

VIFOR 2: A Tool for Browsing and Documentation

Vaclav Rajlich, Sridhar Reddy Adnapally

Department of Computer Science
Wayne State University
Detroit, MI 48202, USA
rajlich@cs.wayne.edu

Abstract.

During the maintenance of legacy systems, the structure and the documentation of the system usually deteriorates, and hence the maintenance becomes progressively harder and harder. In order to reverse this deterioration, it is essential to record the understanding of the system continually, before it is forgotten. It is also important to structure it in such a way that it can be easily accumulated and retrieved. In this paper we are presenting a tool called VIFOR 2, which is oriented towards this situation. The tool combines two technologies: browsing and hypertext documentation. The browser supports rapid navigation around the code. The layered hypertext supports the incremental recording and retrieval of the documentation on the desired level of abstraction (domain, algorithm, representation, etc.) We briefly discuss our own experience with the tool.

1. Introduction

Legacy systems [2] are the systems which have one or several of the following attributes:

- they were implemented many years ago
- their technology became obsolete
- they have been maintained for a long time
- their structure deteriorated
- their documentation is obsolete or misleading
- they became indispensable to the users
- they represent a large investment
- they may contain business rules not available elsewhere
- they cannot be easily replaced
- the original authors are not available.

The maintenance of legacy systems is difficult. It is typical that during the maintenance process, the structure and the documentation of the system deteriorates, and hence the maintenance becomes progressively harder. This is even more serious in a situation where there is a turnover of maintenance programmers, or where the maintenance team is understaffed. In both situations,

maintainers often visit code which is completely unfamiliar to them, or which they visited a long time ago and forgot its functionality. In this particular case, it is essential to record the understanding of the system before it is forgotten, and to structure it in such a way that it can be easily accumulated and retrieved. In this paper we are presenting a tool called VIFOR 2, which is oriented towards this situation. The tool combines two technologies: browsing and hypertext documentation.

Browsers are comprehension tools which extract information from programs and store it in a database. The information is cross-referenced with the code. In order to navigate through the program, the user queries the database and follows the relationships among the entities of the program. Most often, these are the relationships between the definitions and the use of variables, procedures, types, etc. Example of a question answered by a browser is: "Find all functions which call function F ()." Or, "Find all functions which use a global variable G." The definitions and the use may be spread over distant parts of the software and over different files, and the browsers allow for quick navigation. This substantially accelerates "flipping" through the code, and lessens the amount of effort necessary for software comprehension. There has been substantial literature dealing with browsers. Examples of browsers are [4,7].

Hypertext is a powerful tool for software documentation [5,8,9,10,11], and has been used for annotations of code. The annotations are explanations of various constructs of the code. The technique of hypertext annotations was further improved in [6], where the annotations were structured into layers and linked to the top-down theories of software comprehension. In this case, each layer describes a program construct at a certain level of abstraction. The exact number of layers is chosen by the programmer or team manager, and depends on the project and its domain. The typical layers for any program are domain layer, algorithm layer, interface layer, and representation layer. Domain layer contains descriptions in the terminology of the problem domain, which can easily be understood by the end user without

any knowledge of the programming details. Algorithm layer describes the algorithms used in the implementation. Interface layer describes the argument list and side effects of individual functions of the program. Representation layer explains representation of variables and the meaning of various values.

The programmer or project manager may select additional layers whenever necessary. Examples are the layers describing the maintenance history, experience of individual team members with the maintenance of the software, comments on the quality of the code, warnings about known problems, etc. The layering of annotations

allows maintenance programmers to create these layers, and to incrementally add additional information or insight as they are gained during the maintenance. Hence the purpose is to stop or completely reverse the deterioration of the documentation during the maintenance. The accumulated information can be accessed quickly without sifting through the volume of irrelevant detail. The maintainers who are unfamiliar with the system look for a high level view of the system before searching through the details. Maintainers more familiar with the system can proceed to the details directly, without visiting the high levels first. Every maintenance programmer visits

