Browsing Software Architectures With LSEdit

Nikita Synytskyy, Richard C. Holt, Ian Davis
University of Waterloo
Canada

Abstract—**Fact bases produced by software comprehension tools are large and complex, reaching several gigabytes in size for large software systems. To effectively study these databases, ether a query engine or a visualization engine is necessary.**

In our proposed demo we will showcase LSEdit, a full-featured graph visualizer and editor, which is suitable for, but not limited to, visualizing architectural diagrams of software. LSEdit is equipped with advanced searching, elision, layout and editing capabilities. It has been successfully used in the past to visualize extractions of Mozilla, Linux, Vim, Gnumeric, Apache and other large applications.

Index Terms—**Visualization, software architecture.**

I. Introduction

LSEdit is a graph visualizer that is uniquely suited for, although not limited to, displaying architectural diagrams of software systems. LSEdit displays the architecture of a software system by treating it as a set of nested boxes (entities) interconnected by arrows (edges). The nesting of boxes usually, although not always, represents containment, such as containment of a file within a subsystem or of a function within a file. Various types of edges can be used to represent various relationships that exist between entities in a software system, such as "*calls*" or "*accesses*".

By representing system architecture as a series of nested boxes, LSEdit allows its users to focus their exploration on a certain subsystem of the software being examined, thereby reducing the amount of information that needs to be understood at any one time. Elision and layout tools provide the ability to filter available information and present it in a variety of layouts, allowing the user to generate a view of the software that is most meaningful.

LSEdit supports both exploration and editing of the landscape. On the exploration side, it provides a containment tree view of the landscape, not unlike that given by Windows Explorer, a browsing history that the users can view and travel back and forward in, searching by name and type, and elision tools that help users in hiding the data they don't currently want to see. Editing tools include the ability to add, delete and edit nodes and edges and apply configurable layout algorithms. Multiple levels of undo mean that users can edit without fear, knowing that all changes they make can be rolled back.

In our demo we will show a variety of fact bases (also called *landscapes*) that have been extracted for LSEdit in the past. We will include both large and small fact bases. Large fact bases, which represent big software systems like Mozilla or Gnumeric, will allow the users to experience searching and elision capabilities of LSEdit. Smaller landscapes, which represent toy programs, will give the users an idea of how a real application looks in landscape format. The users will be given a hands-on experience; they will have a chance to use LSEdit to explore a large landscape and experience its features for themselves.

The following sections of the proposal are as follows: Section 2 gives a more detailed description of LSEdit's features, and gives some insight into its implementation details. Section 3 describes our demonstration plan. Section 4 describes the benefits of using LSEdit for architecture recovery and gives a brief history of its use in program comprehension tools. Finally, section 5 discusses future development plans for LSEdit.

II. Description of LSEdit

LSEdit is written in Java and runs on any platform that provides a Java Virtual Machine. It can also be embedded in a web page and run across the Internet as an applet. When run as an applet, it can load and display a landscape file located anywhere on the Internet.

The landscapes that LSEdit operates on are stored in Tuple-Attriubte Language (TA) format [1]. TA is a plain-text database format, which means it is easy to generate either automatically or by hand. TA is based on Rigi Standard Format (RSF), first introduced by the Rigi [2] reverse engineering tool. TA is the underlying format of several reverse engineering tools [3][4].

LSEdit's features fall into two main categories: exploration features and editing features. Exploration features help users study the landscape, allowing them to search and navigate

Manuscript received February 9, 2005.

Richard C. Holt is a full professor and a Nortel Networks Chair for Telecommunications Software Engineering at the University of Waterloo, Ontario, Canada. E-mail: holt@uwaterloo.ca

Nikita Synytskyy is a researcher working for the Software Architecture Group at the University of Waterloo, Ontario, Canada. E-mail: nikita@mondenet.com

Ian Davis is a researcher working for the Software Architecture Group at the University of Waterloo, Ontario, Canada. E-mail: ijdavis@uwaterloo.ca.

through the fact base, selectively hide and display data, and manage their navigational experienced. Editing features allow the users to change the landscape by adding, deleting and editing nodes and edges, and automatically or semi-automatically arranging nodes in a landscape into layouts that are helpful in understanding the software being examined.

