Customizing Lotus Notes to Build Software Engineering Tools

Jun Ma, Holger M. Kienle, and Piotr Kaminski
University of Victoria
Victoria, Canada
{majun,kienle,pkaminsk}@cs.uvic.ca

Anke Weber
ExperEdge Technology Partners
Victoria, Canada
weber@experedge.com

Marin Litoiu
IBM Toronto Labs
Toronto, Canada
marin@ca.ibm.com

Abstract

Many software engineering research tools are stand-alone applications that have trouble interoperating with other development tools and do not fit well into the software developers' established work processes. Our main hypothesis is that in order for new tools to be adopted effectively, they must be compatible with both existing users and existing tools.

Typically, software engineering teams in an organization share a set of common applications for their development activities that are a permanent part of each developer's everyday workflow. Among these applications are shrink-wrapped office tools such as Lotus Notes, which are used for, among other tasks, email, scheduling, and project reports and presentations. These office tools, are highly integrated and offer a mature, well-tested working environment, which can be customized to an extent that allows to provide support for advanced software engineering tasks.

This paper describes RENotes, a reverse engineering tool built by customizing Lotus Notes. RENotes targets software developers who use Notes as part of their work environment. We describe Notes' features and how they can be leveraged to layer new reverse engineering functionality on top.

Keywords: Lotus Notes, customization, end-user programmable systems, tool adoption, collaboration, Rigi

1 Introduction

It takes a lot of effort to go from the conceptual design for a new software engineering technique to the development of tools that supports the technique and finally adoption of the tool in industry. Researchers that strive to have their tools adopted struggle to develop tools that satisfy the requirements that software engineers in industry place on them. Common examples of adoption hurdles include difficult installation, lack of documentation, unpolished/awkward user interfaces, and poor interoperability with existing tool infrastructure. As a result, most research tools require a significant learning curve while disrupting the established work process of the software engineer.

Tool development is a significant investment and should focus on the novel features of the tool. Unfortunately, the development of a baseline environment, albeit often trivial to accomplish, requires significant effort before tool-specific functionality can be tackled. This is especially true for GUI-based tools that use a visual manipulation paradigm.

In this paper, we outline a software development approach that leverages a widely-used, shrink-wrapped office tool—Lotus Notes—by building a software reengineering tool, RENotes, on top of it. Notes is the *host application* that provides the baseline environment, including features such as document-based database, collaboration, search, and security. RENotes leverages Notes to provide software reengineering functionality such as artifact filtering and graph manipulation.

We believe that tool implementations that follow this approach have desirable features from the user and developer point of view. Spinellis draws a similar conclusion for the field of visual programming tools [21]:

> "As many visual programming environments are research-oriented, proof-of-concept projects, they cannot easily compete with the commercially-developed, polished, and supported commercial IDEs. 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."

The reminder of the paper is organized as follows. Section 2 gives more background information that guides our research. RENotes, our case study, implements part of the functionality of the Rigi reverse engineering tool. Therefore, Section 3 gives a brief introduction of Rigi and its functionality. In Section 4 we further introduce Notes as the host application for the RENotes case study. We first analyze the baseline environment that Notes provides and then discuss in Section 5 in detail how we customized Notes to build RENotes. Section 6 reviews related work. Finally, Section 7 summarizes our development experiences and RENotes' potential of improved adoption.

2 Background

This section discusses what we mean by tool customization and categorizes both the so-called host applications and the targeted users in more detail in order to provide a better understanding of the requirements of our research.

2.1 Tool Customization

A prerequisite for our proposed tool development method is that the host tool offers sophisticated customization mechanisms. (Such tools have been also referred to as user-tailorable computer systems [14].)

Support for customization can be divided into non-programmatic and programmatic customization mechanisms. Non-programmatic customization is accomplished, for example, by editing parameters in startup and configuration files or with direct manipulation at the GUI level. Programmatic customization involves some form of scripting or programming language that allows the modification and extension of the application's behavior. Programmatic customization is more powerful, but requires a significant effort. There is an initial cost in acquiring the necessary customization skills, followed by development and maintenance costs of the customization.

It is an interesting question to what extent users customize their applications. Page et al. studied the customization changes that users made to the WordPerfect word processor [18]. A surprising 92 percent of users in their study did some form of customization, with 63 percent using macros. They summarize their findings with "users who most heavily used their systems have the highest levels of customization" and "customization features that were simple to adapt (like the Button Bar) tended to have higher incidences of tailoring." The above study suggests that tool builders should expect that users want to customize their software; thus, tools should offer extensive customization support while making it simple and fast to use.

