Building Reverse Engineering Tools Using Lotus Notes

by

Jun Ma
B. Eng., Harbin Institute of Technology, 1996

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

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming to the required standard

Supervisor: Dr. Hausi A. Müller

Abstract

Reverse engineering (RE) tools can help people extract and understand high level abstractions of subject systems to facilitate software maintenance. However, many of these tools suffer adoption problems in industrial practice. Based on the assumption that industrial users are more likely to accept tools built on top of their current working platforms, the ACRE project (Adoption Centric Reverse Engineering) aims to attack the adoption problem by extending some commonly used tool platforms to implement RE features, rather than writing RE applications from scratch. As part of the ACRE project, my research aims to find a solution for building RE tools on top of a particular Commercial-Off-The-Shelf (COTS) host product—Lotus Notes/Domino and validating feasibility of this approach. My hypothesis is that it is practical to build tools on top of Lotus Notes/Domino to provide RE functionality and to exploit selected features of the host product, such as CSCW (Computer Supported Collaborative Work) features, to facilitate RE tasks. In this thesis, I discuss the benefits and drawbacks of building tools with Lotus Notes/Domino. I propose a solution—the ACRENotes Framework consisting of three layers: Data, Control and Presentation. This framework provides a methodology for using Lotus Notes/Domino to store, manipulate and render RE data. It also offers a Notes template and reusable libraries as a starter kit. Based on this framework, I implemented the prototype application CREST (Collaborative Reverse Engineering Support Tool), which provides selected collaborative RE features. By comparing CREST with Rigi, a traditional stand-alone RE research tool, I discuss CREST's advantages and disadvantages. Given this prototype tool and the lessons learned, I believe that

building RE tools on top of the COTS product Lotus Notes/Domino is feasible and practical.

Examiners:

Table of Contents

List of Figures

List of Tables

Trademarks

Acknowledgments

I would like to thank everyone who helped me throughout this research. Particularly, I want to say thank you to my supervisor, Dr. Hausi Müller, who supported, encouraged and guided me throughout my studies.

I would also like to thank my committee members: Dr. Daniel M. Germán, Dr. Jens H. Jahnke. My coworkers in the Rigi group helped me tremendously: Anke Weber, Holger Kienle, Piotr Kaminski, Qin Zhu, Fang Yang, Xiaomin Wu, Grace Gui, David Zwiers, and Fei Zhang.

Finally, I would like to thank my family. I miss you.

Dedication

To my parents and my grandmother

Chapter 1 Introduction

1.1 Motivation

For a long time, forward engineering has been the main focus in software engineering research. Every year, many new forward engineering terms, technologies, processes and tools are introduced to help people design and build new systems as more and more software systems become legacy systems needing proper maintenance. Here, the term legacy system means "Any information system that significantly resists modification and evolution to meet new and constantly changing business requirements [BrS95]." Nowadays, legacy systems are widespread in every domain [Gia99]. In some domains, especially Telecom and Banking, legacy systems constructed tens of years ago are still being used as for the core processing of their business. A legacy system must be continually maintained otherwise it becomes more and more unsatisfying for customers [Leh96]. Actually, huge efforts have been devoted to software maintenance to attack this problem over the past fifteen years [I3E83, I3E98]. For example, in 1993, about 75% of the information system budgets in Fortune 1000 companies was spent on software maintenance [Eas93]. However, legacy systems are still "difficult to understand and maintain due to their size and complexity, as well as their evolution history [MWT94]."

Reverse engineering (RE) technologies, such as visualization tools, provide a promising route to attack this legacy system problem [MJS00]. According to

Chikofsky and Cross, reverse engineering involves analyzing a subject system, identifying its current components and their interrelationships, and creating system representations and high level abstractions [ChC90]. Theoretically, programmers can do this work manually, for example, by reading source code and design documents. However, this approach is extremely tedious, error-prone and time-consuming, especially for large scale systems. Reverse engineering tools can help software engineers out of this tar pit. With the aid of RE tools, programmers can be freed from a lot of time-consuming tasks (i.e., code searching or pattern matching) and, thus, developers can pay more attention to extracting high level information, design information, and abstractions. However, this approach is not perfect because adoption of these tools is challenging for many reasons.

In both academic and corporate settings, RE tools suffer difficulties in becoming an effective and integral part of the standard toolset used by software engineers [Til98]. Huge barriers that prevent tools from being adopted include: their unfamiliarity with users, their unpolished user interfaces, their poor interoperability with existing development tools and practices, and their limited support for the complex work products required by industrial software development [HMK03]. Software engineers seem more likely to work effectively in one stable workspace they are familiar with and are thus reluctant to embrace new tools which do not jive well with their workspace. Implementing RE features directly in the workspace may increase the chance of tool adoption. By exploiting the deep familiarity and expertise that users already have with their favorite workspace, we can leverage the cognitive support developers have built up over long periods of time for new tools [HMK03]. Moreover, the middleware technologies supported by the workspace can also be exploited to facilitate interoperability of new tools. As a result, the interaction

capabilities between RE features and other tools in the workspace could be greatly enhanced. Spinellis drew similar conclusions for the field of visual programming tools [Spi02]. "As many visual programming environments are research-oriented, proof-of-concept projects, they can not easily compete with the commercially-developed, polished, and supported commercial Integrated Development Environments (IDE). In contrast, visual programming based on the reuse of proven existing components in a widely adopted IDE levels the playing field and allows research to focus on program representations and methodologies that can increase productivity rather than supporting infrastructure." All these prospects and potential benefits led to the ACRE (Adoption Centric Reverse Engineering) project.

1.2 Objective and Approach

Based on the assumption that end users are more likely to adopt tools, which are integrating with their current workspace, the ACRE project proposed an approach in which developers use common tools, such as COTS products, in the target users' workspace as baseline platforms and extend them to implement functionality to support reverse engineering. Thus, a key objective of the ACRE project is to investigate the feasibility and practicality of building RE tools on selected baseline platforms and to explore the relative strengths and weaknesses of these platforms.

Many tools or tool suites can serve as baseline platforms because the workspace of end users can vary significantly. Because the functionalities related to office work, such as dealing with structured and unstructured documents, diagrams, version control and electronic mail, are needed by almost every industrial user, office suites are ideal and selected as candidates for ACRE development. In today's market, there are different kinds of office suite products. Microsoft offers the Microsoft Office

Suite and Microsoft Visio, the leading diagramming tool [MOF03]. IBM sells Lotus Notes/Domino [ILD03, ILN03] and Lotus SmartSuite [ILS03]. Sun provides the OpenOffice.org [SOO03]. Adobe offers Acrobat [AAF03], FrameMaker [AFM04], as well as a suite of stellar drawing and rendering tools. In the ACRE project, these platforms are investigated separately in different M.Sc. and Ph.D. theses.

As part of the ACRE project, my research concentrates on one of these candidate COTS products—Lotus Notes/Domino. The central objective of my research is to find a feasible solution to leverage Lotus Notes/Domino's support to implement RE functions and integrate host product features with the RE features. This tool will not satisfy every user, I expect that it will benefit those Lotus Notes users who engage in reverse engineering and program understanding tasks. Moreover, developers of software engineering research tools may follow our approach of building software engineering tools using Lotus Notes/Domino product and thereby saving significant coding effort and at the same time have a better chance on getting the research results evaluated in an industrial setting.

In my thesis, I first discuss requirements for reverse engineering tools and criteria for selecting baseline platforms and then explain the cost and benefits of using Lotus Notes/Domino as a baseline platform to support RE tasks.

Secondly, given the cost-benefit analysis of Lotus Notes/Domino and the proposed development method, a layered solution—the ACRENotes Framework is developed and presented. This solution describes our approach on how to use Lotus Notes/Domino to store, manipulate and render reverse engineering artifacts.

Thirdly, I designed and implemented a prototype application, CREST, based on the ACRENotes Framework. It supports a set of RE features, which have been implemented before in the Rigi reverse engineering environment. In addition, CSCW

(Computer Support Collaborative Work) support and documentation capabilities are added to this application to facilitate collaboration in a reverse engineering team.

Finally, I evaluated the ACRENotes Framework by reviewing CREST and comparing it to Rigi and then draw some conclusions and articulate good practices and lessons learned during my research.

1.3 Outline of the Thesis

This thesis is organized as follows. Chapter 2 provides an introduction to and background on reverse engineering as well as a review of selected related research. Chapter 3 presents requirements for RE tools and a cost-benefit analysis for Lotus Notes/Domino as a host tool. Chapter 4 describes in detail the ACRENotes Framework, a new approach to support reverse engineering tasks under the Lotus Notes/Domino platform. Chapter 5 explains our implementation of the CREST prototype which is based on our ACRENotes Framework. Chapter 6 evaluates the ACRENotes Framework and documents a set of best practices and lessons learned. Finally, Chapter 7 summarizes this thesis and proposes some ideas for future work.

Chapter 2 Background

This chapter contains two sections of background for our research. The first section introduces some essential knowledge about RE and RE tools. The Adoption Centric Reverse Engineering (ACRE) project, which provides background, funding and motivation for this thesis, is especially discussed. The second section reviews related research on COTS (Commercial-Off-The-Shelf) product based software development.

2.1 Reverse Engineering

In the past two decades, legacy system research has been an emerging topic in the software engineering domain because evolving legacy systems have become a critical problem in many organizations all over the world. As a good example, the famous Y2K bug reminded people that evolving legacy systems is so necessary. Legacy systems not only constitute a huge investment made by an organization, but also embed the core business processes that are vital to the organization [New94]. As a result, legacy systems must be maintained, rather than deserted. Nowadays, on one hand, a great number of existing legacy systems developed with traditional techniques require to be well maintained; on the other hand, more and more newly implemented applications are becoming legacy systems. As a result, software engineers will be busy with maintaining legacy systems. The story of Sisyphus comes to mind in this

context. If the fate of the contemporary "Sisyphus" can not be changed, there may be some solutions to simplify their tedious work.

The first key component of software maintenance is to understand the subject software. It has been estimated that fifty to ninety percent of evolution work is devoted to program understanding [Sta84]. Program understanding can be supported by at least three categories of technologies: human browsing, leveraging existing documents and expertise, and computer-aided techniques such as reverse engineering [MJS00]. Human browsing is often hard work for users and it is almost impossible for large scale systems that are compromised of several million lines of code. Documents and expertise are not always available. The documents might be lost or out of date. The original developers may have left the organization that owned or developed the subject software. Therefore, approaches involving reverse engineering are considered as more feasible, practical and reliable.

Reverse engineering involves a series of activities. In general, these activities can be categorized into two phases: identifying components and their dependencies, extracting high level system abstraction. In a first phase, some low level artifacts, such as call graphs, global variables, and data structures, are retrieved. Based on them, higher level information, such as patterns, subsystems, architectures, or business rules, can be extracted in a second phase.

Understanding legacy systems relies heavily on the cognitive capabilities of the software engineers trying to make informed decisions. If the information about legacy systems is not understandable or manageable for software engineers, it will not make much sense. Therefore, information should be properly stored, manipulated and visualized to facilitate human understanding. After that, the internal design structure

can be understood by the software engineers. From this point, software engineers can start to make changes to the subject legacy system.

2.1.1 Reverse Engineering Tools

As described in the last section, many activities are involved in the RE process. Some of these activities have to be done by hand and others are fully automatic. More often than not, these activities are semi-automatic. Most of them can be facilitated or aided by RE tools. Normally, most activities in the first phase of RE can be done automatically with the help of RE tools (e.g., a parser). With the aid of parsers, detailed information can be extracted from source code. In the second phase, most tasks require human involvement. RE tools can assist engineers' work by providing support functions, such as visualizing data, searching by keywords or generating statistics. Figure 2.1 illustrates a common architecture of RE tools.

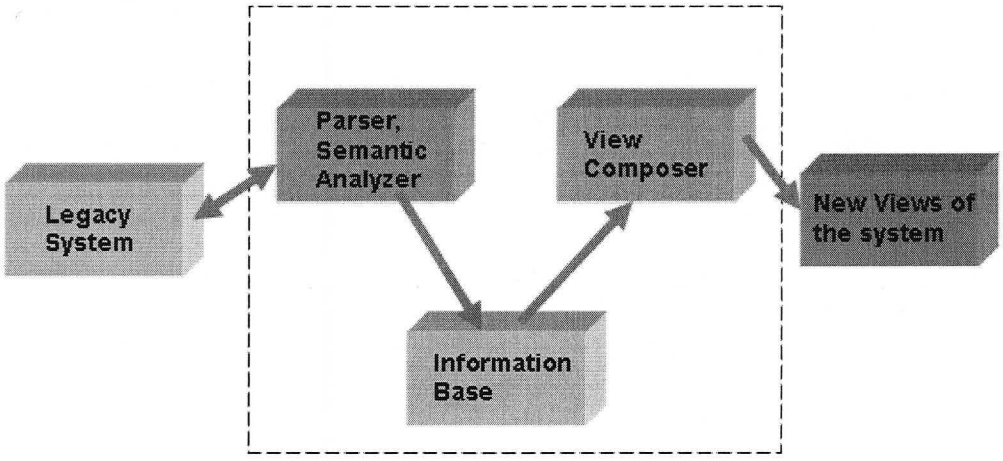

Figure 2.1: Architecture of RE tools [ChC90]

Nowadays, there are many commercial or research RE tools, for instance, Rigi [StM97], SHriMP [WuS00], Klocwork Insight [KLI03], SNiFF+ [SNI04], and Imagix

4D [IMA03]. In this thesis, we use Rigi as an example to illustrate the design and functionalities of a reverse engineering tool.

Rigi is an interactive, visual tool that assists engineers in understanding complex legacy systems [RiB03]. By summarizing, querying, representing, visualizing, and evaluating the structure of a large legacy system, Rigi can reconstruct the abstraction of the system and transfer it to the minds of the software engineers. As a result, the costs spent on maintenance will be reduced [MWT94].

Rigi employs a structure similar to the one shown in Figure 2.1. It consists of three components: Rigi parsers, RSF (Rigi Standard Format) and Rigiedit. Rigi parsers are static analyzers that parse source code, identify elements (e.g., functions and structures) and their relationships (e.g., function calls and data references). Based on this extracted information, an abstract representation of the system is constructed. The parsing system supports multiple programming languages, such as C, C++, PL/1 and COBOL.

A system representation is stored in a text file in RSF format. RSF data are normally composed of a set of triplets. Every triplet represents a data element. For instance, a function can be represented by a triplet like this:

(type 291!main Function)

Such an RSF text file is then loaded into Rigiedit. Rigiedit is a generic graph editor enhanced with domain-specific functionality for RE tasks. Software engineers use this editor to visualize and manipulate data in RSF repositories. RSF data are visualized as typed and directed graphs. Rigiedit supports multiple perspectives. This feature enables software engineers to work on different parts of the subject system. Within one perspective, software engineers can search elements, filter unused nodes, identify components and collapse nodes to make a subsystem. By recursively

applying these steps, a high level abstraction of the subject system can be built step by step.

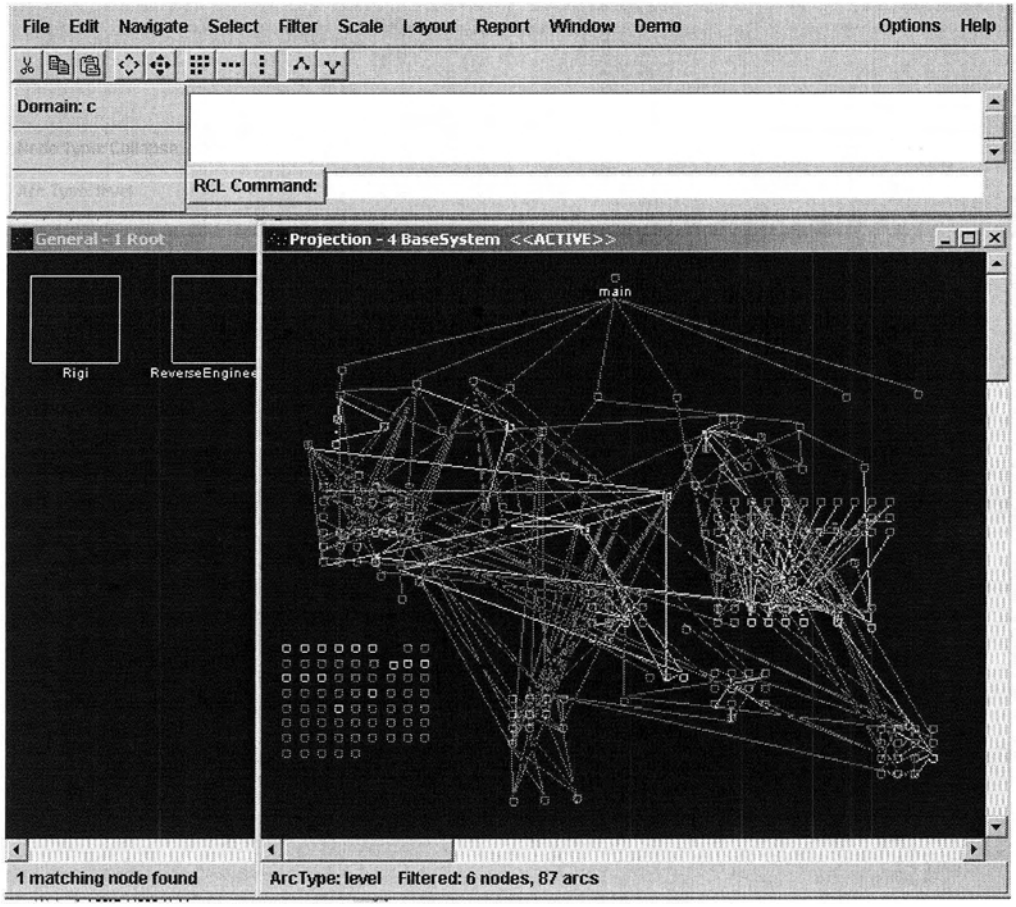

Figure 2.2: Software structure of a ray tracer system visualized by Rigi

Rigi has some distinguishing characteristics as compared to other reverse engineering tools: flexible, scalable, domain-retargetable [TMW93, Til94, Til95], and end-user programmable [Til95]. Its scalability was proven by its application in the IBM CAS (Center for Advanced Study) Program Understanding project. In that project, Rigi was used to analyze source code of the SQL/DS system including several million lines of code [WTM95]. Rigi is developed and maintained by the Rigi group at the University of Victoria. It can be freely downloaded from the Rigi website [RGH03].

