst=295 type=8
arcid=1048611 src=225 dst=199 type=8
arcid=1048600 src=225 dst=179 type=8
arcid=1048599 src=225 dst=11 type=8
arcid=1048601 src=225 dst=275 type=8
arcid=1048607 src=225 dst=255 type=8
arcid=1048606 src=225 dst=207 type=8
arcid=1048602 src=225 dst=163 type=8
arcid=1048598 src=225 dst=243 type=8
arcid=1048603 src=225 dst=291 type=8
arcid=1048596 src=116 dst=273 type=8