An early commercial product that allowed users to create customized applications was HyperCard, thus making "everybody a programmer" [8]. The Emacs text editor can be customized both by parameter setting and by programming in Emacs Lisp. Similarly, the AutoCAD system can be customized with AutoLisp. An example of a highly customizable research tool is Rigi [24] [23]. The Rigi graph editor allows customization by exposing its graph model and user interface with a Tcl/Tk API. Furthermore, Rigi graphs have a generic model that can be customized to different domains. Specification files describe the kinds of nodes and arcs that constitute a certain domain.

Tool builders have recognized the importance of making their software customizable and by now many popular tools (such as Microsoft Office, Lotus Notes, Adobe Acrobat, and Macromedia Dreamweaver) offer scripting support and APIs to accomplish customization.

This paper focuses on the customization of Notes. The feasibility of customization of Notes has been demonstrated by a number of large-scale projects. One such effort reports the following experiences [12]:

> "Users and developers have been positive about Notes. Advantages cited include the relative ease of learning how to develop basic applications, the relatively short development cycle—from a few days to a couple of weeks per applications—and the fact that users' suggestions and feedback can be incorporated into the applications with relative ease and can be done on a continuous basis."

2.2 Host Applications

The development approach that we describe grafts domain-specific functionality on top of highly customizable tool foundations—we call these tools host applications.

There is a broad range of candidates for host applications. In our current research, we focus on office suites (e.g., Microsoft Office, Lotus SmartSuite, and OpenOffice) and productivity tools (e.g., Lotus Notes and Microsoft Project). From the programmer's point of view, these tools have commercial-of-the-self (COTS) characteristics. Other promising candidates are extensible IDEs (e.g., Eclipse).

We classify systems that use COTS components based on the scheme proposed by Carney [2]:

turnkey: These systems use a single COTS component on which they heavily depend. Typically, customization is quite limited and non-programmatic.

intermediate: These systems are also built on a single COTS component, but also "have a number of customized elements specific to the given application." The amount of customization can vary, but does not fundamentally change the nature of the employed COTS component and results in a moderate amount of newly developed customization code.

mixed: These systems contain several (heterogeneous) COTS components to provide large-scale functionality that is otherwise not available. They have a significant amount of glue code and are often difficult to develop and maintain.

In our current research, we target intermediate systems (i.e., only a single host application is selected and customization is done programmatically).

2.3 Target Users

Another important consideration are the users that the application targets. Karsten [11]reviewed 18 case studies of organizations' use of Notes and splits them into three groups: exploratory, expanding, and extensive use of Notes. Organizations with extensive use of Notes had the following commonalities [11]:

> "In all these cases, there were several applications that were tied directly into established work practices. Application development was conducted as a careful process, with prototypes and user involvement."

Thus, in the best case, the host application is a fundamental part of the users' work processes. Evidence of such a mission-critical tool are local developers ("gardeners") that work on customizations for their group. This has been observed, for example, for CAD systems [6]. An other example is the IBM Toronto Lab, which has a dedicated development group to customize Notes.

Host applications that are based on familiar office tools provide a number of potential benefits to users:

a familiar GUI: The user interacts with a familiar environment and paradigm. Application knowledge (a.k.a. cognitive support [26]) has been typically built up by the user over years. Since the users are already familiar with the standard functionality, they can concentrate on learning the additional, domain-specific functionality (incrementally).

tool interoperability: Office tools interoperate among each other via cut-and-paste and (file-based) import/export facilities.

tool support: Popular tools come with a large infrastructure that provides useful information to the user. For example, (on-line) publications discuss how to use a tool most effectively. Mailing lists and discussion forums help troubleshoot users' problems.

Stand-alone research tools are typically found lacking in all of the areas outlined above.

2.4 Selection of Host Applications

Selection of a host application is a trade-off decision. When deciding on a suitable host application, one typically has to choose among several possible candidates. A host application has to satisfy two main criteria: it has to (1) provide a suitable baseline environment for extension, and (2) be familiar to its target users. The former criterion shortens the development life cycle while the latter can accelerate the adoption. Sometimes, these two criteria may conflict with each other. Our host application for RENotes is Lotus Notes. The decision to select Notes was based on its large user base in companies and its customization flexibility. Although we did not formally evaluate the trade-offs, our observations and discussions with developer at the IBM Toronto Lab hold up our assumptions. Notes appears to be the most pervasive application within the IBM Toronto Lab. Besides its use as a group and peer-to-peer communication infrastructure, Notes is the host application for tens of customized applications used by the Lab employees. About seven developers in the Lab are currently involved in development and maintenance of these applications.

3 The Rigi Reverse Engineering Tool

In order to gain experiences and validate our tool development approach, we decided to build a reverse engineering tool on top of Notes that is similar to Rigi as a case study. Rigi has been under development for over a decade in our research group at the University of Victoria. Because of our previous tool building experience with Rigi, we are already familiar with the application domain and can focus on understanding customizations with Notes.