2.1.2 Adoption Centric Reverse Engineering

Although Rigi has been used by quite a few academic and research organizations, it does suffer serious adoption problems in industry. Dr. Müller, who leads the Rigi group, decided to solve this adoption problem with a new perspective. He initiated the ACRE (Adoption Centric Reverse Engineering) project. The main goal of the project is to ease adoption of RE tools by concentrating on selected adoption issues rather than implementing new RE features. The lessons learned from this project are expected to benefit many software engineering research tools.

To improve the adoptability of RE tools, the ACRE researchers decided to target cognitive support and interoperability issues. Based on the experience that end users have, they are more likely to adopt tools that work in an environment or workspace that they use daily and know intimately, it is believed that by exploiting the familiarity and expertise of users one can leverage the cognitive support users have built up over time for new tools [HMK03]. Another interesting observation is that these workspaces often natively support popular technical standards (e.g., data interchange, version control, or CSCW) and hence easily interoperate with other applications. These features can be utilized by RE tools built on top of these host tools to improve interoperability.

In practice, the ACRE project explores several popular platforms, including IBM Lotus Notes/Domino, IBM Lotus SmartSuite, Microsoft Office, Microsoft Visio, Adobe Acrobat, and Adobe GoLive. The ACRE team is validating the feasibility and finding good approaches and solutions for building RE tools on top of these COTS platforms. Progress has been made in many respects. Two papers were published in the proceedings of CASCON 2003 and WCRE 2003. Further investigation and case studies will be performed based on the current research results.

2.2 Related Work

ACRE researchers are not the only people who want to build tools on top of existing products. The Carnegie Mellon Software Engineering Institute (SEI) proposed the CBS (COTS Based System) approach [CBO03], which employs similar ideas.

Different from the traditional development method, CBS research focuses on the techniques and practices of building software systems on COTS products or components. This promising development method is widely accepted by industry and academia. Its benefits are apparent: shrinking budgets, accelerating rates of COTS enhancement, and expanding system requirements [CBI03]. CBS has attractive benefits because COTS products are proven, mature and great in functionality. Moreover, the expertise and knowledge base for COTS products are already in place among end users. However, CBS is also highly challenging. For example, the capabilities of APIs (Application Programming Interface) are limited, the source code and internal design documents are often not open, and the COTS products are usually under the control of a third party company. For a variety of reasons, "COTS tools are only partially accessible and customizable, greatly limiting their reuse [EgB01]." These facts may cause big issues in terms of system integration and maintenance. The problem can become more complex when a CBS application involves multiple COTS products.

A lot of scholars have made excellent progress in this domain. From empirical data, Basili and Boehm summarized ten hypotheses for examining CBS project decisions [BaB01]. Balzer and Egyed provide a COTS integration framework that

employs the "Directional Integration with Notification" theory [EgB01]. Meanwhile, Bao and Horowitz explored the problem of how to integrate COTS products through user interfaces [BaH96]. In industry, T. Rowe Price Investment Technologies built a project tracking system on a set of COTS products which has been adopted in the company [BaK00].

The CBS approach also includes successful examples in software engineering domain. One such system is the Visual Design Editor (VDE) [GoB99]. VDE is a visual domain-specific design environment built with Visual Basic on top of Microsoft PowerPoint. This tool is mainly used to support designers who are working with different visual domain-specific languages. For instance, satellite engineers will prefer to use a design tool that contains satellite icons and terms, rather than one generic graph drawing tool. Constructing this kind of tool may have a high cost and involve tedious GUI development efforts [GoB99]. As a high-quality commercial but domain-neutral platform, PowerPoint is employed as both a graphic middleware and an end-user GUI for domain-specific design. Technically, this editor is implemented as an extension to PowerPoint. It works as the COM (Component Object Model) server to receive events from PowerPoint. It also accesses the run-time data from PowerPoint as its COM client. Moreover, the menu bar and tool buttons of PowerPoint are customized to make the domain-specific features accessible for end users.

These successes in CBS research partially back up the idea of the ACRE project.

2.3 Summary

This chapter reviewed selected concepts and tools of reverse engineering, introduced the ACRE project, and discussed related work in the CBS domain. To

maintain legacy systems well, they must be well understood. With the aid of RE and other techniques, the cost and effort can be greatly reduced. RE tools not only automatically extract artifacts from subject systems, but they also help software engineers to reconstruct high level abstractions. Rigi is a reverse engineering tool, which employs the common "Parser—Information base—View composer" architecture. However, as a stand-alone tool, Rigi suffers adoption problems. To make RE tools more adoptable, the Rigi research group launched the ACRE project to build RE tools on top of frequently used tools or common COTS products. This method is quite close to the CBS approach which brings exciting benefits and already has contributed some successful cases.

Chapter 3 Analysis

This chapter reviews the key requirements for reverse engineering tools, as well as requirements for collaboration in complex RE activities. In this chapter, I also talk about criteria for selecting baseline platforms. After that, I explain how to leverage Lotus Notes/Domino's to implement RE functionalities.

3.1 Requirements for Reverse Engineering Tools

Before designing and implementing RE tools, it is a good idea to go back and re-examine the requirements for such tools. In Wong's thesis [Wong99], 23 requirements for RE tools are identified. Among them, the following ones are particularly important in the context of this thesis.

Requirement	Description
R1	Address the practical issues underlying reverse engineering tool adoption
R2	Provide interactive, consistent, and integrated views, with the user in control
R3	Provide consistent, continually updated, hypermedia software documentation
R4	Integrate graphical and textual software views, where effective and appropriate
R5	Capture both informal and formal information

R6	Exploit expertise and capture concepts in both the problem and computing domains
R7	Support the incremental analysis and continuous understanding of software

Table 3.1: Reverse Engineering Requirements [Wong99]

In addition to these RE specific requirements, some requirements on team collaboration have to be elicited as well, because maintenance of large software systems is typically a highly collaborative activity requiring the combined efforts of a team of maintenance engineers [LoR93]. Nowadays, collaboration is not a required feature since most RE tools are stand-alone. However, with RE tasks becoming more and more complex, collaboration among engineers, such as sharing artifacts, tracing progress and exchanging messages, will become more and more important. Good support for collaboration will likely be helpful for adopting RE tools in the future. Based on the interview with some leading CASE (Computer Aided Software Engineering) tool designers, Henderson and Cooprider identified nine key requirements on the common collaboration activities in the software engineering domain [HeC90]. These requirements are also readily applicable for reverse engineering domain:

1) Maintaining a dialogue with team members
2) Allowing the team to simultaneously work on a single task
3) E-mail capabilities
4) Concurrent use of dictionary and diagrams
5) Group interaction support (e.g. brainstorming)
6) Attaching electronic notes to objects
7) Anonymous feedback or input

8) Notifying engineers if a design change affects their work

9) Building a catalog of macros accessible by the team

Both requirements for RE and the ones for collaboration determine the goals that our new tools are going to achieve. By leveraging support of baseline platforms and middleware techniques, these requirements can be partially or fully satisfied.

3.2 Choosing Baseline Platforms

Choosing a baseline platform is the first step of development in the ACRE approach. A good baseline platform can not directly solve all problems, but can provide more support and possibly ease development efforts. Basically, the ideal candidates not only provide strong cognitive support for end users, but also support powerful extension mechanisms, such as a scripting language, an API (Application Program Interface) or a Plug-in interface, to enable developers to implement extensions and integration hooks effectively. Thus, I believe that a suitable platform ought to have the following characteristics:

- Large user base: To make tools easily adopted in industry, the required expertise should be low. If a baseline platform has many users, the learning curve for tools built on top of this baseline platform will be greatly lowered. It is because the expertise of end users can be leveraged and thereby facilitate tool adoption. In other words, the end users can use their familiar "tools" to do different operations. It will be much better than learning a completely new tool. This is the fundamental assumption of the ACRE project.
- Mature user interface: For every software tool, a well polished User Interface (UI) is one of the most important keys to facilitate users to adopt

this tool. If the users are already familiar with the standard UI functionalities, they can just concentrate on learning the domain-specific functionalities. The ideal user interface should provide powerful cognitive support. It should not only include a standardized menu, a convenient tool bar, and an informative status bar, but also provide multiple effective graphical or textual views for rendering data. Some auxiliary features, such as statistics and searching, should be implemented too.

- Reasonable data representation: Data in RE tools should be represented in a simple and readable format. Therefore, it is easy to manipulate and exchange RE artifacts. At the same time, data should also be stored in a structural repository. As a result, access and manipulation of data will be lightweight and efficient.
- Powerful customization mechanisms and end-user programming capabilities: Automation techniques and scripting languages are two main ways to enable customization and end user programming. For instance, in the Microsoft Office suite, automation techniques (e.g., templates, macros and scripts) are employed to extend or customize functionalities. The scripting language Visual Basic for Application (VBA) is used to implement complex logics. In the RE domain, some complex tasks require a great number of tedious and repetitive operations. Automation can save a lot of time and effort. The script code in automation objects is reusable. Moreover, if the scripting language has already been mastered by end-users, the learning curve is lowered further.
- Strong interoperability: it is necessary for RE tools to interact with other applications. Data exchange between different tools or applications may be

the most common case of interoperability. For example, an analysis result in Rigi is converted to a Word document or a PowerPoint file. Strong interoperability of the baseline platforms can facilitate the connection between RE tools and the outer world.

Selecting a baseline platform also involves trade-offs because any platform has both advantages and disadvantages.

The first platforms investigated in the ACRE project match these characteristics to different degrees. Every candidate has its own specific features. Word is good at text processing. PowerPoint is a good tool for drawing graphs. Excel is excellent in data, table management, and calculations. Acrobat is a welcomed tool for document formatting and publishing. Compared to these tools, Lotus Notes/Domino is a more comprehensive one. Lotus Notes/Domino's abilities include strong support to manipulate text and other data with multiple data formats, native support for collaboration activities, and excellent support for control and data interoperability. Thus, Notes/Domino should be able to satisfy many requirements of RE tools.

3.3 Lotus Notes/Domino

In this thesis, all terms specific to Lotus Notes/Domino start with upper-case characters to differentiate them from the common understanding of these words.

3.3.1 Concept and Application

As a popular groupware product, Lotus Notes/Domino builds a powerful asynchronous messaging environment to facilitate communication, cooperation and collaboration in organizations. Within this environment, a variety information sources in organizations, such as e-mail, calendaring, scheduling and task list, are tightly integrated to improve productivity.

Based on the Client/Server model, Lotus Notes/Domino is often used to construct an internal information platform for an organization. On this platform, all information is represented by Notes Documents. Documents are stored and organized in a Notes Database. Domino works as a server that maintains Notes Databases and provides various services, such as database access, messaging, scheduling and so on; Notes is a client application by which a user accesses information on Domino servers. By the aid of this information platform, communication and collaboration among members in an organization can be well supported.

The most typical applications in organizations include: message exchanging, information sharing, and workflow. By sending and receiving emails, members in an organization can exchange messages. By accessing documents in a shared Notes Database on a Domino server, members get the newest information from others or publish updated information. Because the access is under control of a fine granular authorization mechanism, information access is safe. In most organizations, workflow technique is largely employed as well. Tasks, such as routing documents (projects and reports) and approving requests, can be easily supported by workflow. Workflow applications guide end users to efficiently cooperate. As a result, users can reduce overhead and errors, speed processes, and track the status of projects. [LDH03].

3.3.2 Support of Lotus Notes/Domino

In this section, the support afforded by Lotus Notes/Domino is illustrated. Here, the Mailbox application is used as an example to explain the support.

1. User interface: the user interface of Lotus Notes is mature since it was refined and perfected over six major releases. On the one hand, with the great amount of feedback Lotus developers polish the user interface continuously to be more suitable for industrial usage; on the other hand, users have built up rich end user

expertise with Lotus Notes/Domino over many years. Figure 3.1 is a screenshot of the Lotus Notes Version 6. The general user interface is composed of a menu bar, a bookmark bar, a tool bar, a status bar and a set of window tabs. Users can easily access functions and switch between different windows in a short time. In a Notes Database, Documents are organized by Views or Folders. For example, the Inbox folder of the Mailbox Database collects all incoming emails. Normally, Documents are listed as a table in a View or Folder.

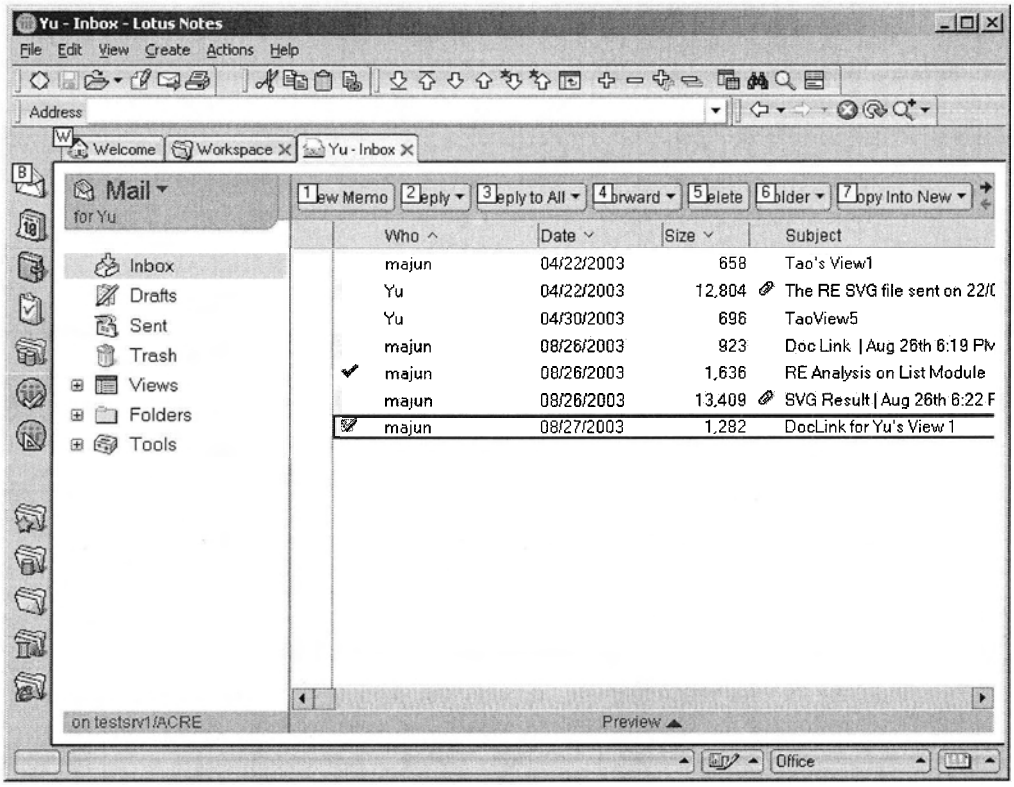

Figure 3.1: Lotus Notes User Interface

2. Document based database (DBD): Database in Lotus Notes/Domino is designed to facilitate data management. Every Notes Document is a basic unit that contains a set of data fields. It is quite similar to a record in RDB (Relationship Database). However, different from a record in RDB, a Document in Lotus Notes/Domino can hold an arbitrary number of fields. Moreover, Documents in Lotus

Notes/Domino support multimedia data natively. Therefore, pictures, video clips and other objects can be freely embedded into Lotus Notes/Domino Documents. In the Mailbox application, every email is stored as a Notes Document. One of the email Documents has an SVG (Scalable Vector Graphics) [AAA03] attachment.

3. Collaboration feature: Lotus Notes/Domino supports collaboration very well. Its native email service, calendar service, and resource service help end users easily exchange messages, manage schedules, trace progress and assign resources. For example, to launch a meeting, a manager can get identify available times of participants and the available resources through Lotus Notes/Domino. To approve a project plan, the plan Document can be routed among several stake-holders.