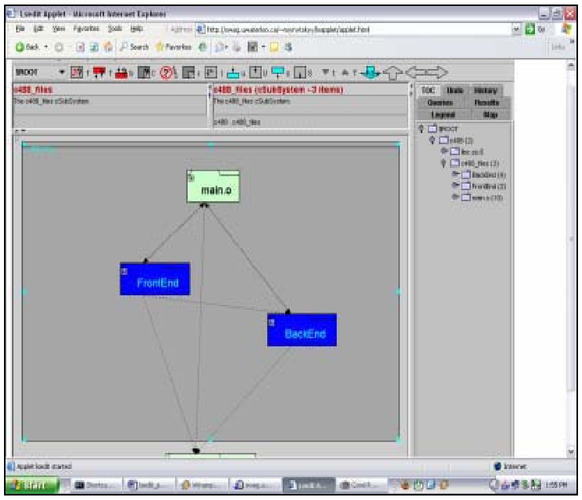

Figure 1: LSEdit running as an applet.

A. *Exploration features*

Navigation. A user uses LSEdit to browse the landscape by entering into the boxes corresponding to architectural entities of the program being explored. When in a particular box, only its children are shown by default; all other boxes are hidden to prevent cluttering the view. The user can navigate deeper and deeper into the landscape, navigating it as if it were a filesystem.

Dependencies between entities are shown as arrows going between the appropriate boxes; these dependencies can also be lifted to provide a higher level of abstraction.

Navigation aides. LSEdit provides several navigational tools that make navigation easier. One is a Table of Contents (TOC) view, which shows the entire landscape in an expandable/collapsible tree. Another is the history; similar to the history feature of Web browsers, it keeps track of all locations visited and allows the user to return to any one of them. The final navigational tool is the map, which always shows the path taken from the root node to the current location.

Searching and elision. Users can search for any box by its name and type. They can also limit the amount of information shown by hiding all boxes or arrows of a given type, or related to a particular entity.

B. *Editing features*

Basic editing. LSEdit allows for creation of new entities and edges, and deletion of existing ones. Location and all other properties of entities can be changed.

Layouts and positioning. LSEdit provides several built-in layout algorithms, which organize the entities in a landscape according to the edges going between them in various ways. Spring and tree layouts are among the ones provided. Entities can also be aligned on both vertical and horizontal axes, distributed in a given amount of space, and resized in a number of ways.

Undo and redo. Any edit that can be done can be also undone. Any edit that has been undone can also be re-done. 100 levels of undo are supported by default, and the limit can be increased if necessary.

III. Demonstration format

During the demo we will have LSEdit running on computers with several large landscapes available for browsing. This will give the participants a chance to use LSEdit to explore a real system. This will expose them to the exploration features that LSEdit provides. To demonstrate LSEdit's ability to run as an applet over the Web, one of the running copies of LSEdit will be running as an applet on an Internet browser. Figure 1 shows a screenshot of LSEdit running as an applet under Internet Explorer.

Several small landscapes representing toy programs will also be available. These examples are valuable because toy examples can be easily understood from their source code alone. The demonstration participants can gain insight into how LSEdit displays software architecture by comparing the landscapes presented with the mental models that they have independently formed.

IV. LSEdit in Program Comprehension

A. *Understanding architecture*

LSEdit simplifies studying software architecture by providing a repository to store information about the architecture of a system as it is discovered. It allows for convenient storage, retrieval and modification of information on containment, dependencies and module structure. LSEdit presents this information using an easily understandable paradigm and provides tools to search, filter and modify it.

Understanding the architecture of a software system without the benefit of specialized tools usually involves two activities: studying the source code to gain detailed knowledge about dependencies between various code entities, and examining the directory structure of the project and/or supporting documentation in order to understand how the software system is divided into sub-parts, i.e. modules. The two bodies of knowledge can then be combined to get a complete picture of the system's components and their interactions.