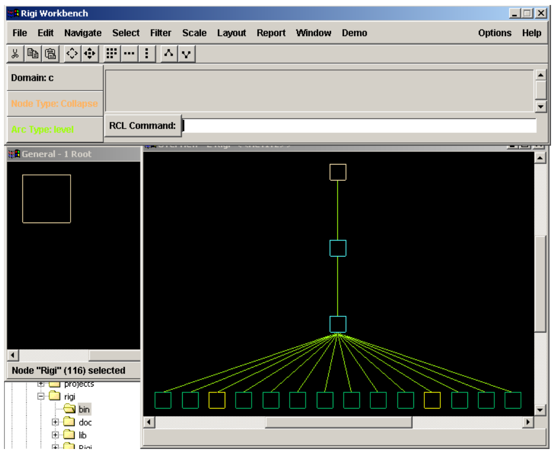


Figure 1: The Rigi reverse engineering tool.

Rigi is an interactive, visual tool designed for program understanding and software re-documentation. Figure 1 shows a snapshot of Rigi. The core of Rigi is a generic graph editor enhanced with domain-specific functionality for reverse engineering tasks. Rigi uses typed, directed graphs to convey information about the target software system. Nodes of the graph represent artifacts of the system (e.g., functions, variables, and types) and directed arcs represent artifact relationships (e.g., functions calls, assignments to variables, and type declarations). A simple example of a Rigi view is the subject system's call-graph. Nodes in the call-graph represent functions of the program and arcs represent calls between functions. Different kind of nodes and arcs are visualized with different colors. Rigi offers operations to select and filter the arcs and nodes in a graph and allows to apply several graph layout algorithms.

Even though Rigi can be customized with Tcl scripting, it is a stand-alone application that is not easy to integrate with other tools. Most notably, short of taking screenshots, Rigi graphs cannot be exported to other applications. This is a severe drawback, because Rigi graphs are the main work products of the reverse engineering activity, usually becoming part of the system documentation [27].

Leveraging office tools to build reverse engineering functionality seems a promising approach because the reverse engineering process for a larger legacy system is both document-intensive and collaborative. An important problem is the task of organizing and archiving of obtained reverse engineering results, so that they are available for future system evolution. As described in Section 4, Notes has features that address these problems.

4 Lotus Notes/Domino

Lotus Notes/Domino is a popular groupware product that is used by many organizations [10]. It is important to realize that Notes has more to offer than email support. In fact, it is often mentioned as the main software for supporting collaboration and its use is extensively studied by researchers in the area of computer supported collaborative work [11].

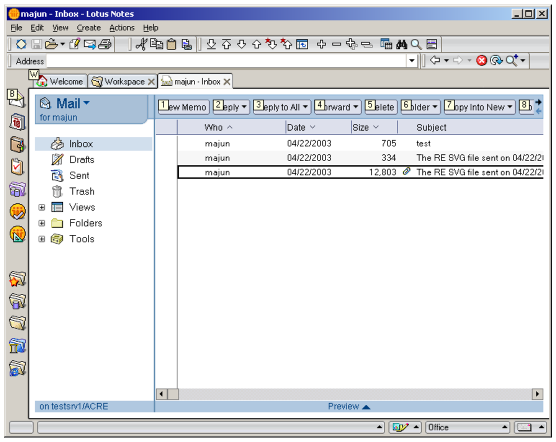
Based on the client/server model, Lotus Notes/Domino is often used to construct an internal information platform for an organization [4]. In this platform, all information is represented as documents, which are stored and organized in databases. Domino acts as the server, providing database access; Notes hosts the client applications that access the database. In Notes, shared databases are the backbone that enables collaboration among members in an organization.

For many organizations Notes is a critical part of their IT infrastructure. An example is Statoil, a Norwegian state-owned oil company [17]. Statoil has a full company license making it one of the world's largest users of Notes. Five years after its introduction in 1992, Notes had diffused to most of the company's 17000 employees, and its use was made mandatory. Statoil's geographical distribution (40-50 sites) makes collaboration-based features attractive. The following collaborative features of Notes are leveraged: email, document management, workflow, electronic archive, group calendar, news and bulletin boards, and discussion databases.

IBM itself is another example of a company that uses Notes extensively. From our observations at the IBM Toronto Labs, Notes has been customized for a wide variety of application, ranging from a simple carpool database to a project application that allows a development team to organize the work-products of their project. With the Domino Web server, Notes applications can be published on the Web. When an HTTP request accesses a database, Domino translates the document to HTML. Thus, certain applications can be made accessible with both Notes and a standard Web browser. At the Toronto Lab, Domino servers within the intranet allow Web access to applications.

4.1 Features

Before deciding on a host application, it is important to understand its key features. The host's baseline functionality and customization mechanisms determines its suitability for building domain-specific functionality on top of it.