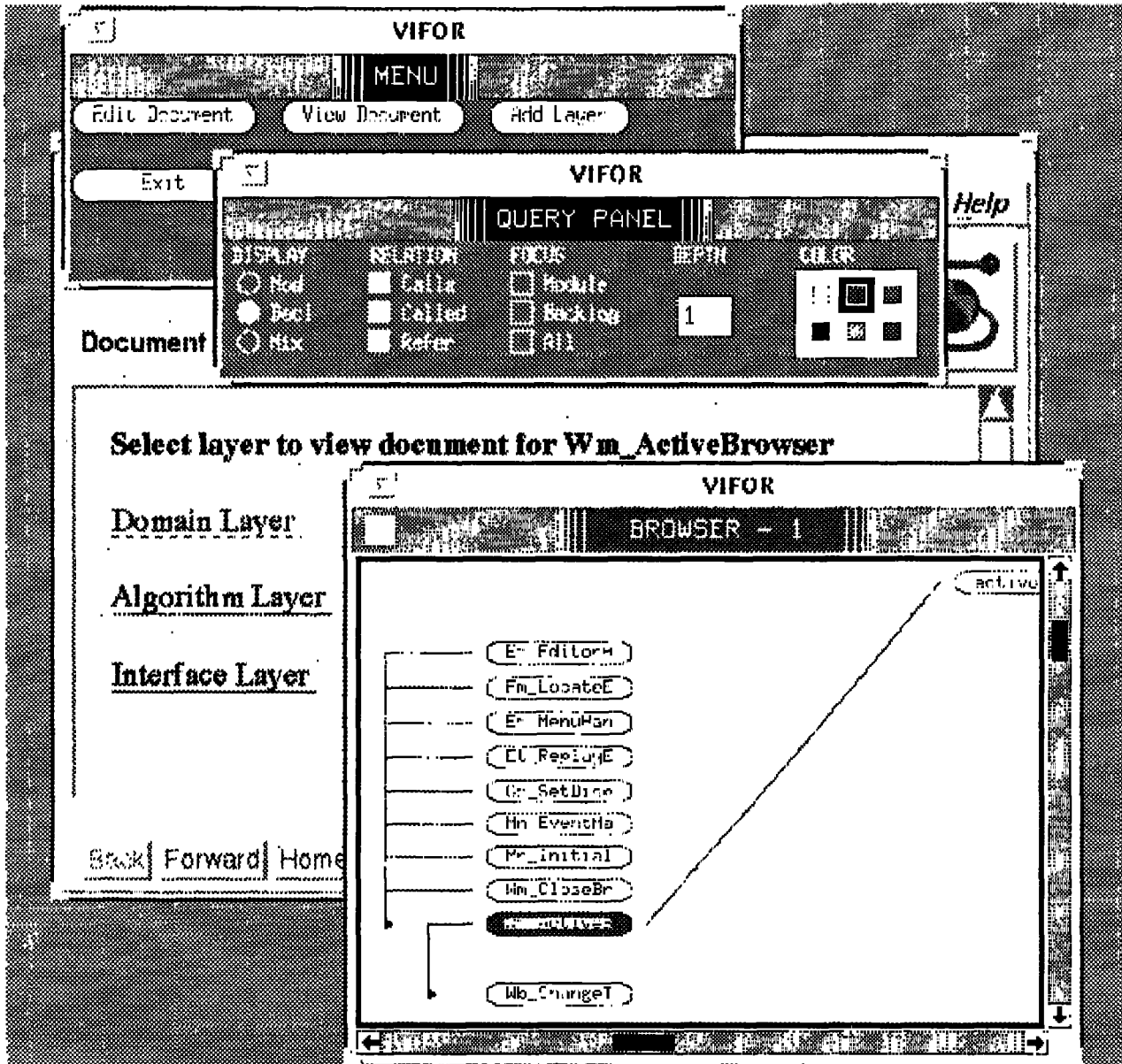


Figure 1.

only the layers and the constructs which are related to his information needs.

In this paper, we describe a tool called VIFOR 2, which combines both browsing and layered hypertext annotations into one tool. Section 3 contains summary of our experience and conclusions.

2. Extending VIFOR with Hypertext

Original VIFOR 1 [4] is a browser supporting the development and maintenance of C and FORTRAN programs. It interacts with the user through a display of

programs in two forms: the traditional code, or the graphic representation. It contains transformation tools for both directions i.e., from code to graphics and from graphics to skeletons of code. VIFOR manipulates program entities like functions, modules and data. A parser extracts from the code all the information about these entities and their relationships. This information is stored in a database which can be later queried. The results of queries are displayed graphically in specially designed windows. Relations in the VIFOR 1 data model are of three kinds: Call relation, Reference relation, and Belongs to relation.