4. Security: Information security is always a serious concern in implementing an information system. Lotus Notes/Domino natively supports secure access of information. On the one hand, users are authenticated with digital IDs which are impossible to fake. On the other hand, Notes data are protected with a fine grained security mechanism. This security mechanism covers seven levels of data: Domain, Server, Database, View, Document, Section, and Field. Some other features, such as encryption field, role, and reader field, provide auxiliary support. With proper configuration, all Notes objects are under security control.

5. Automation and script: To manipulate large amounts of data and automate repetitive tasks, Lotus Notes/Domino offers Action and Agent. Actions, like Macros in Microsoft Word, can extend the capability of the Notes UI by using predefined functions or scripts. For example, a user can define an Action to send the current Document to another user. Agents, more powerful and complex than Actions, can be invoked by a menu click or trigged by a predefined event to run a piece of script code. For instance, in the Mailbox Database an end user can create an event-driven Agent

with Formula. So, every time when a high priority email comes, it will be automatically archived. The scripting languages which can used here are Formula, Lotus Script, JavaScript and Java.

6. Search and index engine: A search engine and a full-text index service have been embedded in Lotus Notes/Domino. By inputting one or more keywords in the search bar, Notes Documents containing the keywords can be collected together and sorted. For example, in the Mailbox Database, an end user can search for all emails related to the ACRE project with the keyword "ACRE".

7. Support for industrial standards: To interoperate with other applications, Lotus Notes/Domino supports a series of industrial standards, including quite a few popular Internet standards, such as HTTP, POP, SMTP, SSL and LDAP. The most important standards related to application development are: Java, XML, CORBA and COM.

3.4 How to Leverage Lotus Notes/Domino's Support

With proper usage, Lotus Notes/Domino can match the RE requirements mentioned in the Section 3.1 fairly well.

- R1: Practical issues for adopting a specific RE tool often come up when users lack experiences. Lotus Notes/Domino has a large user community. Many usage problems have been recorded into the knowledge database. When using a reverse engineering tool built on top of Lotus Notes/Domino, users can just focus on those features new to them because they are already familiar with the workspace and have rich experiences in troubleshooting Lotus Notes/Domino problems.

- R2: Workspace, Page, Bookmark, and other Notes UI components in Lotus Notes/Domino are deliberately integrated to facilitate the interaction between users and data. Views and Folders are employed to show users the backend data with different perspectives.
- R3: Data in Lotus Notes/Domino are represented by Notes Documents. A Notes Document may contain multiple data fields with different formats, including multimedia data and hypermedia links. Documents are shared on the Domino server. All users can access and modify them from their own Notes clients. These Documents are effectively protected. Therefore, they will be kept consistent.
- R4: Lotus Notes/Domino offers various textual presentations for rendering Documents: Form, View, and Folder. View and Folder organize Documents with a textual list. This list shows the basic information about the selected Documents. Form, directly based on Document, shows the detailed field values of a Document. Lotus Notes/Domino does not support graph rendering well, but this problem can be solved indirectly.
- R5: Formal information, such as artifacts generated in the process, can be stored as Documents in a Database. They are shared on the Domino server and protected with some access control mechanisms. Informal information, such as emails and discussions, can be managed with Domino's predefined applications, such as Mailbox and Discussion Board.
- R6: RE data can be imported into Lotus Notes/Domino and represented as Notes Documents. Therefore, the metaphor of RE is transformed to the Lotus Notes/Domino domain. The converted data are more understandable for Lotus

Notes/Domino users because their expertise with Notes can be largely exploited.

- R7: RE data and generated artifacts are stored as Documents in Notes Databases. Team members can freely access and update these shared Documents. Intermediate results can be exported to other data formats, such as GXL (Graph eXchange Language) [HoW00] and SVG. Therefore, the analysis may be continued by other engineers. If a new version of the subject software is parsed, the updated source code could be parsed. The parsed data could be re-imported into Lotus Notes/Domino with version attributes. It is possible to run Notes Agent to re-analyze the new data to update the existing perspective.

Lotus Notes/Domino in itself is a popular groupware product, which has powerful CSCW (Computer Supported Collaborative Work) capabilities. Therefore, most collaborative requirements mentioned in the former section are well supported, except real-time related features.

Lotus Notes/Domino is not all encompassing. It does have a couple of drawbacks. One is its poor capability in graph rendering. This makes RE data visualization a hard task. The other one is about real time communication. Because Lotus Notes/Domino depends on an asynchronous messaging mechanism, it does not support real-time operations directly. However, we believe that these drawbacks do not overwhelm its benefits. To some degree, they are acceptable and can be solved.

3.5 Summary

In this chapter, seven key requirements for designing reverse engineering tools were discussed. In order to facilitate analyzing large legacy systems, collaborative activities between team members should be effectively supported, for example

discussing and scheduling. This chapter listed some requirements for collaboration in the RE domain. To build tools on top of baseline platforms, a suitable platform has to be carefully chosen. I described the characteristics of a potential baseline platform. I believe that Lotus Notes/Domino might be a good candidate because it is quite versatile. After introducing selected concepts about Lotus Notes/Domino, I explained how to leverage support afforded by Lotus Notes/Domino.

Chapter 4 Approach

This chapter discusses an approach of building RE tools on top of Lotus Notes/Domino. The core of this approach is the ACRENotes Framework, which includes a set of technical designs and methods. In this thesis, all application instances implemented with the ACRENotes Framework are called ACRENotes applications. Different from those "pure" frameworks, such as J2EE and CORBA, the ACRENotes Framework does not define new terms or implement its own standards (e.g., interfaces, protocols, etc.). The main goal of the ACRENotes Framework is to leverage existing support of Lotus Notes/Domino to enable RE functionalities.

4.1 Strategy

The basic methodology of ACRE is to build RE tools on top of commonly used tools or COTS products. This method requires integrating RE features with a COTS product. According to Valetto, there are three main categories of integration methods [VaK95]:

- White Box: With this category, the source code of a COTS product is available and accessible. So, developers can directly modify the source code to build a custom tool based on the pre-existing COTS product.
- Grey Box: The source code of a COTS product is not accessible or modifiable. However, the COTS product provides an extension language, a plug-in or component architecture, or an API to enable external tools to

interact with its internal objects. Quite a few COTS products support this integration method. Some of them implement script languages, such as VBScript and JavaScript, for extension. Others provide an API which can be used from a programming language, such as C and Java.

- Black Box: The source code is not accessible. There are no extension points. The only available resource is the executable binary file. To integrate this kind of product is to wrap the application and treat the COST product as a black box. The wrapped application then interacts with end users, builds the input parameters in some appropriate data formats, sends the prepared data to the wrapped COST product, retrieves the results from the COST product and renders the result for end users.

The integration method afforded by Lotus Notes/Domino can be categorized as a Grey Box integration method. Lotus Notes/Domino provides both extension languages and APIs. This characteristic leads to two possible strategies of building RE tools with Lotus Notes/Domino:

- Connect existing RE tools with Lotus Notes/Domino through the Lotus C/C++ API
- Build a Notes application with RE features using Lotus Notes/Domino's extension languages

With the first strategy, a reverse engineering tool is required to be installed as well as Notes. RE data are stored and manipulated in the RE tool, while all core RE functions are executed within this tool. Notes is used to get user input, trigger the RE functions and render the returned results. With the second strategy, the story is different. RE data are imported into Notes as Documents. The Documents are displayed and manipulated with Notes Forms or Views. All RE operations will be

performed just within Notes. The external RE tool is unnecessary. Table 4.1 contrasts the two strategies.

	The first strategy—Integration through the C/C++ API	The second strategy—Build a Notes application
Coupling between components	Tight, at the Control and Data Level	Loose, at the Data Level
Dependency	Weak. Many technical options: C, C++, Visual Basic, DLL (Dynamic Link Library) and COM.	Strong. Dependent on one specific tool—Notes
RE features	Leverage the existing functions	Re-implement functions in Notes
Extensibility	Strong. Dependent on different techniques, such as the C/C++ API	Strong. Dependent on Notes
Maintainability	Weak. Any change can affect the whole system	Strong. Notes guarantees backward compatibility for applications
Complexity	Quite complex. Involves some complicated techniques, such as DLL and COM	Relatively simple. Involves Basic Notes-related techniques, such as LotusScript and Agent
Reliability	Weak. C/C++ API is not safe; DLL and COM are	Strong. LotusScript or Java program are relatively

	Relatively error-prone.	Safe. All operations are native.

Table 4.1: Contrasting two development strategies

After researching the points in the above table, we found that the second strategy is preferable because it is simple, loosely coupled, reliable, maintainable and extensible. This approach also has some disadvantages such as high dependency on Lotus Notes and its effort for implementing RE features. However, in general, the flaws of the second strategy are affordable, while its benefits are much more attractive compared to the benefits of the first strategy.

With the idea of the second strategy, the ACRENotes Framework is constructed by applying standard Notes development principles and leveraging open techniques offered by Notes. On the one hand, the framework uses Forms and Views to render data; on the other hand, it uses Agents, Actions and script languages to manipulate data. For those requirements that are not directly implemented in Lotus Notes/Domino, open techniques are used. For example, Notes does not fully support graph rendering. Therefore, graph exploration is not available for ACRENotes applications by default. Due to Notes's strong support for Java, this problem can be solved by integrating a small Java Swing browser into ACRENotes applications.

4.2 Architecture

The ACRENotes Framework implements the MVC (Model-View-Controller) Design Pattern. It consists of three layers: the data layer, the control layer and the presentation layer. The data layer defines how to store data and exchange data with external applications. The control layer specifies how users manipulate the data. The

presentation layer deals with how to render data. Sections 4.3, 4.4 and 4.5 discuss these layers in detail.

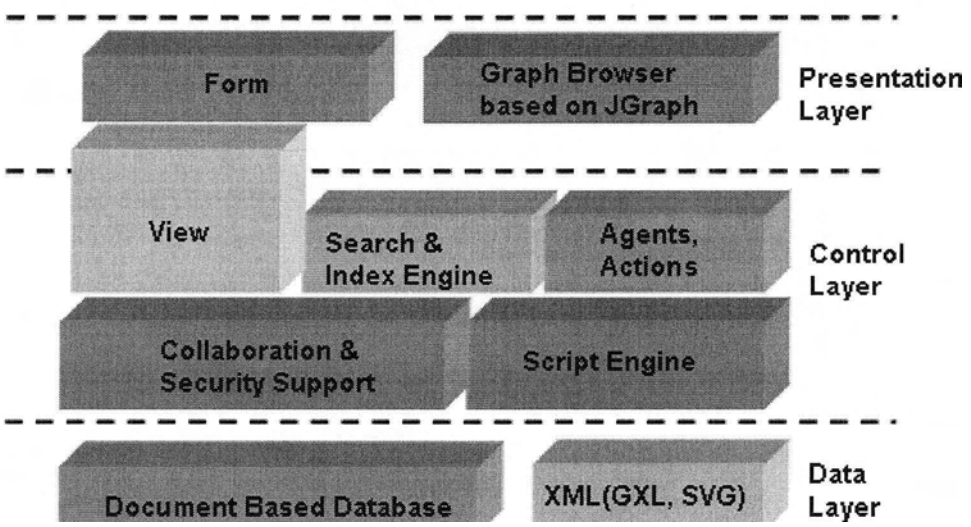

Figure 4.1: ACRENotes Architecture

4.3 Data Layer

The main objective of the data layer is to provide a data repository and enable data exchange with external applications. Here, an external application means a software product independent of Lotus Notes/Domino (e.g., Rigi or Microsoft Word). Thus, the data layer facilitates interoperability between ACRENotes applications and the outer world.

4.3.1 Data Repository

In this section, three kinds of Notes Document are introduced to explain the data repository part of the ACRENotes Framework.

To analyze a system, the fundamental RE data should be extracted at first. Normally, RE tools, such as Rigi, employ a parser to analyze source code and retrieve

the RE data. The data include two types of data items: one is the node item, for example, function, data structure, and variable; the other one is the edge item, for example, function call and data reference. Different RE tools have different ways to store the data. In Rigi, RSF is used. An RSF file is a plain text file that contains a set of triplets. The "type 291!main Function" string is the triplet for a function named main.

In the ACRENotes Framework, a data item is represented by a Notes Document. The attribute values of data items are stored into the fields of the Document. Every Document has a unique ID field for identification. Other than this ID field, some other required items, such as Name and Type, are used. A Node Document contains following fields: ID, Name, Type and Attributes. An Edge Document contains fields: ID, Type, FromNode, ToNode and Attributes. An Attributes field is employed in every Document to store auxiliary information, for example, the line number and the file name. The value of this field is a textual key-value list. Every entry of this text list is composed of two parts: the attribute name and the attribute value, which are separated by an equal sign. The following string is an example, "filename=element.c".

The Notes Documents discussed so far are used for storing the fundamental RE data, which are not modifiable during the analysis process. Another kind of Document, the ACREView Document, represents a perspective of a specific analysis task, which is updated from time to time in the analysis process. It can include all data items in the whole subject system or those data items that come from one single source file. Its size depends on the scale of the subject system. An ACREView Document commonly contains the following fields: ID, Name, NodeSet, ViewData and Description. The ID field is a unique identifier for an ACREView Document. The

Name field gives a brief human-understandable explanation. The NodeSet holds the IDs of all Node items involved in the task. These IDs are used as links to retrieve the related data items. The ViewData field stores an XML string that is used by the graph browser in the presentation layer and discussed in detail in the next sections. The Description field records comments of analysts. ACREView Documents are independent from each other. So, users can create multiple ACREView Documents for a specific task. In this case, they represent different perspectives of the same subject system.

Other than the above Documents, another Notes Document named Domain Document is employed. This Document defines the currently active domain model. A domain model is a representation that captures the structure and composition of artifacts within a domain [Til94]. In other words, it identifies those elements acceptable in this domain model. While processing data items, for example importing data, the current domain's definition is used to verify whether an imported data item is qualified. If not, some exceptions are thrown and handled. Other than defining acceptable elements, the domain models also store domain setting information, such as color. Introduction of domain Documents makes an ACRENotes application domain re-targetable and its environment customizable. An ACRENotes application can have multiple domain Documents. However, only one is enabled at a time.

In a domain Document, the domain definition is represented by an XML string. Table 4.2 contains an example of a domain definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<Domain:Domain xmlns:Domain="http://www.rigi.uvic.ca/~acre/domain"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.rigi.uvic.ca/~acre/domain Domain.xsd">
```

```
<NodeTypes>
        <NodeType Color="#33ff33">Function</NodeType>
        <NodeType Color="#ffff33">Data</NodeType>
</NodeTypes>
<ArcTypes>
        <ArcType Color="#3333ff">call</ArcType>
        <ArcType Color="#ff6666">data</ArcType>
</ArcTypes>
</Domain:Domain>
```

Table 4.2: A Domain Definition Sample

At this stage of the implementation, this domain model has to be predefined. In the future, the model information might be extracted and imported from the GXL file.

4.3.2 Data Exchange—GXL and SVG

To support interoperability, the ACRENotes Framework should at least have the capability to import data from external applications or export data to them. Basically, this process is a kind of data integration. The loosely coupled data integration simplifies the interaction between ACRENotes applications and other applications.

To exchange data with external applications, GXL (Graph eXchange Language) is employed. "GXL is designed to be a standard exchange format for information that is derived from software [HoW00]." Although GXL supports nested graphs, I only use its simplest form in the ACRENotes Framework. A GXL file used in the ACRENotes Framework has a unique root element named graph. This element

contains a list of node elements and a list of edge elements. Every node or edge element is identified by a unique value stored in its ID attribute. Every node or edge element also has a type sub-element and a set of attribute sub-elements. The type element is actually an XLink [DMD01] locator which identifies the type. The attribute elements are used to add extra information about the element, for example, the name of the corresponding data item and the file name in which the data item is located. An edge element has two more attributes than a node element. They are "From" and "To". These two attributes store the ID values of the start node and the end node. Table 4.3 shows a simple GXL file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE gxl SYSTEM "gxl.dtd">
<gxl xmlns:xlink="http://www.w3.org/1999/xlink">
<graph id="SimpleGraph">
        <node id="v1">
                <type xlink:href="schema.gxl#Function" />
                <attr name="name">
                        <string>main</string>
                </attr>
                <attr name="file">
                        <string>listtest.c</string>
                </attr>
                <attr name="lineno">
                        <int>18</int>
                </attr>
        </node>
```

```
<node id="v2">
        <type xlink:href="schema.gxl#Function" />
        <attr name="name">
            <string>mylistprint</string>
        </attr>
        <attr name="file">
            <string>listtest.c</string>
        </attr>
        <attr name="lineno">
            <int>4</int>
        </attr>
    </node>
    <edge id="e12" from="v1" to="v2">
        <type xlink:href="schema.gxl#call" />
    </edge>
  </graph>
</gxl>
```

Table 4.3: A sample GXL file

The ACRENotes Framework employs Java Agent techniques to convert GXL files to Notes Documents. When running a Java Agent, a user first inputs the location of a GXL file. The Agent loads the GXL file and parses it into a DOM tree. Next, the Agent traverses this DOM tree and extracts DOM elements. For every retrieved DOM element, a new Notes Document is created and extracted values are filled into the fields of the Document. The current domain model is referred to while converting the

GXL data. As a result, any element not part of the domain model will cause an exception.