Who	Date	Size	Subject
majun	04/22/2003	705	test
majun	04/22/2003	334	The RE SVG file sent on 04/22/2
majun	04/22/2003	12,803	The RE SVG file sent on 04/22/2

Figure 2: Lotus Notes mail application.

The user leverages the following features of Notes' baseline environment when using RENotes:

User interface: Notes has a mature (if idiosyncratic) user interface that has been consecutively refined over six major releases. Figure 2 shows a snapshot of the latest release, Lotus Notes 6. The user interface includes a menu bar, a bookmark bar, a tool bar, a status bar and a set of window tabs. Tabs make it easy for user to switch to different applications and databases. Most of the GUI elements can be customized. Furthermore, each tab has its own set of task-specific menu items and buttons. The mail tab, for example, has a button to compose a new email ("New Memo") and a pull down menu with several options on how to reply to the currently selected email ("Reply").

Document-based database (DBD): The documents in each database are organized with views and folders. In Notes, every document is a basic data unit that contains a set of data fields. In this respect, a document is similar to a record in a relational database; however, a document in Notes can be composed of an arbitrary number of fields. Since all documents adhere to this schema, it is possible to access, manage, and manipulate diverse documents in a uniform manner. Notes databases support many data types, including text, pictures, sound, video, file attachments, embedded objects, and applets.

Script automation: Users often take advantage of customization to automate work. A study about the customization behavior of 51 users in a Unix environment found that users "were most likely to customize when they discovered that they were doing something repeatedly and chose to automate the process" [13].

Notes provides users agent and actions to accomplish automation. Actions are integrated into the GUI and activated by users (e.g., by pressing a button). Agents, which run on the server, can be triggered by certain events (e.g., expiration of a timer).

Collaboration: Typical Notes applications are message exchange, information sharing, and workflow. By sending and receiving emails, members exchange messages in an organization. By accessing document in shared databases on servers, distributed information exchange, retrieval, storage, and consolidation is facilitated. Workflow applications guide users through certain tasks that they have to perform as part of their work. Such guidance can reduce overhead and mistakes, thus speeding up processes.

Search: Notes has automatic search capabilities as well as full-text indexing support. Users can give keywords in the search bar to retrieve matching documents sorted by significance. For example, users can search a certain folder in the mailbox database.

Security: In many large organizations, access to information in databases needs fine-grained access control as well as security mechanisms for authentication. Every Notes user has a digital ID and access can be granted at different level, from the server down to individual document fields.

5 RENotes Case Study

RENotes is our reverse engineering application that we built as a case study to gain experiences with our approach to tool-building. It leverages Notes features wherever possible, supplementing them with custom functionality where needed. In this section, we describe RENotes' architecture and implementation, list the supposed benefits to its adoption, and relate our experiences with implementing and using the application.

5.1 Architecture and Implementation

RENotes has been implemented with standard three-layer architecture. Its major components are shown in Figure 3. RENotes represents the structure of the system under examination with a typed, directed, attributed graph, similar to the one used in Rigi (see Section 3). The graph is initially produced using existing source code parsers (e.g., Rigi's `cparse` for C, or the CPPX fact extractor for C++ [3]) and saved in Graph Exchange Language (GXL) format [9]. GXL is a XML-based exchange format popular in the reverse engineering community. Its syntax is defined with an XML Document Type Definition (DTD).

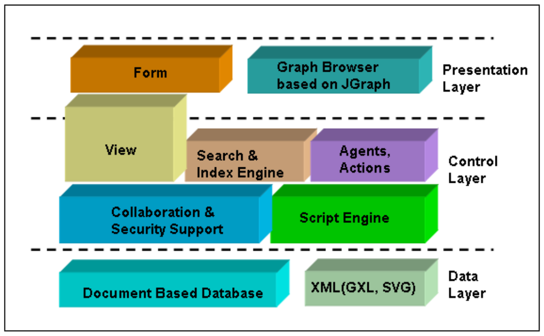


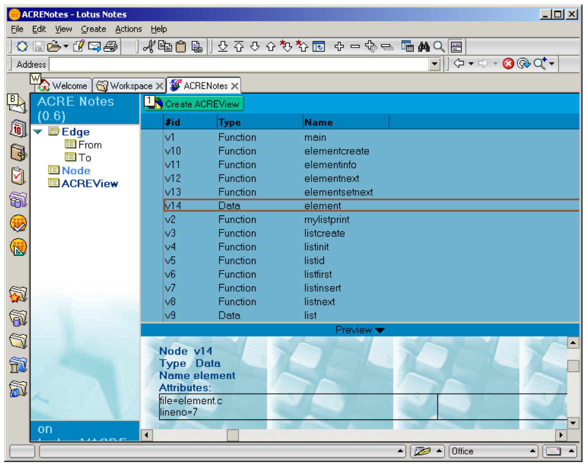
Figure 3: RENotes' layered architecture.