Unfortunately for programmers and researchers, this activity is rather difficult, because it requires the person doing the study to sift through, and commit to memory, very large amounts of data. This process is inevitably error-prone—in any system of more than a trivial size the researcher will not be able to discover all dependencies, will get some of them wrong, and will forget some of the results.

Keeping notes in this situation is essential, and LSEdit's main strength is that it serves as a notebook. Rather than starting out blank, it comes pre-initialized with all the facts discovered by a fact extractor. These typically capture all dependencies within the software system, and provide a general outline of the module structure, as derived from the directory structure. LSEdit's exploration features provide for easy searching and viewing of that information.

The source code does not contain all information on the software's structure. Some of it is contained in documentation, in comments, and in the minds of the developers. As the researcher discovers this information, it can be incorporated into the landscape, until the landscape adequately reflects the system architecture.

B. Historical use

LSEdit has a long development history and over the years has been used as a visualization engine for many reverse engineering tools developed at the Software Architecture Group (SWAG) at the University of Waterloo. It was first developed for use with the Portable Bookshelf [5]. As a part of that system, it was used to visualize the extracted architecture of the Linux kernel [6][7].

An upgraded version of LSEdit is currently used as a part of several architecture recovery and design analysis toolkits: the LDX/BFX pipeline [3][8] and the SWAG Kit [4][9]. As part of these packages, it is used in graduate software architecture courses taught at the University of Waterloo.

LSEdit is free software. It is available for download, together with supporting documentation, on the LSEdit homepage[10].

V. Future Work

When exploring a large fact base, it is beneficial to have both a query engine and a visualizer. The Software Architecture Group of the University of Waterloo has experience developing both classes of tools, and our next project aims to integrate the two in a single application.

A query engine integrated with a visualizer would yield the combined power of both classes of tools: the expressiveness and power of command-line queries will be complemented by the instant feedback and ease of understanding that only a graphical interface can provide.

Such an application will also be able to capitalize on the synergies of the two engines. For example, it could provide additional feedback by visually displaying the results of executed queries, or by allowing users to build queries about the objects currently visible in the graphical display.

References

[1] "An Introduction to TA: the Tuple-Attribute Language", R. C. Holt, March 1997 (updated July 2002). Available on-line: http://plg.uwaterloo.ca/~holt/papers/ta-intro.htm

[2] H. A. Müller, K. Wong, and S. R. Tilley. "Understanding software systems using reverse engineering technology." The 62nd Congress of L'Association Canadienne Francaise pour l'Avancement des Sciences Proceedings (ACFAS 1994)

[3] "Linker-Based Program Extraction and Its Uses in Studying Software Evolution", Jingwei Wu and Richard C. Holt, in Proceedings of the International Workshop on Unanticipated Software Evolution, Barcelona, Spain, March 28, 2004.

[4] "Union Schemas as the Basis for a C++ Extractor", Thomas Dean, Andrew Malton, and Richard Holt: Working Conference on Reverse Engineering, Stuttgart, Germany, Oct 2-5, 2001.

[5] "The Software Bookshelf", P. Finnigan, R. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. Muumller, J. Mylopoulos, S. Perelgut, M. Stanley, and K. Wong. IBM Systems Journal, Vol. 36, No. 4, pp. 564-593, November 1997

[6] "Linux as a Case Study: Its Extracted Software Architecture", Ivan T. Bowman, Richard C. Holt and Neil V. Brewster, ICSE '99: International Conference on Software Engineering, Los Angeles, May 1999.

[7] "Software Bookshelf of the Linux Kernel", Available on-line: http://swag.uwaterloo.ca/pbs/examples/linux/index.html

[8] LDX and BFX homepage. Available on-line: http://swag.uwaterloo.ca/qldx/index.html

[9] SWAG Kit homepage: http://swag.uwaterloo.ca/swagkit/index.html

[10] LSEdit homepage: http://swag.uwaterloo.ca/lsedit/index.html