Technically, exporting data to GXL is possible. The basic method is to traverse all Notes Documents or elements in the graph browser to build a DOM tree and then write this DOM tree as a GXL file. SVG is another possible export format. Unlike GXL documents, SVG documents can directly be shown as an interactive graph in any web browser with an SVG rendering plug-in installed (e.g., plug-in from Adobe). This system representation based on SVG is portable and can be easily visualized. SVG's portability and its visualization capabilities can facilitate effective transferring and understanding of a generated system representation. Section 4.5 illustrates SVG output in more detail.

4.4 Control Layer

The main goal of the control layer is to provide users with a way to manipulate data, make ACRENotes applications interact with other Notes applications, and enable access control.

4.4.1 Manipulating Data

Through Views and Folders of Lotus Notes/Domino, users are able to manually browse, select, open and edit Documents in an ACRENotes application. Because Views are defined with specified rules, users can filter out the data that they are not concerned with. Multiple Views help users to inspect the data model from different perspectives. Users are able to define customized Views to investigate the data model. For example, if a user wants to analyze the call relationships among functions, he/she can access the "Functions" View that only contains Function

Documents. Meanwhile, the search and index engine embedded in Lotus Notes/Domino can be used to quickly find such Documents.

Other than the manual way, programmable Agent and Action are helpful. Just like macros in Microsoft Word, Agent and Action are two automation mechanisms built into Lotus Notes/Domino, which contain snippets of script code. Therefore, they can support complex tasks. For example, to add one extra attribute to all function Documents, an Agent can be invoked to iterate over Documents and add a new field to each Document.

4.4.1.1 Script Support

Lotus Notes/Domino supports four kinds of different script languages: Formula, LotusScript, JavaScript and Java. Furthermore, a non-programmable automation called Simple Action is also supported by Notes. The following table makes briefly contrasts these approaches to automation.

Script	Complexity	Programming	Ability
Simple Action	Lowest	No	Tens of predefined operations
Formula	Easy	Weak	Simple logic, mainly work on the current document
LotusScript	Normal	Strong	Access all Notes/Domino objects
JavaScript	Normal	Strong	Access all Web objects
Java	Highest	Strongest	Access all Notes/Domino objects except UI objects; Access external resources

Table 4.4: Contrasting script languages in Lotus Notes/Domino

According to the characteristics of these script languages, we arrived at the following recommendations. For average end users of ACRENotes applications, the Simple Action mechanism is most suitable. Developers of ACRENotes applications should use Formula to implement simple tasks and employ LotusScript to accomplish complex goals. If the subject application involves interaction with external applications, Java should be used. Because JavaScript can only access web objects, it should be just used in designing web related applications.

To facilitate the building of ACRENotes applications, the ACRENotes Framework provides several reusable script libraries. In particular, we implemented two LotusScript libraries and one Java library. The two LotusScript libraries are RERigi and ACREMessage. The RERigi script library contains 12 RE related functions; the ACREMessage script library contains 10 functions that support collaborative operations, such as creating mail messages. The name of the Java library, acre.jar, is used to construct the data model and the graph model in the graph browser. It can be reused for building other Java applications either as discussed in Section 4.5.

4.4.2 Glue code to Integrate Notes Applications

Interaction between ACRENotes applications and other Notes applications constitutes a control integration problem. Because LotusScript is the native script language designed for Lotus Notes/Domino, we use LotusScript to glue ACRENotes applications to other Notes applications, such as Mailbox and TeamRoom. We expect to implement this interaction as follows: while an end user is working on a Document in an ACRENotes application, he/she can invoke an Action to create a new Document in another Notes application. Moreover, some data of the ACRENotes Document are automatically transferred into the created Document of that Notes application. The

following sample code demonstrates how to create a mail Document in the workspace and attach the document link of the current Document to it.

```
Set docLink = workspace.CurrentDocument.Document
Set db = GetMailDb()
Set MailUIDoc = CreateUIMail(workspace,db)
Set MailDoc = MailUIDoc.Document
Set RTFItem = GetRTFItem(MailDoc, "Body", True)
Call AttachDocLink(RTFItem, docLink, "the attached docLink")
Call MailDoc.save(True,True)
Call MailUIDoc.Close()
Call workspace.EditDocument(False,MailDoc)
```

Table 4.5: Sample Glue Code

This snippet of code creates a new front-end Document (MailUIDoc) with the Mail Database, retrieves its back-end Document (MailDoc) object, and sets the ACRENotes Document link in it. The code closes the front-end Document object and reopens it in the current workspace to make updates visible. This sample code can be easily modified to connect ACRENotes applications with other Notes applications.

4.4.3 Access Control

RE tools built on top of Lotus Notes/Domino support multiple users. Within this kind of environment, multiple users are allowed to access and modify data at the same time. Thus, information security is an important concern for many users. If there are inadequate access control mechanisms, important information may be exposed to intruders; work artifacts could be modified and permanently deleted by mistake; a small programming error might affect the entire system.

The ACRENotes Framework supports access control using Lotus Notes/Domino native security mechanisms, especially the Access Control List (ACL) and Reader/Author fields. Background knowledge is assumed for understanding the following sections. Slapikoff's article about ACL is a good introduction for beginners [SiL04]. The ACL in the ACRENotes Framework employs a role-based policy to apply read/write control. There are four predefined roles to represent the four types of users, respectively: DataManager, ViewManager, NormalUser and AdvancedUser. Table 4.6 describes the respective responsibilities and accessible resources for the four types of users.

Role Name	Responsibility	Accessible Resource
DataManager	Manage all RE data Documents — importing and modifying RE data; assigning permissions for data Documents	RE data Documents Read/Write
ViewManager	Manage all ACREView Documents — Modifying any ACREView Document and assigning reader permission	ACREView Documents Read/Write
AdvanceUser	Access all RE data Documents by default, analyze a specific subset of the RE data.	All RE data Documents (Read); ACREView Documents Read/Write
NormalUser	Access the part of RE data Documents that are accessible and analyze them	Part of RE data Documents (Read); Some ACREView Documents Read/Write

Table 4.6: The Role and Responsibility Table

To achieve this goal, access control is applied at three levels: database, document, and property.

At the database level, all users are granted the "Author" access privilege. Therefore, they can only create new documents and edit those documents. Four user Roles are defined and will be assigned to different types of users.

At the document level, the Reader and Author fields are required for every document in an ACRENotes application. They can refine or override the settings in the ACL. The Reader field determines who can read a document. The Author field determines who can edit a document if a user has only Author access privilege to the database. The value in a Reader or an Author field can be a user name, a group name, or a Role. As a result, fine grained access control can be applied to a person, a group of persons, or a type of persons. For example, to make all data documents visible to all advanced users, just set Role to "AdvancedUser" in the Reader fields of all data documents.

At the property level, the security property of design items is utilized. If a user's user name, group name or Role is listed in the security property of a design item, he/she can use this item. If not, the access is prohibited. For example, the ImportGXL agent can only be used by data managers. The "DataManager" role is put in the security property page of the ImportGXL agent. Therefore, this agent is invisible for other users.

In general, users' access to data and functions is strictly controlled by the access control mechanism.

4.5 Presentation Layer

The main purpose of the presentation layer is to show details of RE data and visualize RE data. Users can use two types of presentations: textual and graphical.

4.5.1 Textual Presentation

In the ACRENotes Framework, there are two types of textual presentations: Form and View. A Form in Lotus Notes/Domino, which contains fields, static text, graphics, and other objects, can be used to render details of a Document. A View in Lotus Notes/Domino, which looks like a textual list, organizes a set of Notes Documents with some predefined filter rules.

4.5.2 Graphical Presentation

The basis of a graphical presentation is an ACREView Document. A user can select those corresponding node Documents and create an ACREView Document. Each ACREView Document is a customized representation of the currently analyzed system. This representation is rendered as a graph of nodes and arcs, which can be shown with a customized graph browser. Within this browser, users can move, edit, select and filter nodes and arcs. Because Lotus Notes/Domino does not directly support graph rendering, we implemented this graph browser using Java Swing. At the run time, it is activated by calling a Java Agent named "ShowGraph".

The design of this browser is based on the MVC (Model-View-Control) design pattern and is implemented using the JGraph [Ald04] library. JGraph is an open source Java Swing component which supports graph drawing, editing and manipulating. Figure 4.2 exhibits the basic structure of the graph browser.

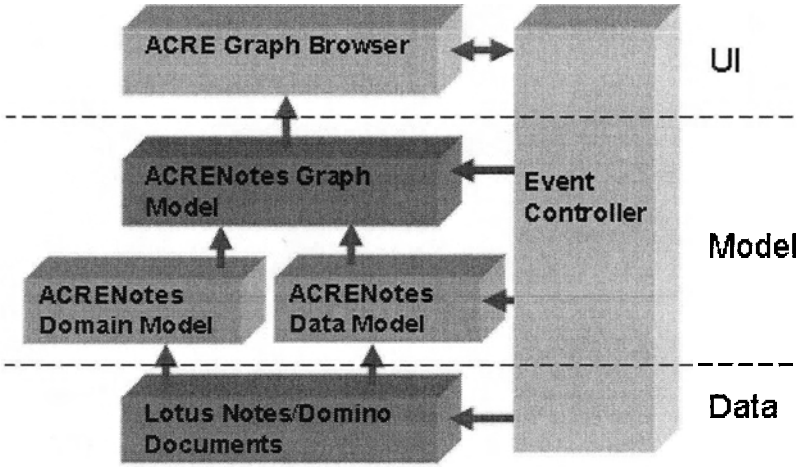

Figure 4.2: The Architecture of the Graph Browser

While this browser is being initialized, the node Documents referred to in the current ACREView are parsed. Extracted data are used to construct the ACRENotes domain and data model. Next, the ACRENotes graph model is created based on the two models. This graph model contains a JGraph object and provides a set of public methods for manipulating the graph object. After inserting the JGraph object into a Java Swing frame, the GUI of the browser appears. An event controller is registered for the Swing frame. Trigged by events, this event controller manipulates the data and graph model, or directly accesses the fundamental RE Documents.

A jar file named "acre.jar" contains the classes used by the ACRENotes domain, data, and graph models. This jar file also contains a set of utility classes, such as SVGOutputer. This configuration makes the core of the graph browser reusable for other Java graph applications. The GUI initialization code and the event controller are defined in the "ShowGraph" Agent. Therefore, developers of ACRENotes applications are able to tailor or reconfigure the graph browser by rewriting code in the Agent.

Presentation information, such as locations and sizes, is also stored into the ACREView Document. The presentation data are parsed as a long XML string, which is verified by an XML schema named AXL. The definition of the AXL schema and a sample are listed in the Appendix.

4.6 Instantiation

In the ACRENotes Framework, we create an ACRENotes template to facilitate development of ACRENotes applications. It includes a Notes template database and several supportive Java libraries. The Notes database is named acre.nsf. The jar libraries are acre.jar and jdom.jar. This template contains some basic Forms, Views and Agents related to the RE domain. The script libraries we mentioned in Section 4.4 are also embedded. Using this template as a starting point, developers can quickly customize the template to create a tool for specific RE tasks.

4.7 Summary

In this chapter, we performed an analysis of two different solutions for customizing Lotus Notes/Domino to build RE tools. Based on the choices we made, a solution named ACRENotes Framework emerged. This ACRENotes Framework contains three layers: the data, control and presentation layers. As the data layer, a Notes Database is used as a data repository; GXL and SVG are employed to exchange data with external applications. At the control layer, data in the ACRENotes Framework can be automatically manipulated or modified with some automation mechanisms, such as Agents and Actions. Interactions between ACRENotes applications and other Notes applications can be supported by some reusable glue code. Access control is applied in the ACRENotes Framework to protect data. At the presentation layer, in addition to textual presentations, a Java Swing based graph

browser was implemented to visualize system representations of the target system. It is designed based on the MVC design pattern and implemented using JGraph. Finally, the ACRENotes Framework can be instantiated with a predefined template. Developers can customize this template application to make it more task-specific.

Chapter 5 Prototype—CREST

To verify the effectiveness of the ACRENotes Framework, a prototype named CREST (Collaborative Reverse Engineering Support Tool) was constructed. This chapter discusses its design and illustrates how to use CREST to analyze a small subject system.

5.1 Design and Implementation of CREST

CREST is an application constructed based on the ACRENotes Framework. It is supports selected RE functionalities and collaborative activities.

Some RE features already implemented in Rigi were chosen as the required features of CREST.

- Import/Export RE data
- Organize RE data
- Generate/Modify perspectives, including layout (Spring and Suguiyama) and filtering (by type, by selected nodes)
- Find dependency (children, parents, forward tree and backward tree)

Technically, Collapse/Expand feature can also be implemented by dynamically creating Documents for collapsed nodes. But this process is a little complex. I believe that the above features are already good enough as a proof of concept. Therefore, the Collapse/Expand feature is not included in the current implementation.

CREST supports the following collaborative features: send ACREView Document links, assign RE tasks, publish intermediate results, and apply access control mechanisms.

CREST can be run either on a Lotus Notes client or a Domino server. Because a single Notes client can not support collaboration features and implement access control, the Domino based configuration is preferred. In this configuration, an end user can access the prototype installed on a Domino server through a local Notes client. To deploy CREST in this configuration, the Domino administrator needs to create a blank ACRENotes application on the Domino server by copying the ACRENotes template. On the Notes client side, a user needs to copy two jar files (jdom.jar and acre.jar) into a local directory and set the JavaUserClasses entry of the Notes.ini file to point to these two files. The CREST UI is shown in Figure 5.1.

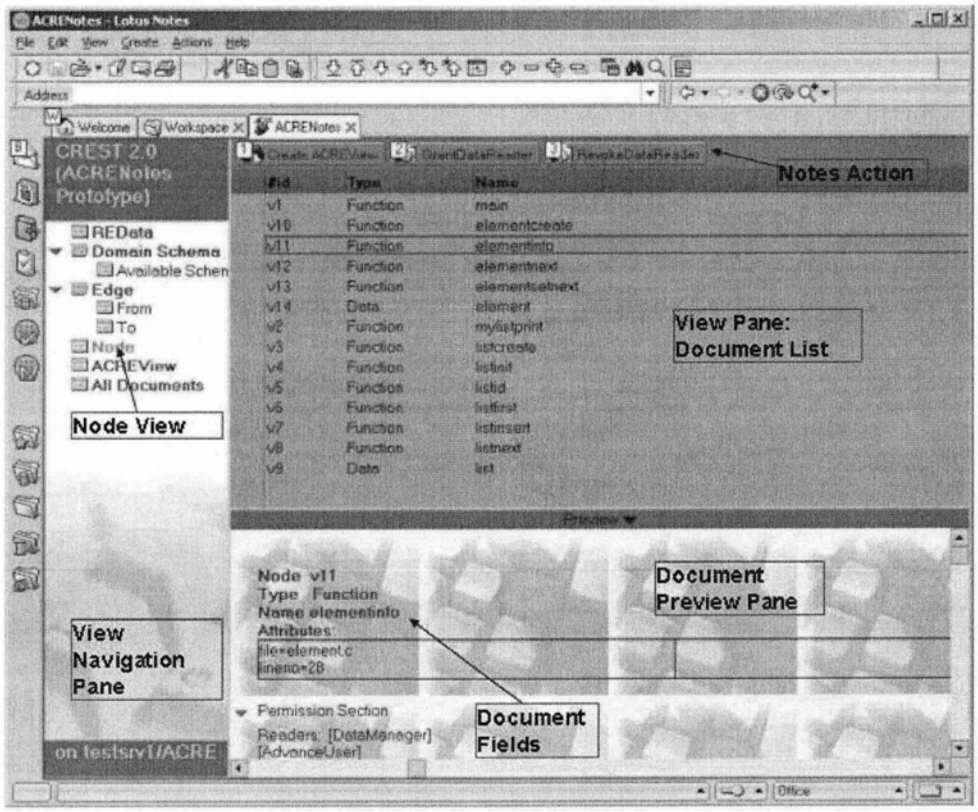

Figure 5.1: The main UI of CREST

Sections 5.1.1, 5.1.2 and 5.1.3 discuss the design in detail with respect to different layers. The design items of CREST are listed in Appendix C.

5.1.1 Data Layer

Since Section 4.3 explained the details of Documents and GXL, this section focuses on the process of importing GXL documents and exporting SVG documents with CREST.

The GXL conversion is done by a Java agent named ImportGXL. The security property of this agent is assigned to data managers so that only users with the "DataManager" Role can use this agent. After the "DataManager" user invokes this Agent, a file dialog prompts the user to ask for the location of the GXL file. Next, the Agent calls JDOM [HuM04] to parse the GXL file and generate a JDOM document object. During this process, JDOM validates the GXL file with the GXL DTD (Document Type Definition) [RSS01] to check whether it is well formed and legal. The Agent traverses all node and edge elements from the JDOM document tree, and creates new Notes Document objects one by one. All these Notes Documents are stored in the Notes database.

The process of SVG output is a little complex. Basically, the process is the reverse of the GXL conversion. When a user clicks the "Output SVG" menu item in the graph browser, the current JGraph object is passed to an object of the helper class SVGOutputer. The SVGOutputer object uses the ACREDomain Model and a template SVG file to create an initial JDOM document object. Next, all elements in the JGraph object are traversed to create JDOM element objects, which are attached to the JDOM document object. When the process of building the JDOM document object is over, an XMLWriter object was invoked to save the document object as an SVG file.