The generated GXL file can be imported into RENotes as a Notes database through a Java agent. In order to perform the transformation, the GXL file is parsed into an XML Document Object Model (DOM) tree. The XML DOM tree is then traversed and elements are converted into the Domino Object Model.

Notes exposes its internal state with the Domino Object Model [25]. This model allows programmatic access to and manipulation of the databases and application services. It has been implemented for a broad range of languages, including the Formula language, Lotus Script, Visual Basic, JavaScript, and Java. Each object defines a set of properties and methods. For example, the `NotesDocument` object represents a document in a database and has the method `AppendItemValue` to add a new field to the document.

The graph to database mapping is simple: each node and arc is mapped to a separate new document with an automatically generated unique identifier. The type and other attributes of each graph element are saved in the corresponding document's fields. Figure 4 shows a database with a small graph of 14 nodes. During the import, the graph is also checked against a source code language-specific domain schema, encoded in XML and held in a separate Notes document. This ensures that the graph is well-formed and meaningful so that other tools can use it safely.

Once the data has been imported, users can manipulate the documents with all the



#id	Type	Name
v1	Function	main
v10	Function	elementcreate
v11	Function	elementinfo
v12	Function	elementnext
v13	Function	elementsetnext
v14	Data	element
v2	Function	mylistprint
v3	Function	listcreate
v4	Function	listinit
v5	Function	listid
v6	Function	listfirst
v7	Function	listinsert
v8	Function	listnext
v9	Data	list

Node v14
Type Data
Name element
Attributes:
file=element.c
lineno=7

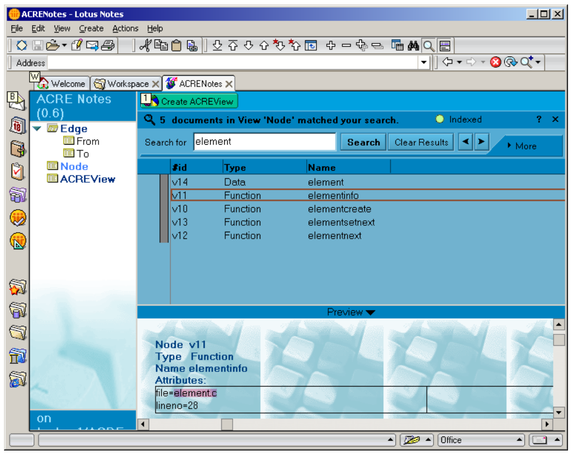
Figure 4: Nodes in a sample RENotes database.

usual Notes tools. They can search for specific nodes or arcs by keyword (cf. Figure 5), or create filtered, sorted views of the (automatically indexed) database based on complex queries. The documents can also be accessed through Notes' standard automation features, allowing users to write ad-hoc scripts to perform more complex operations such as bulk attribute changes or transitive closures on the system's directed relationships. The user can also select from a set of predefined scripts that perform typical reverse engineering tasks such as to find all callers of a function or accessors of a field. These existing scripts provide useful templates as a starting-point for users that want to write their own scripts.

Access to the RENotes databases is controlled by Notes' security features; RENotes defines some common user roles with various degrees of privilege, restricting users' actions with fine granularity. All this functionality is leveraged unchanged from Notes and should be familiar to its users.

5.2 Visualization

The generic textual list views provided by Notes are often not optimal for exploring the structure of a system, so RENotes provides a custom-built graphical visualization. The user



5 documents in View 'Node' matched your search. Indexed

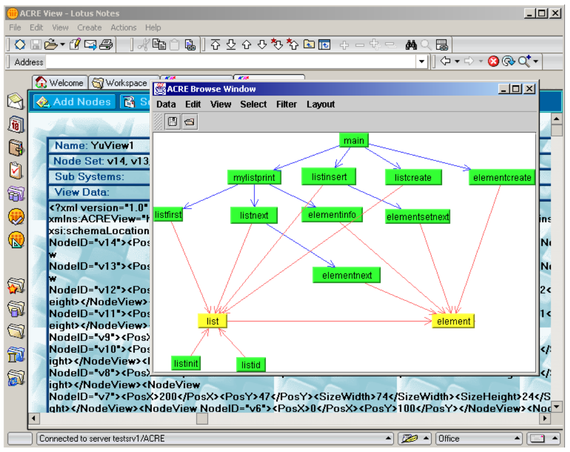
Search for element

#id	Type	Name
v14	Data	element
v11	Function	elementinfo
v10	Function	elementcreate
v13	Function	elementsetnext
v12	Function	elementnext

Node v11
Type Function
Name elementinfo
Attributes:
file=element.c
lineno=28

Figure 5: Keyword search of a RENotes database.

first selects a subset[1] of nodes to visualize. References to the nodes are gathered into a perspective document from which the user can pop up the RENotes graph browser (cf. Figure 6).