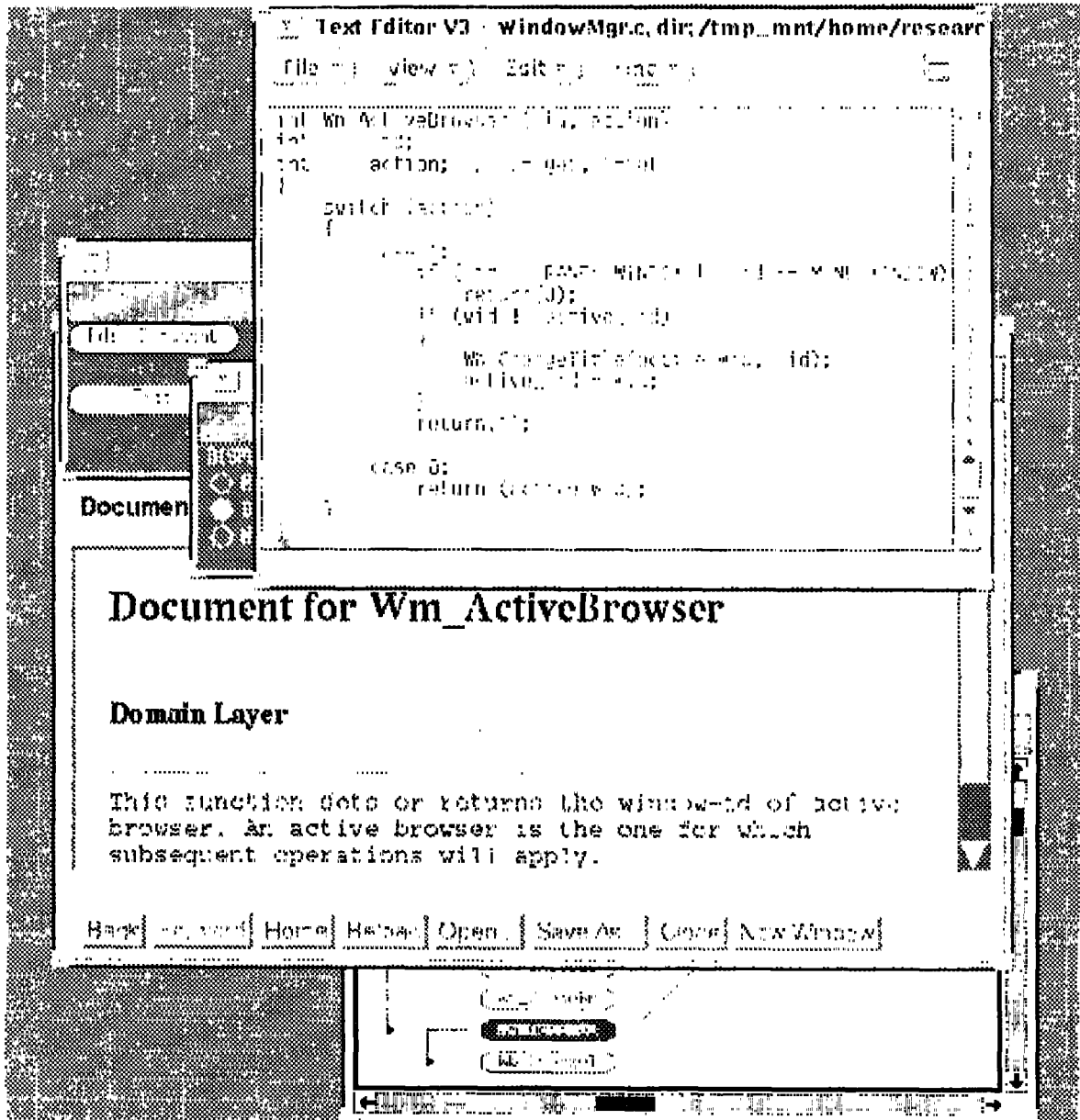


Figure 2.

The user enters his requests through the Menu window. The Query panel window allows the construction of database queries. For more detail, see [4].

We created VIFOR 2 by extending VIFOR 1 with hypertext documentation. For the hypertext documentation, we choose the html language because of its wide availability and familiarity. We chose mosaic as the hypertext browser. This means that VIFOR 2 directly operates mosaic, by specifying for it the documents to view, directing it to open new windows, etc. The VIFOR 2 interface consists of the old interface of VIFOR 1, augmented by a new menu panel called "Document menu". The document menu has three items: "Edit Document", "View Document" and "Add Layer", see Figure 1. Also in Figure 1, the Query panel displays the specifications of a query, and the results of that query are in the window "Browser - 1". "Browser - 1" contains highlighted function "Wm_ActiveBrowser", all functions which call it, all functions which are called by it, and external data referenced by it. In the background there is a mosaic window which shows all the layers of documentation of Wm_ActiveBrowser.

Figure 2 contains a screen which originated from the previous one, after two actions: the Domain layer of the documentation was selected, and the editor was invoked for the function "Wm_ActiveBrowser". The top window contains this editor and displays the code, and the mosaic window now displays the contents of the domain annotation for the function.

VIFOR 2 stores the documentation of all entities in separate files in a separate directory, in html format. In this directory, VIFOR 2 maintains the names of all layers that were created for that project. For example, if a function named "read_input" is documented in three layers then the documentation of these three layers is stored in three different files. There is also a special file with a menu interface for all documentation files.