5.1.2 Control Layer

CREST utilizes eight types of Notes Views to enable users to manipulate Documents. The Domain Schema, Domain Schema\Available Schemas, and REData Views are only visible for "DataManager" users. By enabling a suitable Domain Schema Document in the "Domain Schema\Available Schemas" View, a "DataManager" user can switch CREST to a different domain. In the "REData" View, the "DataManager" user can browse all RE data Documents.

Other Views (Node, Edge, Edge\From, Edge\To and ACREView) are accessible for "ViewManager", "AdvancedUser" and "NormalUser". With the first four Views, users can browse RE data Documents from different perspectives. With the "ACREView" View, users can manage their own ACREView Documents. Thanks to the search and index engine embedded in Notes, users are able to find Documents by keywords in any View quickly.

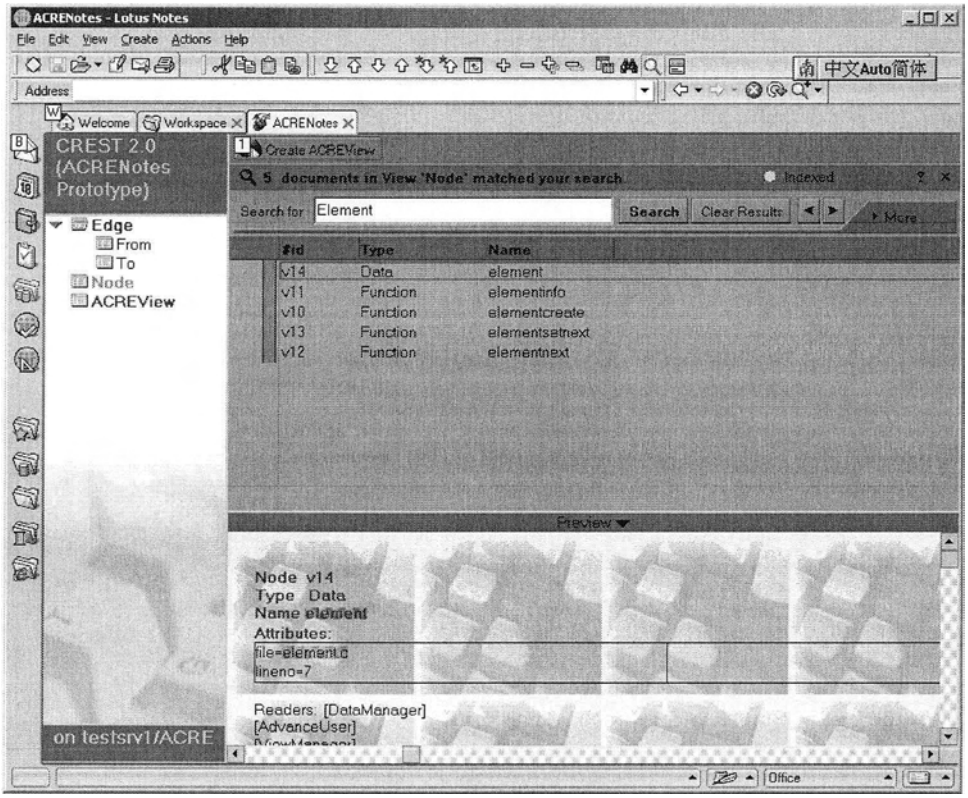

Figure 5.2: Search View in CREST

In CREST, most automation tasks are implemented based on Notes Agents and Actions. The ImportGXL Agent is described in Section 5.1.1. The ShowGraph Agent is explained in Section 5.1.3. This section only talks about Actions. The "GrantViewReader" Action and "RevokeViewReader" Action, included in the "ACREView" View, are used by the "ViewManager" user to control access to ACREView Documents. The "GrantDataReader" Action and "RevokeDataReader" Action are embedded in the "Node" View and the "Edge" View. They are used by the "DataManager" user to grant or revoke access to all fundamental RE Documents. All four Actions are implemented with LotusScript. In the "Node" View, the "Create ACREView" Action enables users to create an ACREView Document with selected Node Documents.

In the "DomainSchema" Form, an Action named "Enable Current" lets the "DataManager" user enable a Domain Schema. In the "REView" Form, three Actions are added to support collaboration activities. They are "Email DocLink", "Assign Task" and "Create TeamRoom Topic". These three Actions enable CREST to interact with two other Notes applications: Mail and TeamRoom. Here, we take the "Assign Task" Action of the "ACREView" Form as an example. The "Assign Task" Action first creates a blank Todo Document object in the Mail application. Next, this Action extracts useful attribute values from the open ACREView Document and constructs a brief task memo. This memo and the Document Link pointing to the current ACREView Document are attached to the "Body" field of the Todo Document object. Next, the Form UI bound to this new Todo Document object is activated. A user can add more comments and set task properties, such as the due date and the priority.

Finally, the user can send out this Todo Document. As a result, a task is assigned to the recipients.

The other two Actions employ a similar design. The only difference is that the "Email DocLink" Action sends out one descriptive email with a Document Link and the "Create TeamRoom Topic" Action creates a new Discussion Topic Document in the "TeamRoom" application. All these Actions are implemented with LotusScript code based on the glue code introduced in Section 4.4.1. Additional collaborative tasks, such as replying, accepting, rejecting, and adding responses, can be completed in the Mail or TeamRoom application.

5.1.3 Presentation Layer

As discussed in Section 4.5.1, Form and View are two components to support textual presentations. Figure 5.1 shows the outlook of the "Node" View. In Notes, a View is normally constructed based on Formula criteria and some property settings on the View's outlook. In the "Node" View, the criteria Formula is "Form = "Node". Three columns, "#id", "Type" and "Name", are defined to list corresponding values of Node Documents. A Form is used to show details of a single Document. Normally, a Form is composed of a group of fields. Each field can have one or more field values. These fields can be rendered as labels, checkboxes, calendars, radio boxes, dropdown lists, and so on. Figure 5.3 is a screenshot of a Node Form.

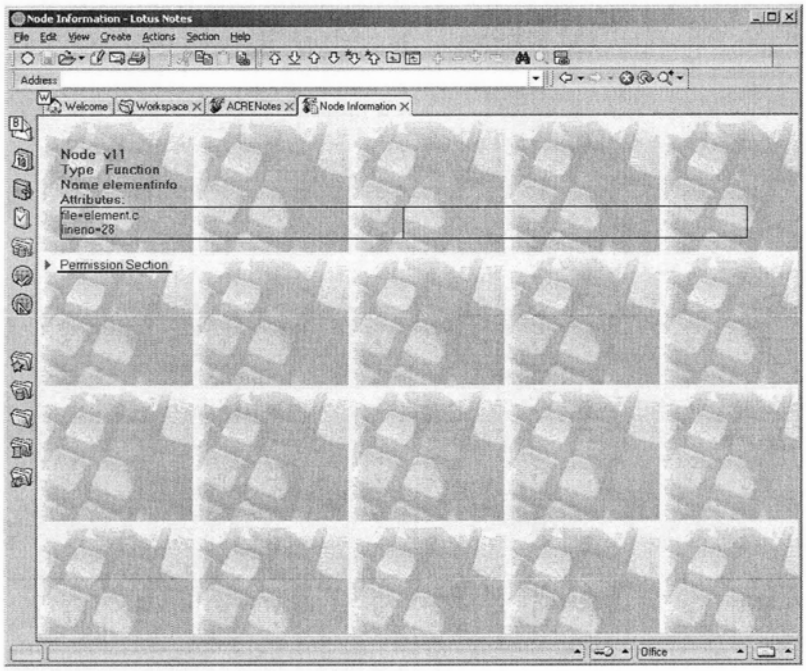

Figure 5.3: Node Form

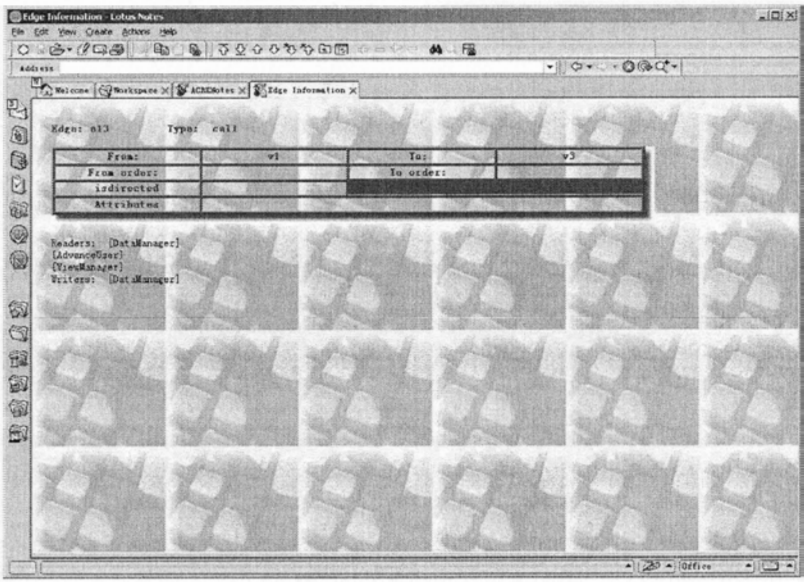

Figure 5.4: Edge Form

The graph browser is another component in the presentation layer of CREST. It is invoked by the "ShowGraph" Agent, which is located in the "Action" menu of

Notes. This browser offers a set of features that can be activated from its menu bar. Table 5.2 describes these features.

Menu	Menu Item	Description
Data	Load View	Load the presentation data from the current ACREView Document.
	Save View	Save the presentation data of the current graph into the ACREView Document
	Output SVG	Export the current graph into a SVG file
Edit	Undo	Undo the past operation
	Redo	Repeat the past operation
View	Zoom in	Enlarge the graph view
	Zoom out	Shrink the graph view
Select (At least one node should be selected firstly)	Get Children	Highlight the children nodes of the selected node
	Get Parent	Highlight the parent nodes of the selected node
	Get Forward tree	Highlight the whole forward tree based on the selected node
	Get Backward tree	Highlight the whole backward tree based on the selected node
	Select All	Highlight all nodes in the current graph
	Clear Selected	Unselect all selected nodes
Filter	Filter by Type	Based on the selected types, hide nodes and edges

	Filter Selected	Filter out all selected nodes from the current graph.
	ClearFilter	Unset all filters to restore the graph
Layout	Spring Layout	Rearrange all nodes with Spring Layout
	Suguiyama Layout	Rearrange all nodes with Suguiyama Layout
Help	About	Show some information, such as version.

Table 5.1: CREST graph browser features

The graph browser is composed of four components: The kernel, the initialization module, the browser frame and the event controller. The kernel part encapsulates a set of Java classes, which are packed in the "acre.jar" file. The other parts are implemented in the "ShowGraph" Java Agent. We employ this design to offer developers an opportunity to customize the graph browser. In other words, developers can change the code in the "ShowGraph" Java Agent to make different graph browsers. Because the basic structure of the graph browser is introduced in Section 4.5.2, we just discuss here how to customize the browser. The "ShowGraph" Agent includes three Java files: JavaAgent.java, CrestBrowserFrame.java and ViewController.java. The JavaAgent.java is the starting point of the "ShowGraph" Agent, which is responsible for initializing the graph browser. It retrieves the related Notes Documents to instantiate data model objects, such as the ACREDataModel and ACREGraphMoel. Next, it creates a ViewController object using the model objects as parameters. Because the View Controller implements the ActionListener interface, it can respond to events activated in the graph browser. Next, the JavaAgent creates the CRESTBrowserFrame object. This object takes the CREGraphModel object and the ViewController object as its data members. After that, the graph browser shows its UI to end users. When an end user selects a menu item in the graph browser, a user event

is triggered. The ViewController object gets this event and extracts its command string. Based on this command string, the ViewController manipulates the embedded model objects according to the predefined code. Therefore, to customize the graph browser's outlook and behavior, the code in CRESTBrowserFrame.java and ViewController.java has to be adjusted.

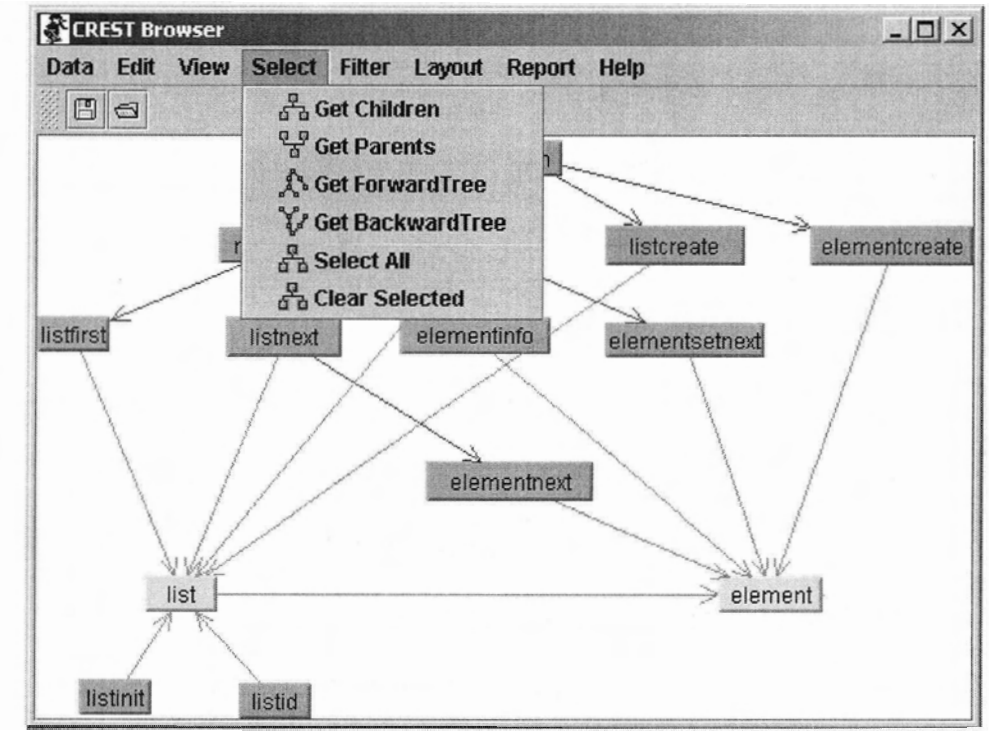

Figure 5.5: CREST Browser UI

5.2 Sample Scenario

In this section, three sample scenarios are employed to illustrate how to use CREST to analyze a small system named List. Implemented in C, the List system contains just five files: list.h, element.h, list.c, element.c and listtest.c. The total number of Lines Of Code (LOC) is 180. There are twelve functions and two structures defined in this system. There are twelve references between functions and variables, including function calls and data references.

5.2.1 Task and Configuration

Let us assume that a small maintenance team is responsible for this project. The main goal of this team is to analyze the system and generate a report with some necessary artifacts, such as call graphs. The team will take a GXL file as the data source, which contains all reverse engineering artifacts extracted from the source code.

Let us suppose this maintenance team has three members: Qin, Yu and Tao. Qin is the team leader who controls this team in general. As a senior software engineer, Yu is the analyst of the team. She reviews the whole system, assigns subtasks to other members and integrates results from members. Tao is a junior software engineer. He works as an assistant and is responsible for subtasks assigned by Yu. One subtask focuses on the functions in the element.c file.

To assist this maintenance team, the Domino administrator makes some necessary configurations on the Domino server. He creates the CREST database and grants access privileges as follows:

User	Access Level	Role
Qin	Author	DataManager and ViewManager
Yu	Author	AdvancedUser
Tao	Author	NormalUser

Table 5.2: The ACL Configuration of CREST

Next, he creates one TeamRoom Database using the TeamRoom template. He makes Qin the "Manager" of the TeamRoom application. The other two members are assigned "Author" access.

So far, the basic work environment is constructed. For each member of this team, three Notes applications are accessible. They are: CREST, the TeamRoom and the personal Mail application. Figure 5.5 shows a typical workspace.