Name: YuView1
Node Set: v14, v13
Sub Systems:
View Data:

```
<?xml version="1.0"
xmlns:ACREView="
xsi:schemaLocation
NodeID="v14"><Pos
w
NodeID="v13"><Pos
w
NodeID="v12"><Pos
eight></NodeView>
NodeID="v11"><Pos
eight></NodeView>
NodeID="v9"><Pos>
NodeID="v10"><Pos
ight></NodeView><
NodeID="v8"><Pos>
ight></NodeView><NodeView
NodeID="v7"><PosX>200</PosX><PosY>47</PosY><SizeWidth>74</SizeWidth><SizeHeight>24</S
ght></NodeView><NodeView NodeID="v6"><PosX>0</PosX><PosY>100</PosY></NodeView><Noc
```

Connected to server testsrv1/ACRE

Figure 6: Visualization of a RENotes database.

The browser—written in Java using the open-source, Swing-based JGraph [1] graph editing toolkit and embedded in the RENotes database—provides a visual representation of the nodes and all relationships between them. All graphical elements are connected to the underlying Notes documents, using the domain schema to map their properties to visual attributes. The browser offers basic navigation, manipulation (most notably filtering of nodes and arcs), and layout controls to help the user investigate the system's structure.

When the user exits the graph browser, its state is saved in the corresponding perspective document, encoded as XML. This allows the developer to resume exploration of the graph in a subsequent session. Each perspective document thus represents a separate persistent view on the system graph; the same node may be independently present in many perspectives. The graph visualization can also be saved in Scalable Vector Graphics (SVG) format [5], so that it can be embedded into other documents or shared with people who are not using RENotes or do not have the access privileges necessary to open a perspective document.

6 Related Work

There are other office tools besides Notes that are promising candidates for host applications. In related projects in our group we extend Microsoft Visio and PowerPoint to visualize and manipulate Rigi graphs [16]. Other research, discussed in the following, has leveraged office tools as well to build software engineering functionality.

Desert is an open tool environment consisting of several loosely coupled components [19]. One of these components is a specialized editor for source code and architecture documentation. This editor is based on Adobe FrameMaker and uses a wide variety of FrameMaker's functionality, such as syntax highlighting with fonts and colors, graphic insets, and hypertext links. FrameMaker is extended via the Frame Developer's Kit API.

Riva and Yang have developed a software documentation process that uses Rigi to visualize software artifacts [20]. They used Rigi's scripting capabilities to export this information to Visio as a UML model. The authors take also advantage of Visio's ability to export Visio UML drawings as HTML to Web-enable the documentation.

The Visual Design Editor (VDE) is a domain-specific graph editor implemented with

[1]The subset may, in fact, be the whole graph.

VisualBasic on top of PowerPoint [7]. VDE personalizes PowerPoint with new pull-down menus and icons. The authors state: "PowerPoint offers a highly functional GUI for interactively designing presentation graphics. Virtually every part of that GUI is useful, *without modification*, as part of our design editor."

Tilley and Huang report on their experiences with an industrial client in implementing a software visualization and documentation system in Visio [22]. Visio was selected after evaluating the visualization capabilities of several candidate tools. The authors were constrained in their technology choices by the client's policies. For example, "the company reasonably requested that professional support be available for whichever tools were selected. This requirement immediately ruled out almost all academic and research tools." Among the identified benefits of Visio was that the client already employed Visio in their development process and had a set of custom-developed stencils to represent their software artifacts.

7 Conclusions

Even at this early stage, the RENotes project has generated some interesting insights and potential benefits to adoption, though more work is necessary to evaluate the effectiveness of our approach.

7.1 Development Experience

Building our application in Lotus Notes rather than stand-alone has greatly reduced our development effort. The benefits of reusing Notes functionality more than offset the overhead of adapting to a new development environment, reducing lines of code by an (estimated) order of magnitude. This also allowed us to build several prototypes to verify and test the feasibility of our ideas.

The JGraph framework proved also very helpful: the graph browser application weighs in at less than 4000 lines of code. By comparison, Rigi has about 30000 lines of C/C++ code, though it provides a richer feature set than RENotes does. Overall, the majority of the effort was directed at leveraging the potential the Notes environment provides, as opposed to writing new code from scratch.

As with any framework, there are also some limitations. Notes does not let Java clients control its user interface, preventing the graph browser from tightly binding visualized node selection to a Notes document view. While other APIs (for example, the Notes C API) may afford more control, we did not consider these languages suitable for rapid development of reliable software. There are also potential issues with scaling RENotes to handle larger systems. While we did not try large-scale experiments, we are cautiously optimistic on this count since Notes' database kernel has been refined and optimized over the last decade.