Example of the documentation for the function "main" is in Appendix.

3. Our Experience

In our experience, layered annotations combined with a browser substantially improve the productivity of maintenance, particularly in a situation of rapid turnover of the maintenance team. The new people on the team can start maintenance quickly, because they have the documentation available in a form which is easy to access and logically structured. The documentation is accumulated incrementally over time. As the maintainers acquire understanding of certain parts of the system, they record that understanding incrementally in hypertext.

In our project, we have been maintaining and evolving the family of VIFOR browsers. Our project is characterized by a rapid turnover of students. Typically, a student works on the project for one year or less. When an experienced student is departing from the project, there

is seldom an opportunity for him to train newly coming students to assume his responsibilities. In this situation, the browser and annotations supported by VIFOR 2 allowed us to preserve the knowledge of the code, and communicate it to the newcomers on the project. Every maintenance action is finished with the documentation phase, where the maintainer records his understanding of the code in appropriate layers for appropriate constructs. In this way we were able to accumulate documentation for the most critical parts of the software, and improve it over time. VIFOR 2 is for us an enabling technology allowing us to maintain and evolve the code. Without it, the knowledge of the code would be lost in the turnover, and the software could not be maintained.

Based on our experience, we expect that this technology will find useful application in other situations with similar needs.

References

- [1]. R. Brooks, "Towards a theory of the comprehension of computer programs." *International Journal of Man-Machine Studies* 18, No.6 (1983) 543-554.
- [2]. P. B. Carroll, "Computer glitch: Patching up software occupies programmers and disables systems." *Wall Street Journal*(January 22, 1988), p.1.
- [3]. T. A. Corbi, *Program Understanding: Challenge for the 1990s*, IBM Systems Journal, Vol 28.
- [4] V. Rajlich, N. Damaskinos, W. Khorshid, P. Linos, *VIFOR: A Tool for Software Maintenance, Software Practice and Experience*, Jan, 1990, 67-77.
- [5] E.J.Younger and K.H.Bennett, *Model-Based tools to Record Program Understanding*. In: *WPC '93, Second Workshop on Program Comprehension*. pp: 87-95.
- [6]. V. Rajlich, R. Gudla, J. Doran, *Layered Explanations of Software: A Methodology for Program Comprehension*, In: *Third IEEE Workshop on Program Comprehension*, Nov. 14-15, 1994, Washington, D.C.
- [7] U. De Carlini, A. De Lucia, G. A. Di Lucca, G. Tortora, "An Integrated and Interactive Reverse Engineering Environment for Existing Software Comprehension." In *WPC '93, Second Workshop on Program Comprehension*. pp. 128-137.
- [8] P. Brown, "Integrated Hypertext and Program Understanding Tools," *IBM Systems Journal*, Vol. 30, No. 3, 1991
- [9] W. Horton, "Is hypertext the best way to document your product?: An Essay for Designers," *Technical Communication*, 38:20-32 (Feb' 1991).
- [10] C. Franklin, "Hypertext Defined and Applied," *Online* 13:37-49 (May 1989).

[11] N. Fletton, M. Munroe, Redocumenting software systems using Hypertext Technologies, Proc. CSM 1988, 54-59

Appendix

In this appendix, we present the annotated function "main" of VIFOR 2.

User's view:

VIFOR parses the source code to extract the important parts of the program like modules, declarations, and relations like call, reference. This information is stored in a database, which can be later queried through buttons of Query panel. The results of the query are displayed in the Browser window.

Call Interface: `main (argc,argv)`

Input parameters:

argc: An integer representing number of command line arguments.

argv: Array of strings containing command line arguments.

Output parameters: None

Return Value: None

Algorithm:

This is the driver for the system.

Parse command line arguments for project name and other valid options (*Fu_argmts* function call).

Read in project configuration file (*FM_Init* function call).

Initialize database. (*DB_Init* function call).

Load the project database (*DB_Load* function call)

Set up browser structures for the host and open initial windows (*Mn_Initialize* function call).

Wait for user events (*Mn_EventManager* function call).

Code:

```
main(argc,argv)
int argc;
char **argv;
{
    int status;
    char host[50];
    struct timeval tp;
    struct timezone tzp;

    if(signal(SIGCHLD,
        Dm_MosaicExited)==SIG_ERR) {
        perror("signal ");
        exit(1);
    }

    Fu_argmts(argc, argv, host);
    FM_Init();
    Dm_Init();
    DB_Init("PROJECT", &status);
    DB_Load(&status);
    Mn_Initialize(host);
    Mn_EventManager();
}
```