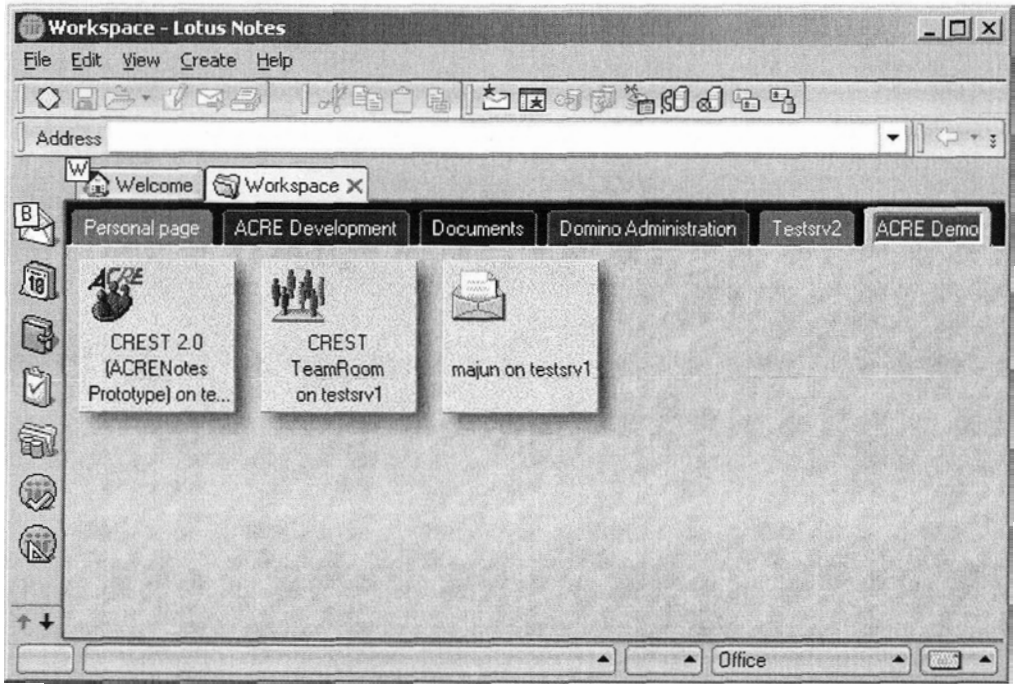

Figure 5.6: Notes workspace

The activities that this team has to do include: importing data, analyzing, generating artifacts, exchanging opinions and forming the final report. The following sections describe about how CREST supports team work for this process.

5.2.2 Data Management

The first activity to do is to import fundamental RE data from external data sources. At the beginning, all RE data is stored in a GXL file. As a "Data Manager", Qin first creates a Domain Schema Document that defines the acceptable elements. The domain in this task is a simplified C domain that accepts two Node types (Function and Data) and two Edge types (call and data reference). Next, Qin runs the

"ImportGXL" Agent and uses a GXL file named list.gxl as the data source. The Agent generates a DOM tree and creates corresponding Notes Documents. These RE Data Documents are organized into two Views: Node and Edge. Figure 5.6 exhibits the Node View and the "ImportGXL" Agent.

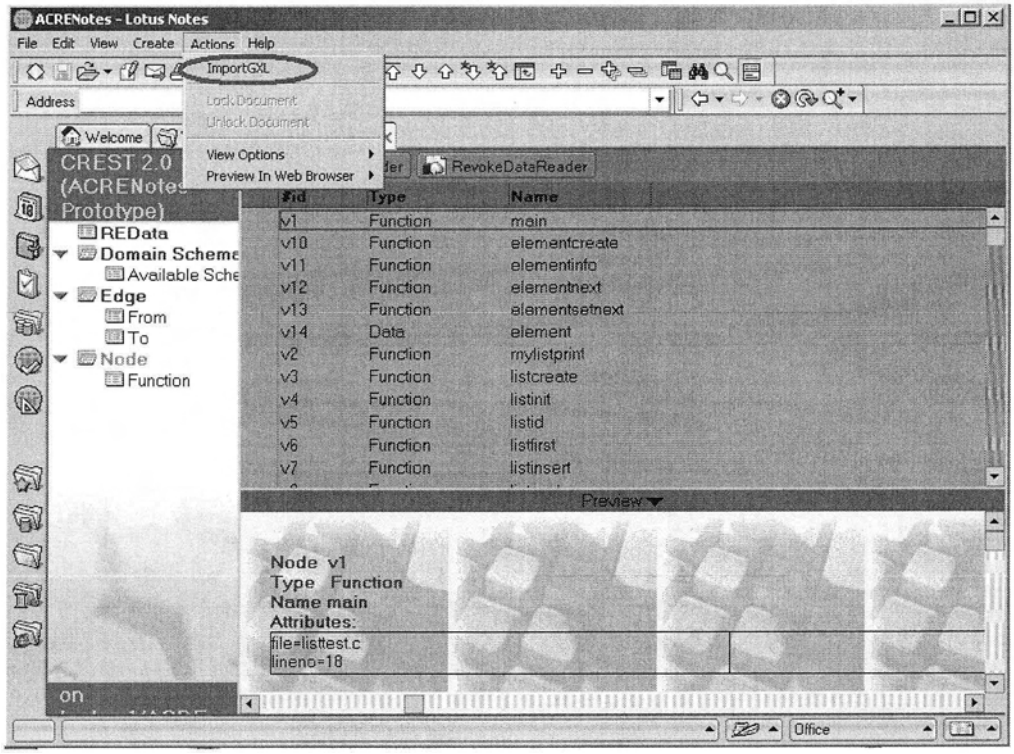

Figure 5.7: Screenshot of the Node view

By default, Yu can access all data Documents because she is an "AdvancedUser". However, Tao can not access the documents. To let Tao analyze those Documents related to the element.c file, Qin uses the Notes embedded engine to search all related Documents that contain the keyword "element.c". He clicks the "GrantDataReader" button and grants Tao the privilege of reading these Documents. As a result, these Documents become visible to Tao.

5.2.3 Analysis Activity

Yu starts to analyze the List system. First, she opens the Node View. After that, she selects all Documents in the Node View and clicks the "Create ACREView" button. A new ACREView Document is created. Yu names it "YuView1". From the ACREView View, Yu opens this newly created Document.

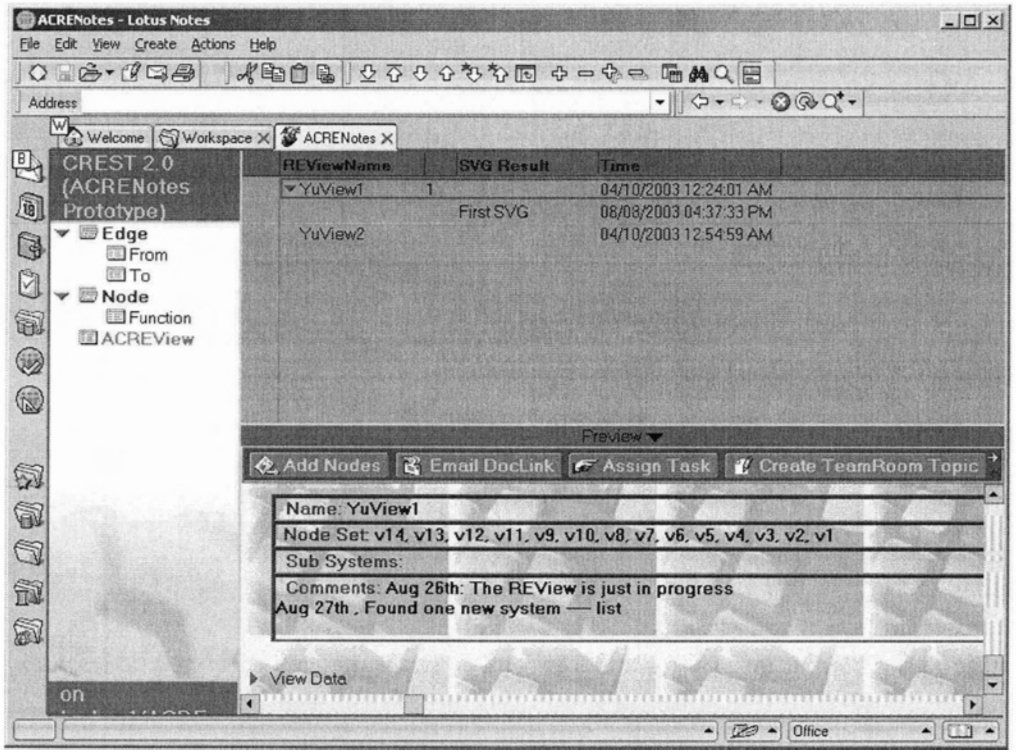

Figure 5.8: Screenshot for the User Yu

In the context of this ACREView Document, Yu chooses the "ShowGraph" Agent from the "Actions" menu. A Swing based graph browser shows up. At the very beginning, nodes in the browser are arranged according to Sugiyama Layout. Yu rearranges all nodes and resizes them to facilitate understanding of the artifacts. Finally, a hierarchical structure of nodes can be rendered as follows.

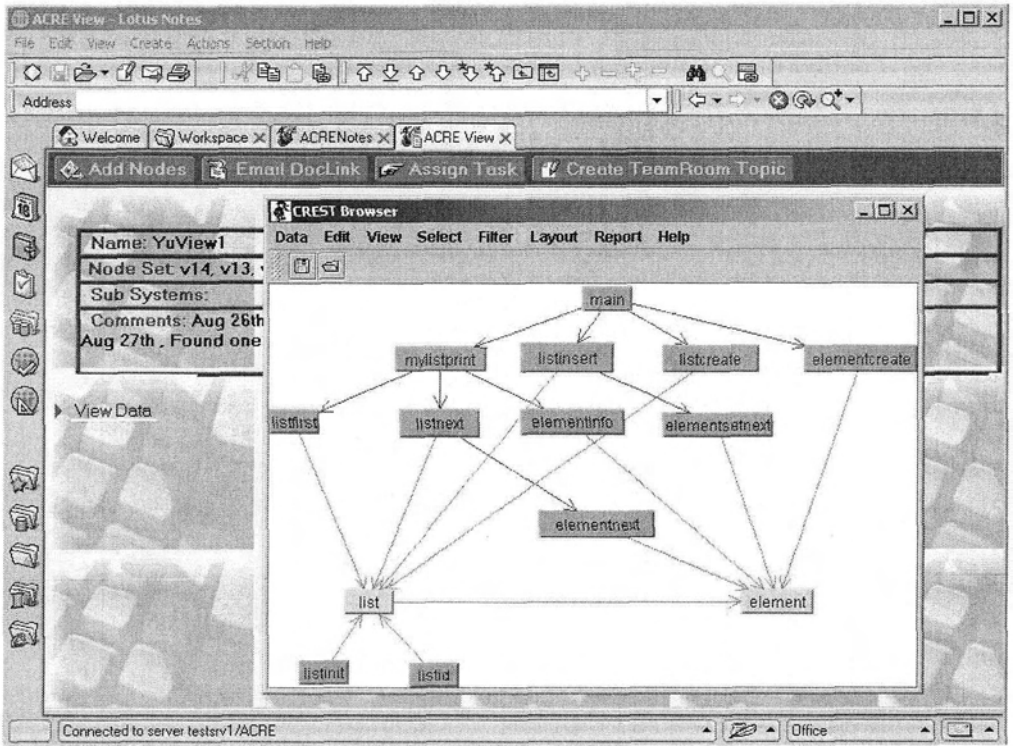

Figure 5.9: Screenshot of the graph browser

Within this graph, Yu finds that there are two unreachable Function nodes: listinit and listid. Using the graph browser, Yu can further analyze the system. For example, Yu can select the "element" node and click on the "GetBackwardTree" menu item to find the dependents of this "element" node. Therefore, Yu can understand which function is directly or indirectly affected by a change to the "element" data structure. From time to time, Yu makes some snapshots with SVG to record intermediate analysis results. These SVG files are attached in the Response Documents of the ACREView Document to track the analysis process. Yu can also click the "Email Doclink" button to send the link of the current ACREView Document to others. As a result, other people can study the currently updated perspective by opening the ACREView Document via that link.

5.2.4 Collaboration Activity

As the analyst of the team, Yu assigns some subtasks to Tao. She selects those Documents related to the "element.c" file and creates an ACREView Document named TaoView1. She opens this Document and clicks the "AssignTask" button. A "ToDo" Document is created. The Document Link of the created ACREView Document is attached to the "ToDo" Document. After making some necessary settings, such as the priority and the due date, Yu sends this "ToDo" Document to Tao.

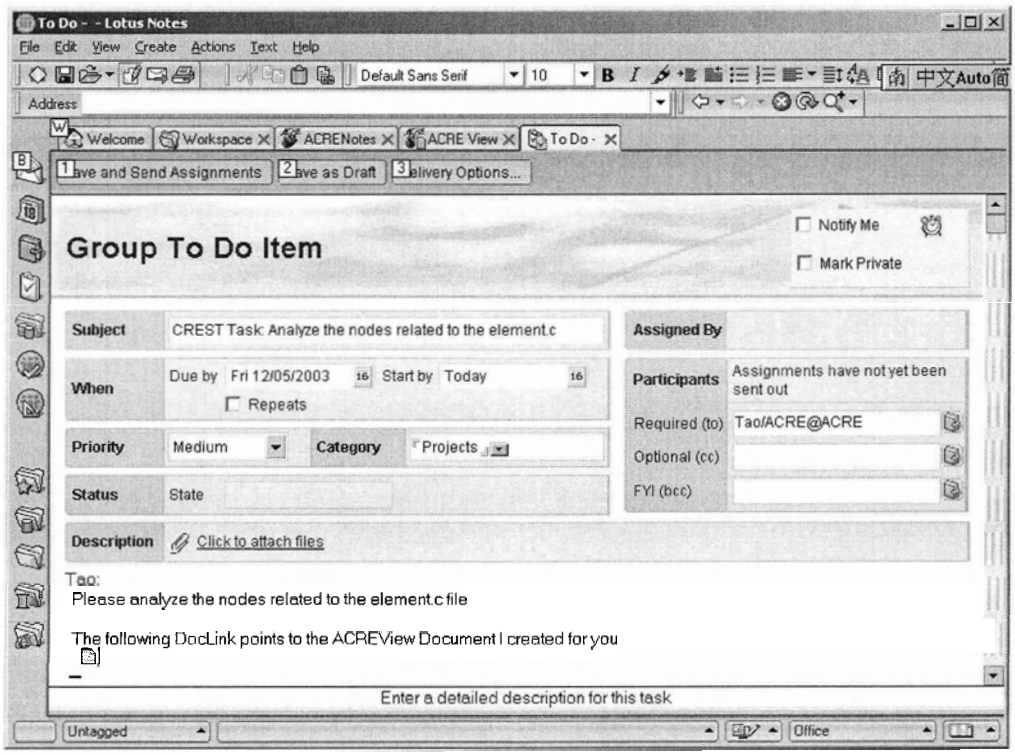

Figure 5.10: Create a Todo Task

Tao finds this "ToDo" Document in the "To Do" section of his personal Mail application. He clicks the "Respond" Button to accept this assignment. A reply mail is automatically sent back to Yu.

Through the graph browser, Yu creates an SVG file as the intermediate snapshot of the target system. She wants some feedback from Qin. Therefore, she

opens the Document containing the SVG file and clicks the "Email SVG" button. An Email Document with the SVG attachment is automatically generated. Yu assigns Qin as the recipient and fills the subject and body fields. After clicking the "Send" button, she sends this email to Qin.

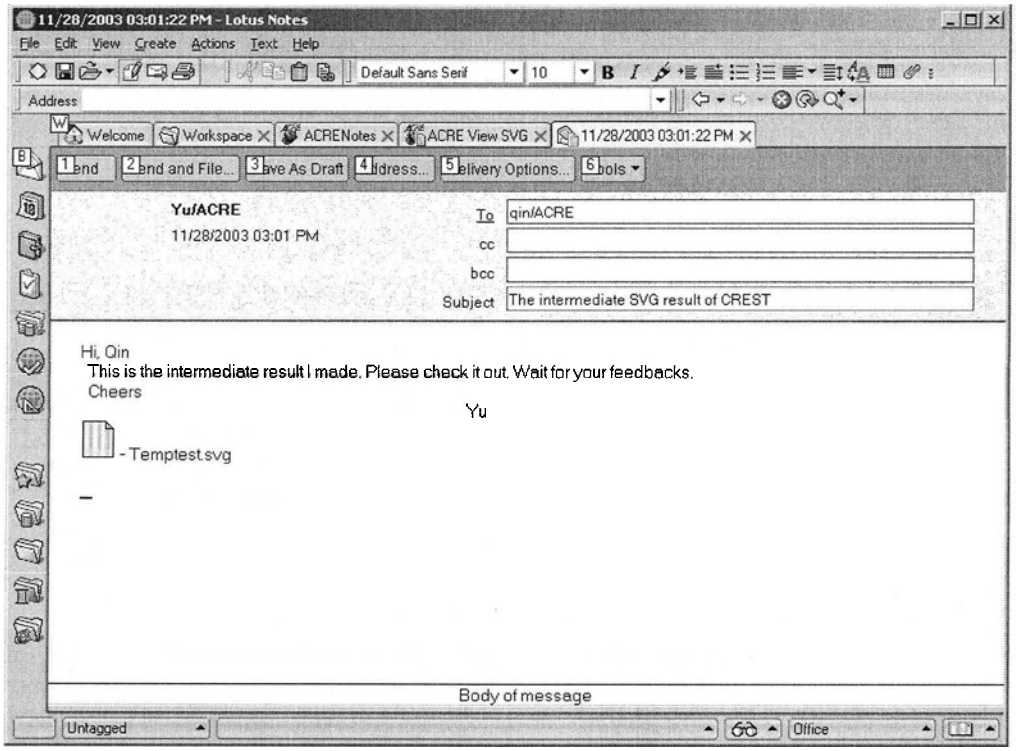

Figure 5.11: Email the SVG artifact

Discussion between team members is a good way to improve understanding and initiate new ideas. Thus, Yu wants all members in the team to participate in forming the final report. Therefore, she initiates a Discussion Topic Document in the "TeamRoom" Database. After opening the Document containing the SVG file, she clicks the "Create TeamRoom Topic" button. A "Topic" Document is created in the "TeamRoom" Database and the SVG file is automatically attached to it. After adding some descriptive text, Yu saves and closes this Document. As a result, everybody in the team can read this Document and write down his or her feedback with "Response"

Documents. Figure 5.11 shows the screenshot of the "Topic" Document created by Yu.