7.2 Benefits to Adoption

RENotes made substantial strides towards improved adoption by building on top of Notes. Notes' relatively large user base (as compared to research prototypes) means that support is plentiful and basic training easy to obtain. Since RENotes reuses many of Notes' features, the learning curve is significantly lowered for existing users, and even new users benefit from Notes' support network. Notes also has a mature, coherent user interface and many convenient tools, such as search and filters. Together with Notes' popularity, this should make RENotes more appealing to domain experts with no reverse engineering experience.

Notes support for ad-hoc end-user programmability is also critical to reverse engineering efforts. Many reverse engineering tasks are tedious to perform manually and, if so accomplished, cannot be recorded and reused. Notes is fully scriptable and even offers users a choice of programming languages that cover the spectrum of formality, letting them pick a tool appropriate for the task at hand. The abundance of scripting options also makes it more likely that a user already knows at least one of them, further lowering the barrier to adoption, and comparing favorably with research prototypes that normally support at most one scripting method.

Building on Notes also lets us leverage its support for collaboration, which should become more important as reverse engineering projects grow more complex. RENotes databases also integrate well into the Notes workspace and can be linked to other databases employed by the user. By providing GXL import and SVG export capabilities, RENotes also integrates well with applications outside Lotus Notes, making it easier to fit into an existing workflow.

7.3 Future Work

In future work, we will add more of the Rigi functionality and reverse engineering capabilities to RENotes. Furthermore, we would like to conduct a user study with software engineers at the IBM Toronto Lab to gain a better understand if, and how, RENotes lowers the adoption barrier. Furthermore, we want to observe how effectively Notes' baseline environment is utilized by RENotes users and how users will integrate RENotes into their work environment and processes.

The work on RENotes described in this paper is part of ACRE V1.0 [16], the first version of the software evolution environment under development at the University of Victoria as part of our Adoption-Centric Reverse Engineering (ACRE) project [15]. In addition to RENotes it consists of several other software visualization engines on top of various office products (e.g., Microsoft Excel, PowerPoint, and Visio). We plan to compare and evaluate our experiences with these approaches and use the implementations for user studies targeted to the question of how to further (industrial) adoption of research tools.

Acknowledgments

This work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Consortium for Software Engineering (CSER), and the Center for Advanced Studies (CAS), IBM Canada Ltd.

About the Authors

Jun Ma is a Master student in Computer Science at the University of Victoria, Canada. He received a B.E. from the Harbin Institute of Technology in China. His research interests include software engineering, XML technology, and CSCW.

Holger M. Kienle is a Ph.D. student in Computer Science at the University of Victoria, Canada. He received a Master of Science degree in Computer Science from University of Massachusetts Dartmouth and a Diploma in Informatics from University of Stuttgart, Germany. His interests include software reverse engineering, exchange formats for re-engineering, program analyses, and domain-specific languages.

Piotr Kaminski is a Ph.D. student in Computer Science at the University of Victoria, Canada. He received a Master of Science degree in Computer Science from the University of Victoria, and a Bachelor of Mathematics from the University of Waterloo, Canada. His interests include software engineering, aspect-oriented programming, the semantic web and human-computer interaction.

Anke Weber is a principal of ExperEdge Technology Partners, an IT consulting company based in Victoria, Canada. Prior to co-founding ExperEdge, she has gained a variety of experiences in the IT field in positions as a co-ordinator for a European Union-funded project at the Paderborn Center for Parallel Computing, as a technical editor and writer at dSPACE Inc., an international supplier of tools for developing and testing new mechatronic control systems, and most recently as a Software Engineering Research Associate in the Department of Computer Science at the University of Victoria, Canada. She received her Master's Degree in Computer Science from the University of Dortmund in Germany. Among here many roles, she appreciates most to work as a designer, editor, and writer in both the technical and non-technical worlds. Her current research interests include adoption-centric software engineering, web site evolution, and "live" systems documentation.

Dr. Marin Litoiu is member of the Centre for Advanced Studies at the IBM Toronto Laboratory where he initiates and manages joint research projects between IBM and Universities across the globe in the area of Application Development Tools. Prior to joining IBM (1997), he was a faculty member with the Department of Computers and Control Systems at the University Politechnica of Bucharest and held research visiting positions with Polytechnic of Turin, Italy (1994 and 1995), and Polytechnic University of Catalunia (Spain), and the European Center for Parallelism (1995). Dr. Litoiu's other research interests include distributed objects; high performance software design; performance modeling, performance evaluation and capacity planning for distributed and real time systems.

References

[1] Gaudenz Alder. JGraph home page. `http://jgraph.sourceforge.net/`.

[2] David Carney. Assembling large systems from COTS components: Opportunities, cautions, and complexities. In *SEI Monographs on the Use of Commercial Software in Government Systems*. Software Engineering Institute, Carnegie Mellon University, June 1997.

[3] Thomas R. Dean, Andrew J. Malton, and Ric Holt. Union schemas as a basis for a C++ extractor. *Eighth Working Conference on Reverse Engineering (WCRE '01)*, pages 59–67, October 2001.

[4] Mike Falkner. *Using Lotus Notes as an Intranet*. Wiley, 1997.

[5] Jon Ferraiolo. *Scalable Vector Graphics (SVG) 1.0 Specification*. W3C, September 2001. `http://www.w3.org/TR/2001/REC-SVG-20010904/`.

[6] Michelle Gantt and Bonnie A. Nardi. Gardeners and gurus: Patterns of cooperation among CAD users. *Conference on Human Factors in Computing Systems (CHI 92)*, pages 107–117, May 1992.

[7] Neil M. Goldman and Robert M. Balzer. The ISI visual design editor generator. *IEEE Symposium on Visual Languages (VL '99)*, pages 20–27, September 1999.

[8] Williams Gregg. Hypercard: Hypercard extends the machintosh user interface and makes everybody a programmer. *Byte*, pages 109–117, December 1987.

[9] Richard C. Holt, Andreas Winter, and Andy Schürr. GXL: Towards a standard exchange format. *Seventh Working Conference on Reverse Engineering (WCRE '00)*, pages 162–171, November 2000.

[10] IBM. Lotus home page. `http://www.lotus.com`.

[11] Helena Karsten. Collaboration and collaborative information technologies: A review of the evidence. *The DATA BASE for Advances in Information Systems*, 30(2):44–65, Spring 1999.

[12] Tung Lai Lai and Efraim Turban. One organization's use of Lotus Notes. *CACM*, 40(10):19–21, October 1997.

[13] Wendy E. Mackay. Triggers and barriers to customizing software. *Conference on Human Factors in Computing Systems (CHI 91)*, pages 153–160, April 1991.

[14] Allan MacLean, Kathleen Carter, Lennard Lövstrand, and Thoams Moran. User-tailorable systems: Pressing the issues with buttons. *Conference on Human Factors in Computing Systems (CHI 90)*, pages 175–182, April 1990.

[15] Hausi A. Müller, Margaret-Anne Storey, and Ken Wong. Leveraging cognitive support and modern platforms for adoption-centric reverse engineering (ACRE). *CSER Research Proposal*, November 2001.

[16] Hausi A. Müller, Anke Weber, and Ken Wong. Leveraging cognitive support and modern platforms for adoption-centric reverse engineering (ACRE). *3rd International Workshop on Adoption-Centric Software Engineering (ACSE 2003)*, pages 30–35, May 2003.

[17] Bjorn Erik Munkvold and Robert Anson. Organizational adoption and diffusion of electronic meeting systems: A case study. *ACM 2001 International Conference on Supporting Group Work (GROUP '01)*, pages 279–287, September 2001.

[18] Stanley R. Page, Todd J. Johnsgard, Uhl Albert, and C. Dennis Allen. User customization of a word processor. *Conference on Human Factors in Computing Systems (CHI 96)*, pages 340–346, April 1996.

[19] Steven P. Reiss. The Desert environment. *ACM Transactions on Software Engineering and Methology*, 8(4):297–342, October 1999.

[20] Claudio Riva and Yaojin Yang. Generation of architectural documentation using XML. *9th Working Conference on Reverse Engineering (WCRE 2002)*, pages 161–169, October 2002.

[21] Diomidis Spinellis. Unix tools as visual programming components in a GUI-builder environment. *Software—Practice and Experience*, 32(1):57–71, January 2002.

[22] Scott Tilley and Shihong Huang. On selecting software visualization tools for program understanding in an industrial context. *10th International Workshop on Program Comprehension (IWPC 2002)*, pages 285–288, June 2002.

[23] Scott R. Tilley. Domain-retargetable reverse engineering II: Personalized user interfaces. *1994 International Conference on Software Maintenance (ICSM '94)*, pages 336–342, September 1994.

[24] Scott R. Tilley, Hausi A. Müller, Micheael J. Whitney, and Kenny Wong. Domain-retargetable reverse engineering. *Conference on Software Maintenance (CSM '93)*, pages 142–151, September 1993.

[25] Tommi Tulisalo, Rune Carlsen, Andre Guirard, Pekka Hartikainen, Grant McCarthy, and Gustavo Pecly. *Domino Designer 6: A Developer's Handbook*. IBM Redbooks, December 2002.

[26] Andrew Walenstein. Improving adoptability by preserving, leveraging, and adding cognitive support to existing tools and environments. *3rd International Workshop on Adoption-Centric Software Engineering (ACSE 2003)*, pages 36–41, May 2003.

[27] Kenny Wong, Scott R. Tilley, Hausi A. Müller, and Margaret-Anne D. Storey. Structural redocumentation: A case study. *IEEE Software*, 12(1):46–54, January 1995.