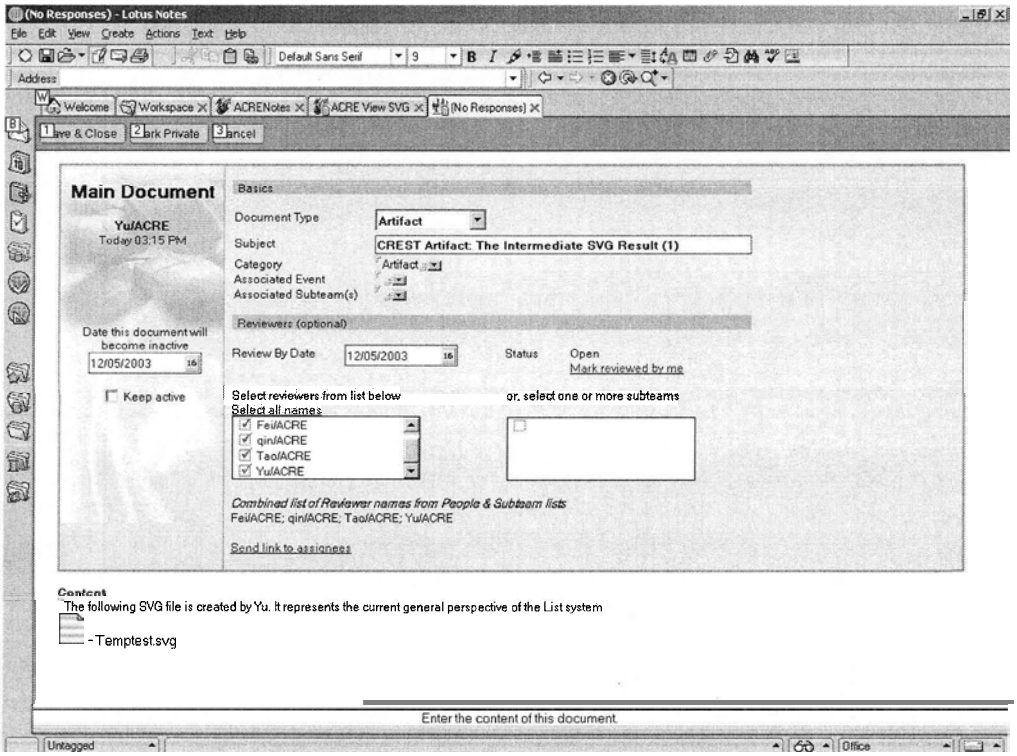

Figure 5.12: Create a discussion topic in Teamroom

5.3 Summary

In this chapter, we described how to build a CREST prototype based on the ACRENotes Framework and how to use it to support a maintenance team in analyzing a small system. The construction of CREST does not require much effort. The starting point is the template mentioned in Section 4.6. Some design items, especially Views and Forms, are added and customized to make the application more specific for RE tasks. Glue code ties CREST with other Notes applications, such as Mail and TeamRoom. A Java Swing based graph browser is used to visualize the structure of the analyzed system. Its outlook and the event handler can be customized in the

"ShowGraph" Agent. Three sample scenarios illustrated how a maintenance team analyzes a small system using CREST. The activities involve in managing RE data, visualizing the hierarchical structure of the system, and collaborating between members (e.g., assigning tasks, emailing artifact, and launching discussion) are also outlined.

Chapter 6 Evaluation

In this chapter, I first provide a review of the functionalities of the prototype application and then perform a comparison between CREST and Rigi in terms of non-functional features. Finally, I articulate some research practices.

6.1 Prototype Review

CREST implements the RE features listed in Section 5.1. Using CREST, end users are able to

- ✓ Import RE data from GXL files; export perspectives into SVG files;
- ✓ Organize RE data with different categories (Notes Views); locate RE data with a built-in search engine;
- ✓ Create perspectives; render perspectives as graphs; customize perspectives using layouts and filters; and
- ✓ Find dependencies with visualized perspectives.

CREST basically satisfies the RE requirements mentioned in Chapter 3. CREST employs as its main environment the Lotus Notes's interactive workspace with its rich feature set. Notes Views and Forms are used to manage textual RE artifacts. A graph browser is integrated to visualize different RE perspectives. CREST employs Notes Documents to store and exchange hypermedia/multimedia data. Informal messages, such as email messages and discussion notes, are managed by native Notes applications which are integrated with CREST. Because Lotus

Notes/Domino has such a huge user community, the existing user expertise in Lotus Notes is exploited when using CREST. By sharing data and perspectives, CREST supports continual and incremental analysis across team members.

In addition to the reverse engineering functionalities, CREST also implements a set of collaborative features that are not well supported by most reverse engineering tools. With CREST, team members can exchange current perspectives, assign RE tasks, and discuss problems on-line.

Although CREST implements some RE functionalities including CSCW support, it is not as loaded with as much RE functionality as Rigi is for example. However, CREST shows that building RE features on top of Lotus Notes/Domino is feasible and practical. To make such a tool really adoptable by industry, significantly more work is still needed.

6.2 Comparison

With respect to non-functional characteristics, CREST has the following advantages over Rigi reverse engineering environment.

Firstly, CREST has a broader application scope than Rigi. Rigi is a stand-alone tool. Its data and perspectives are not readily and easily shared among different users. CREST is based on the client/server model. All data are stored on the server side. Different users can easily share the RE data and intermediate perspectives. They can also view, modify and comment on the perspectives. CREST not only enables users to work on the same task simultaneously, but also facilitates collaboration among team members.

Secondly, CREST has more potential users than Rigi. The Rigi manual [Wong98] includes 168 pages. To get familiar with its operations, users definitely

need more instruction and training. CREST is built on top of Notes, which has many more end users who already have significant expertise using the Notes UI and metaphor. These users can pay more attention to RE specific operations, instead of spending much time on the UI and fundamental features. Hopefully, it is more likely that CREST be accepted by Lotus/Notes users than Rigi for example.

Thirdly, CREST employs a highly structural and indexed data repository supporting multiple data formats. Rigi uses RSF as the data storage format. RSF is actually a plain text format. Although RSF is human-readable, it is still difficult to locate data items efficiently because an RSF file may consist of too many triplets. Therefore, an RSF file can not be directly understood and manipulated by users. CREST represents data with Notes Documents, which support type fields and multimedia data. The data are structured using different Notes Views. Therefore, the data are not only readable, but also well organized. Moreover, these Documents are indexed. Lotus Notes/Domino provides a built-in search and index engine. As a result, the operation of locating data is greatly accelerated.

Fourthly, CREST includes more extensive interoperability capabilities than Rigi. Rigi mainly employs two ways to interact with other applications: RSF is used to exchange data; Tcl/Tk allows control integration. Tcl/Tk is not as popular as LotusScript. In contrary, CREST employs more recent, more popular and up to data interoperation techniques. XML-based GXL is used as a data exchange format. SVG, used to transfer intermediate perspectives, is also widely supported. Based on Lotus Notes/Domino, CREST affords various interoperation mechanisms: Java, LotusScript, C/C++ API, COM and CORBA. It even supports the latest and greatest technique—Web services.

Fifthly, the effort of developing CREST is much less than that of building Rigi. Rigi was constructed from scratch. Even though the time and effort spent in architecting and designing Rigi are not taken into account, there was still a great amount of work spent in coding and testing. The source code of Rigi contains apprximately 30,000 lines of code. With respect to CREST, there are 4029 lines of Java code and 740 lines of LotusScript code. Among them, 3272 lines of Java and 358 lines of LotusScript were written as reusable libraries for ACRENotes Framework. A big benefit is that developers of new applications do not need to implement the UI and other basic components. This saves significant amounts of time and effort. Moreover, by using the ACRENotes template, the implementation work is greatly simplified.

CREST has some drawbacks. The first one is related to data integration. So far, there is no native parser available within CREST. As a result, users can not directly handle source code. Parsers should be added to extract artifacts from source code and covert them into GXL format data, allowing CREST to use data directly. Rigi, on the other hand, has several parsers to analyze code of several programming languages, including C, C++, Java, COBOL and PL/I. Scalability is yet another problem within the ACRENotes Framework. This problem does not occur in the data layer because a Notes Database can hold millions of Notes Documents. The scalability problem originates from the graph browser in the presentation layer. The graph browser can not support many nodes in one graph because JGraph does not render complex graphs efficiently and effectively. Significant development work has gone into Rigi to make it scalable. Figure 6.1 shows the graph representation of a Ray Tracing system, which is a demo program included with Rigi. This graph has 261 nodes and 282 edges. Function names make the graph messy because they are rather long. Therefore, node ids are shown in the graph. Compared to the graph in Figure 5.8, the response time for

manipulating this graph is apparently slower, but still acceptable. In contrast, Rigi can handle more than 5,000 nodes in its graph editor with a short response time. Another problem is related to extensibility. Although Lotus Notes/Domino provides a variety of customization mechanisms, extensibility of CREST is still confined to the context of Lotus Notes/Domino. It is not as flexible as applications developed from scratch.

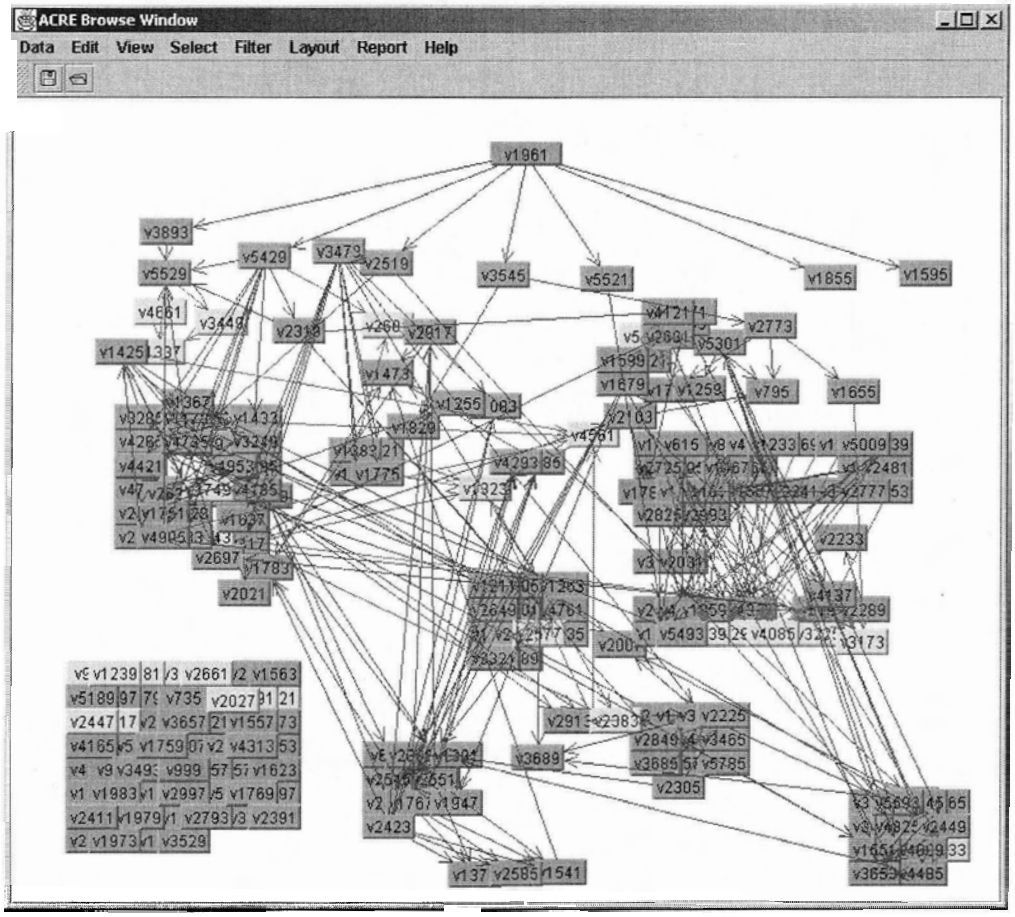

Figure 6.1: Ray Tracer demo in CREST

6.3 Good Practices and Lessons Learned

In my research, I identified a few good practices and lessons and offering future researchers some useful advice. These practices and lessons are categorized into two groups. One deals with technical issues and the other with the methodology.

The identified good technical practices are the following:

- Java Agent: Java Agent is the most powerful mechanism to extend functionality in Lotus Notes/Domino. A good practice when programming with Java Agent is to separate the kernel logic classes from Java Agent itself. The classes can be packed in jar files and reused in different Java Agents. Therefore, Java code becomes more reusable and easier to maintain. A small trick is that the Java classpath must be declared in the JavaUserClasses entry of the Notes.ini file, rather than the environment variable.
- Script library: It is helpful to collect commonly used functions in one or more script libraries. Because functions in script libraries are reusable, a great amount of programming work can be saved. Users ought to continuously re-factor functions in libraries to make them more useful.
- Glue code: To make ACRENotes applications fit with other Notes applications, some glue code is required. Actually, this code is based on the same mechanism: opening backend Notes databases, creating a backend document object, filling the document with data and opening its front-end UI object.

In the process of developing CREST, I encountered several unresolved issues:

- Java/Notes two-way interaction: the graph browser of CREST can freely manipulate Lotus Notes/Domino backend Document objects with Java. But it has no capability to control the UI objects of the Lotus/Domino. In other words, it is impossible for the browser to execute any UI-related operations, such as open a View in the current Lotus Notes workspace. It is because the Java interface of Lotus Notes/Domino does not expose any UI objects at all. Thus, from the perspective of end users, the graph browser and the Lotus Notes workspace may not be perfectly integrated or synchronized. So far,

there is no good solution for this problem. A possible answer is to use Java bridge techniques to access Lotus Notes/Domino's COM objects. However, this solution is quite heavyweight. The best expectation is that IBM/Lotus provides such an interface for UI objects in the next release. Insufficient support of the baseline platform is often the most annoying problem during COTS based development.

With respect to methodology, I found that it is good to

- ✓ Build a prototype application as a "trace bullet": At the beginning of my research, I did not clearly know our goals and the support afforded by Lotus Notes/Domino. A prototype application can help clarify and refine ambiguous questions step by step.
- ✓ Apply a layered architecture: The layered architecture clearly defines the scope and responsibility of its components. It is good approach when integrating different applications.
- ✓ Leverage existing research experience: On the one hand, our research group has much experience with Rigi. On the other hand, the time and resources are limited. So, we use Rigi as the base stepping stone to extract requirements, initiate ideas and conduct comparisons. This method does save time and effort.

We also identified a problem of negative effect with respect to existing experience. Relying too much on existing experience confines and constrains the minds of the designers. For instance, we implemented the prototype as a visualization tool. However, a visualization tool may not be the best tool that can be implemented with the ACRENotes Framework.

6.4 Summary

The CREST prototype shows that it is feasible and practical to build reverse engineering tools on top of Lotus Notes/Domino. CREST not only implements RE features similar to Rigi, but also supports collaborative work. Compared to Rigi, CREST has broader scope, a better data repository, stronger interoperability capabilities, more potential users, and significantly less development effort. However, it also has drawbacks with respect to performance and extensibility. A set of practices, lessons and problems I found during my research are also articulated. These practices and lessons may constitute useful advice for future researchers.

Chapter 7 Conclusions

7.1 Summary

The goal of the ACRE project is to extend a baseline platform or COTS product to create more adoptable RE tools. As part of the project, my research concentrated on the solution of constructing RE tools on top of Lotus Notes/Domino.

I firstly introduced some background knowledge for the RE domain, especially the ACRE project and reviewed related research regarding CBS.

Next, the requirements for RE tools were discussed. These requirements also include collaboration in software maintenance tasks. Moreover, the support and benefits provided by Lotus Notes/Domino were articulated. I also analyzed the problem of how to leverage these benefits and avoid barriers.

The core of this thesis, the ACRENotes Framework, was proposed. It involves three layers: data, control, and presentation. Quite a few technical methods and practices are included in this framework.

Based on this framework, the prototype CREST was implemented. The effectiveness of the ACRENotes Framework was validated by checking whether this prototype meets the requirements.

By comparing CREST with Rigi, I found that CREST did have more advantages than Rigi, including broader scope, a better data repository, stronger

interoperability, and so on. Some good practices and lessons learned during my research were collected and documented as hints for further research.

In general, my experience is largely positive. I believe that it is feasible to build RE tools on top of Lotus Notes/Domino. The built RE tool has RE features similar to the traditional Rigi reverse engineering tool. It includes additional features, such as CSCW, which are not part of Rigi. However, the research is far from complete. A lot of work is left for future research.

7.2 Contributions

I can not claim that I have completely solved the adoption problem of RE tools. But I believe that I did contribute to the solution of this problem. The following list shows my major contributions to the research in this specific domain.

- Investigated the support, benefits and drawbacks of Lotus Notes/Domino.
- Proposed a solution—the ACRENotes Framework, a three layer architecture containing a set of methods and practices.
- Designed and implemented a prototype called CREST to validate the feasibility of the ACRENotes Framework.
- Performed a comparison between CREST and Rigi to identify CREST's cons and pros.
- Collected good practices and lessons learned in the process of my research and development.

7.3 Future Work

7.3.1 Case Study

At this point CREST is basically ready for an industrial case study. A case study is useful because feedback from users is valuable for demonstrating the effectiveness of an ACRE tool. We are particularly interested to figure out which features of the native tool and which features of the extended tool are utilized to conduct specific reverse engineering tasks. Moreover, it would be worthwhile to quantify the cognitive support leveraged from the host tool while performing a set of reverse engineering tasks.

However, due to obvious obstacles it is not easy to conduct such a case study. Users have different levels of expertise with the baseline platform. Participants might also have different capabilities with respect to mastering a new tool. Some features, such as collaboration, are also difficult to evaluate. We have to figure a good strategy for such a case study to minimize the negative effects of these difficulties and isolate factors as much as possible.

7.3.2 Improving Real-time Collaboration

By now, the ACRENotes Framework supports collaboration among team members of a reverse engineering project well. However, it is not perfect. Basically, the current collaboration capabilities are based on asynchronous communication. Nowadays, instant communication is more and more preferable in people's daily life and work in part due to the instant messaging phenomenon. The great success of the ICQ and MSN Messengers is a good evidence to this effect. Instant communication, for example instant messaging and real-time whiteboarding, can greatly facilitate team work. We would like to introduce this feature into the ACRENotes Framework in the future. Thanks to the instant communication product of IBM/Lotus—Sametime, the

integration effort would probably be manageable for a research team. Fortunately, Sametime can be seamlessly integrated with Notes and provides a powerful Java API for developers.

7.3.3 Building Parsers

At this point, ACRENotes applications have to retrieve GXL data from some external parsers. This is not an ideal solution for normal users because the process for importing data is rather complicated. The source code of a system is analyzed by a parser; the generated data is converted to GXL format; finally, the GXL data is imported into a Notes Database. This process not only involves installing external parsers, but also a great amount of user involvement. Because the data are not kept synchronized with the source code, this imported data might be out of date. Moreover, this process is also error-prone. To solve this problem, a parser that directly interacts with ACRENotes applications is needed. Using the JavaCC technique [JCC04], parsers for different programming languages could be implemented for ACRENotes. Because the implementation of these parsers is 100% pure Java, the parsers can be easily invoked via the Java Agent in ACRENotes applications. For end users, the method is automatic, programmable and easy. Source code would be automatically parsed by Java Agent and directly imported into a Notes Database.

7.3.4 Other Issues

There are still other issues worth investigating. A potential problem concerns the application of the ACRENotes Framework to other domains. I believe that the idea of building tools on top of Notes can be applied in contexts other than reverse engineering. According to my personal experience, the ACRENotes Framework is suited for example for requirements engineering. Some important goals of

requirements engineering includes: effectively manage requirements documents, precisely capture changes in requirements, and understand the requirements. The ACRENotes Framework's capability in managing documents could be used to manipulate requirements documents. The CSCW support of the ACRENotes Framework could facilitate the communication between customers and analysts. Therefore, it would be worthwhile to apply the ACRENotes Framework in a different software engineering context.

Bibliography

[AAA03] O. Andersson, P. Armstrong and H. Axelsson, Scalable Vector Graphics (SVG) 1.1 Specification, http://www.w3.org/TR/SVG, 2003

[AAF03] Adobe, Acrobat Family, http://www.adobe.com/products/acrobat/main.html, 2003

[AFM04] Adobe, Adobe FrameMaker 7.1, http://www.adobe.com/products/framemaker/main.html, 2004

[Ald04] G. Alder, the Home Page of JGraph, http://www.jgraph.com/, 2004

[BaB01] V. Basili and B. Boehm. "COTS-Based Systems Top 10 List," *Computer*, Vol. 34, pp. 91-93, May 2001

[BaH96] T. Bao and E. Horowitz, "Integrating Through User Interface: A Flexible Integration Framework for Third-Party Software," *Proceedings the 12th Annual International Computer Software And Applications Conference*, pp. 336-342, Toronto, Canada, August 1996.

[BaK00] L. Balk and A. Kedia, "PPT: a COTS integration case study," *IEEE the 22th International Conference on Software Engineering (ICSE-00)*, pp. 42-49, Limerick, Ireland, 2000

[BrS95] M.L.Brodie and M.Stonebraker, "Migrating Legacy Systems: Gateways, Interfaces & The Incremental Approach," Morgan Kaufmann, 210 pages, 1995.

[CBI03] SEI, COTS-Based Systems (CBS) Initiative, http://www.sei.cmu.edu/cbs/, 2003

[CBO03] SEI, CBS Overview, http://www.sei.cmu.edu/cbs/overview.html, 2003

[ChC90] E.Chikofsky and J.Cross. "Reverse engineering and design recovery: A taxonomy," *IEEE Software*, Vol. 7, pp. 13-17, January 1990.

[DMD01] S.DeRose, E.Maler, and D.Orchard, " XML Linking Language (XLink) Version 1.0", http://www.w3.org/TR/xlink/, June 2001.

[Eas93] A. Eastwood, "Firm fires shots at legacy systems", Computing Canada, Vol. 19, pp. 17, 1993

[EgB01] A. Egyed and R. Balzer, "Unfriendly COTS Integration— Instrumentation and Interfaces for Improved Plugability," *Proceedings the 16th IEEE International Conference on Automated Software Engineering*, pp. 223-231, San Diego, USA, November 2001

[Gia99] G. Giaglis, "Focus issue on legacy information systems and business process engineering: on the integrate design and evaluation of business process and information systems," *Communications of the AIS*, Vol. 2, No. 5, July 1999.

[GoB99] N. Goldman and R. Balzer, "the ISI Visual Design Editor Generator," *Proceedings the IEEE Symposium on Visual Languages*, pp. 20, Tokyo, Japan, September 1999

[HeC90] J. Henderson and J. Cooprider, "Dimensions of I/S Planning and Design Aids: A Functional Model of CASE Technology," *Information Systems Research*, Vol. 1, pp. 227-254, 1990.

[HMK03] H. Müller, M. Storey and K. Wong, ACSE Project Synopsis, http://www.acse.cs.uvic.ca/index.html, October 2003.

[HoW00] R. Holt and A. Winter, "A Short Introduction to the GXL Exchange Format," *Proceedings the 7th Working Conference on Reverse Engineering (WCRE'00)*, pp. 299, Brisbane, Austrilia, November 2000.

[HuM04] J. Hunter and B. McLaughlin, JDOM, http://www.jdom.org/index.html, 2004

[I3E83] IEEE Computer Society, "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Standard 729-1983*: The Institute of Electronical and Electronics Engineers, Inc.: New York, 1983.

[I3E98] IEEE Computer Society, "IEEE. Standard for Software Maintenance," *IEEE Std. 1219-1998*: The Institute of Electrical and Electronics Engineers, Inc.: New York, 1998.

[ILD03] IBM, Lotus Domino, http://www.lotus.com/products/product4.nsf/wdocs/dominohomepage, 2003

[ILN03] IBM, Lotus Notes, http://www.lotus.com/products/product4.nsf/wdocs/noteshomepage, 2003

[ILS03] IBM, Lotus SmartSuite, http://www.lotus.com/products/smrtsuite.nsf/wPages/smartsuite?OpenDocument, 2003

[IMA03] Imagix, Imagix 4D, http://www.imagix.com/products/products.html, 2003

[JCC04] JavaCC, https://javacc.dev.java.net/, 2004

[KLI03] Klocwork, Insight, http://www.klocwork.com/products/insight.asp, 2003

[LDH03] Lotus, Domino Administrator Help, http://doc.notes.net/uafiles.nsf/docs/Domino6PR2/$File/help6_admin.exe, 2003

[Leh96] M. M. Lehman, "Laws of Software Evolution Revisited," *Proceedings of the 5th European Workshop on Software Process Technology*, pp.108-124, Nancy, France, October 1996

[LoR93] R. Lougher and T. Rodden. "Supporting long-term collaboration in software maintenance," Proceedings the Conference on Organizational computing systems, pp. 228-238, Milpitas, USA, November 1993.

[MJS00] H. Müller, J. Jahnke, D. Smith, M. Storey, S. Tilley and K. Wong, "Reverse Engineering: A Roadmap," *Proceedings Future of Software Engineering*, pp. 47-60, Limerick, Ireland, June 2000.

[MOF03] Microsoft, Microsoft Office Online Home Page, http://office.microsoft.com/home/default.aspx, 2003

[MWT94] H. Müller, K. Wong, and S. Tilley, "Understanding software systems using reverse engineering technology," *Proceedings of the 62nd Congress of L'Association Canadienne Francaise pour l'Avancement des Sciences (ACFAS '94)*, pp. 41-48, Montreal, Canada, May 1994.

[New94] M. Newby, "Legacy systems, software maintenance and computing curricula," *Proceedings Software Education Conference*, pp. 96-102, Dunedin, New Zealand, November 1994.

[RGH03] Rigi, Rigi Group Home Page, http://www.rigi.cs.uvic.ca, 2003

[RiB03] H. Müller, Rigi, http://www.rigi.csc.uvic.ca/rigi/blurb/Blurb.html, 2003

[RSS01] R. Holt, A. Schürr and S. Sim, GXL—Document Type Definition (DTD), http://www.gupro.de/GXL/dtd/dtd.html, 2003

[SiL04] R. Slapikoff and R. Lipton, The ABC's of Using the ACL, http://www-10.lotus.com/ldd/today.nsf/0/be08e4acfc72cd72852565d9004cb61c?OpenDocument, 2004

[SNI04] Wind River, SNiFF+, http://www.windriver.com/products/development_tools/ide/sniff_plus/, 2004

[SOO03] Sun, OpenOffice.org, http://wwws.sun.com/software/star/openoffice/, 2003

[Spi02] D. Spinellis. "Unix tools as visual programming components in a GUI-builder environment," *Software–Practice and Experience*, Vol. 32, pp. 57-71, January 2002.

[Sta84] T. A. Standish, "An essay on software reuse," IEEE Transactions on Software Engineering, SE-10(5), pp. 494-497, September 1984.

[StM97] M. Storey and H. Müller, "Rigi: A Visualization Environment for Reverse Engineering," *Proceedings International Conference on Software Engineering (ICSE-97)*, pp. 606-607, Boston, Massachusetts, May 1997

[Til94] S. Tilley, "Domain-retargetable reverse engineering. II. Personalized user interfaces," Proceedings International Conference on Software Maintenanceon, pp. 336 -342, Victoria, Canada, September 1994.

[Til95] S. Tilley, "Domain-retargetable reverse engineering. III. Layered modeling," Proceedings International Conference on Software Maintenanceon, pp.52 -61, Nice, France, October 1995.

[Til98] S. Tilley, "Coming attractions in program understanding II: Highlights of 1997 and opportunities of 1998," Technical Report CMU/SEI-98-TR-001, Carnegie Mellon Software Engineering Institute, February 1998.

[TMW93] S. Tilley, H. Muller, M. Whitney, K. Wong, "Domain-retargetable reverse engineering," Proceedings International Conference on Software Maintenance, pp. 142-151, Montreal, Canada, September 1993

[VaK95] G.Valetto, G.E.Kaiser, "Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments," Proceedings of 7th IEEE International Workshop on CASE, pp. 40-48, Toronto, Ontario, Canada, July, 1995.

[Wong98] K. Wong, "The Rigi User's Manual - Version 5.4.4, " http://ftp.rigi.csc.uvic.ca/pub/rigi/doc/rigi-5.4.4-manual.pdf, June 30, 1998.

[Wong99] K. Wong, "Reverse Engineering Notebook, " PhD Thesis, Department of Computer Science, University of Victoria, October 1999.

[WTM95] K. Wong, S. Tilley, H. Müller and M. Storey, "Structural redocumentation: a case study, " *IEEE Software*, Vol. 12, pp. 46-54, January 1995.

[WuS00] J. Wu and M. Storey, "A Multi-Perspective Software Visualization Environment," Proceedings IBM CAS Conference 2000 (CASCON 2000), pp. 41-50, Toronto, Canada, November 2000.

Appendix A: AXL Sample

The sample AXL

```
<?xml version="1.0" encoding="UTF-8"?>
<ACREView:View xmlns:ACREView="http://www.rigi.uvic.ca/~acre"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.rigi.uvic.ca/~acre ACREView.xsd">
<SubView>
    <NodeView NodeID="v14">
        <PosX>376</PosX>
        <PosY>244</PosY>
        <SizeWidth>58</SizeWidth>
    </NodeView>
    <NodeView NodeID="v13">
        <PosX>313</PosX>
        <PosY>103</PosY>
        <SizeWidth>88</SizeWidth>
    </NodeView>
    <NodeView NodeID="v12">
        <PosX>215</PosX>
        <PosY>181</PosY>
        <SizeWidth>92</SizeWidth>
        <SizeHeight>22</SizeHeight>
    </NodeView>
```

```
<NodeView NodeID="v11">
        <PosX>200</PosX>
        <PosY>100</PosY>
        <SizeWidth>83</SizeWidth>
        <SizeHeight>21</SizeHeight>
</NodeView>
<NodeView NodeID="v9">
        <PosX>60</PosX>
        <PosY>244</PosY>
</NodeView>
<NodeView NodeID="v10">
        <PosX>426</PosX>
        <PosY>49</PosY>
        <SizeWidth>90</SizeWidth>
        <SizeHeight>22</SizeHeight>
</NodeView>
<NodeView NodeID="v8">
        <PosX>104</PosX>
        <PosY>100</PosY>
        <SizeWidth>64</SizeWidth>
        <SizeHeight>23</SizeHeight>
</NodeView>
<NodeView NodeID="v7">
        <PosX>200</PosX>
        <PosY>47</PosY>
```

```
        <SizeWidth>74</SizeWidth>
        <SizeHeight>24</SizeHeight>
</NodeView>
<NodeView NodeID="v6">
        <PosX>0</PosX>
        <PosY>100</PosY>
</NodeView>
<NodeView NodeID="v5">
        <PosX>112</PosX>
        <PosY>303</PosY>
</NodeView>
<NodeView NodeID="v4">
        <PosX>24</PosX>
        <PosY>301</PosY>
</NodeView>
<NodeView NodeID="v3">
        <PosX>313</PosX>
        <PosY>49</PosY>
        <SizeWidth>77</SizeWidth>
        <SizeHeight>22</SizeHeight>
</NodeView>
<NodeView NodeID="v2">
        <PosX>100</PosX>
        <PosY>50</PosY>
        <SizeWidth>73</SizeWidth>
```

```
        </NodeView>
        <NodeView NodeID="v1">
                <PosX>249</PosX>
                <PosY>2</PosY>
        </NodeView></SubView>
</ACREView:View>
```

Appendix B: Domain Schema

```
<?xml version="1.0"?>
<schema xmlns=http://www.w3.org/2001/XMLSchema
targetNamespace=http://www.rigi.uvic.ca/~acre/domain
xmlns:ACREDomain="http://www.rigi.uvic.ca/~acre/domain">
    <complexType name="DomainElements">
        <sequence minOccurs="1" maxOccurs="1">
            <element name="NodeTypes"
            type="ACREDomain:NodeTypeGroup" minOccurs="1"
            maxOccurs="1"/>
            <element name="ArcTypes"
            type="ACREDomain:ArcTypeGroup" minOccurs="1"
            maxOccurs="1"/>
        </sequence>
    </complexType>
    <element name="Domain" type="ACREDomain:DomainElements"/>
    <complexType name="NodeTypeGroup">
        <sequence minOccurs="1" maxOccurs="unbounded">
            <element name="NodeType" type="string"/>
        </sequence>
    </complexType>
    <complexType name="ArcTypeGroup">
        <sequence minOccurs="1" maxOccurs="unbounded">
            <element name="ArcType" type="string"/>
```

```
            </sequence>
        </complexType>
    </schema>
```

Appendix C: CREST Design Elements

Category	Sub-Category	Item Name	Description
Frameset		MainFrameSetNotes	The main frame structure of the UI
Page		Database Location	Show the location information of the Database
		OutlinePage	Render the outline structure in one page
Form		DomainSchema	Create or modify the domain information
		Node	Render or modify Node detail Information
		Edge	Render or modify Edge detail Information
		REView	Render or modify ACREView Information
		REViewSVG	Store the SVG output
		TemplateName	Show the Database name
View		Domain Schema	Show the currently used schema
		Domain Schema\Available Schemas	Show all available schemas

		(IN_Domain_Schema)	Hidden View, used in coding
		REData	Used by DataManager to manipulate all RE data
		Node	Group all Node Documents together
		Edge	Group all Edge Documents together
		Edge\From	Sort all Edge Documents by the From attribute
		Edge\To	Sort all Edge Documents by the To attribute
		ACREView	Group all ACREView Documents
Share Code	Agent	ImportGXL	Import RE data from GXL
		ShowGraph	Activate the graph browser to visualize ACREView
		DomainConfig	Change the domain schema
	Outline	MainOutline	Structure all visible design items
	Shared Actions	GrantDataReader	
		GrantViewReader	
		RevokeDataReader	
		RevokeViewReader	

	Script Library	ACREMessage	Support collaborative activities
		RERigi	Support RE operations