WhereTheHeck: a Web based source code navigator

D.Menasce[1], *S.Magni*[1], *F.Prelz*[1], *L.M.Barone*[4,2], *L.Dell'Agnello*[3], *E.Leonardi*[2]

[1] Institute1, INFN-Milano,Italy
[2] Institute2, INFN-Roma,Italy
[3] Institute3, INFN-CNAF,Italy
[4] Institute4, University of Rome "La Sapienza"

Abstract

In this paper we present a novel approach to WEB based source code navigation, in the context of the WINNER R&D project on distributed computing tools. We aim to provide developers and code maintainers, usually located in different countries, a powerful and portable way to browse and search the code of large software projects through the Web. Unlike other tools, based on static hyperlinks and ad-hoc parsers, this tool employs dynamical linking and public domain, wide used compilers for the parsing step. Our package, named WhereTheHeck, allows the user to specify elements of the source code (language tokens such as variables, common blocks, classes, pointers) as regular expressions through a WEB form under Netscape. The regular expression is then used to find matches in the code using a token database updated offline on a regular basis. The retrieved information is presented to the user as a list of hyperlinked instances of the requested token (file names containing the token, line number within the code etc). From the list one can link to the corresponding code formatted as a dynamically generated HTML page (with all relevant hyperlinks). The whole package has been developed in PERL and JavaScript, making it highly portable. We made the choice of using the `f2c` (for FORTRAN) and `gcc` compilers (for C and C++, both on the public domain) as parsing engines, which again ensures a high degree of portability. The tool is currently in alfa test for FORTRAN and C, and the navigation functionality for C++ will be implemented in the near future. WhereTheHeck is made available to the user community as a tar file (downloadable from the WEB).

Keywords: code, hyperization, browser, regexp

1 Introduction

Large software projects in HEP have always posed the problems of documentation, maintainability and learning curve; although no definitive solution have been found yet, the ability to navigate efficiently and quickly the source code has been longly recognized as an essential tool for maintenance and new developments. Unfortunately this ability is not fully provided even by the most sophisticated editing programs, not to mention the problem of remote access to source code by users distributed over the world. Web-based tools have been proposed in the past to handle this problem; using the Web naturally solves the remote access obstacle, provided the source code has been made available to the Web itself. Code browsing however implies a process of transformation of the source code into HTML pages where language entities may become at the same time links and targets for the navigation. This process is non trivial both technically and conceptually: known problems are creation of the entity dictionary, navigation rules, instability of the code which implies in turn instability of the generated Web pages, need for non intrusive tools. In this paper we present a tool named WhereTheHeck (WTH), developed in the context of an INFN R&D project called WINNER[1], which tries to solve most of the problems outlined above, providing powerful navigation tools through FORTRAN and C code, and soon C++.

2 Architecture

By design, we decided to implement powerful navigation tools throughout the source code only; WTH does not allow to navigate through other kinds of information (such as manuals, help pages, OMT diagrams etc.). On the other hand we decided to provide something more than a simple list of language entities for the user's navigation; this decision brought us to implement a search engine based on the use of regular expressions[2], allowing in principle queries of arbitrary complexity on language elements.

We also decided to produce a tool not based on any commercial product but at the same time not constraining the user with very specific packages or hardware platforms. So WTH has been implemented in Perl, with some parts written in Javascript for the Web interface.

The architecture of WTH is based on the following components:

- the language parser
- the tokens database
- the Web interface
- the installation package

In this section we will discuss in detail these components.

2.1 The Language Parser

In order to build a Web out of a software package one needs to parse the source code and to recognize and extract all the language entities which may be relevant to the navigation. The extent of this process may go from a limited set of language entities (subroutines and common variables in FORTRAN, classes in C++) to the whole set of tokens (as the index of a DO-loop or a private variable in a class), as is done by a real compiler. Of course the wider the tokens set, the more difficult the implementation of the parser. It should be noted that some parsers, either independent or bound to commercial products, exist on the market; their cost is generally high, and they are normally available only on some operating systems.

Because writing a parser is not a typical HEP task, and we wanted to use software freely available, highly portable and with large parsing capabilities, we chose to use and modify the lexical analysis component of the GNU[3] compilers. We used `f2c` for the FORTRAN and `gcc` for the C and C++ languages. Modifications were needed to extract the token name, type and location during the compilation phase, writing them out to an external file. These component is written in `lex-yacc` and is accessible in the GNU compiler code, although modifying it is not trivial.

A problem arises with C and C++ macro directives (e.g. `#define`); directives are expanded during precompilation, and the compiler parser provides the location in the file of anything coming from macros **after** the expansion. This will not match usually with the location in the original file, preventing navigation to these entities. The problem has been solved writing an inverse precompiler (`uncpp`) which, taking the location produced by the compiler, of a token from a macro computes back the original location for the same token in the not-expanded code.

2.2 The Tokens Database

The tokens database is the repository of all the language entities; it allows to generate a Web out of the source code turning each token into a hyperlink (possibly according to a configuration file which specifies the Web connectivity, i.e. where does one go clicking the name of a subroutine) and allows to look for a token or a token list by a regular expression. The database contains, for each token, the complete information extracted from the parser (name, type, row and column

number in a file); if a token appears many times it has one entry in the database and the related information is stored as a cascade of linked lists.

The database has been written specifically for this project in C and made a PERL module in the standard way. It consists of many files organized as linked lists to optimize the search speed (it takes less than a second to find one token out of four millions on the average CPU). The problem with the database consists in its synchrony with the corresponding code. If the code is stable for long times the database may be generated once; in the case of code development one must update the database regularly (for instance every night) to keep up with code modifications. In this stage of the project it is not possible to extract the code from a code management repository, like CVS; to generate the database and navigate the code actual source files must be available.

2.3 The Web Interface

The Web interface was designed as an abstraction layer on the token database; the navigation tool is completely generic and the same for all languages (C, C++ and FORTRAN). The Web pages are created on-demand via PERL CGI scripts; the pages contain Javascript instructions for a better user interactivity. This choice limits, for the time being, to Netscape as the only browser for navigation, but it improves both user friendliness and navigation speed. The search engine is activated through input fields and it may search tokens in the database as well as strings or patterns in the code, always using regular expressions. Each instance of subroutine call, function or class reference is an actual hyperlink to all the definitions of such entities in the directory tree spanned by the parser; no other hyperlinks have been made available to the user, since navigation is guided by the user input form shown in Fig. 1, where a snapshot of a WTH session is pictured. Netscape pops up a pair of windows; the one on the left is used as a user input form (where queries are formulated as regular expressions), while output is presented on the multiframe window on the right. We have envisaged two navigation modes; the first one (shown in the example) queries the token database to find matches of the specified regular expression. The second mode allows for generic searches of *any* substring (again expressed as a regular expression); in this second mode the search is performed on the original source code files instead of the token database. The two mode covers any possibile kind of search or browsing a user may wish to perform, thus ensuring a complete navigation throughout the whole source package.

2.4 The Installation Package

WTH may be easily installed from a `tar`-file invoking only two scripts:

1) `INSTALL` to compile the modules needed by the package, i.e. the database module and the modified versions of `f2c` and `gcc`;
2) `CONFIGURE` to configure the scripts so that they can be activated by the local Web server through CGI.

To run WTH one has to activate a single script `wth.pl` which parses via a menu a specified directory tree, creating the complete infrastructure for the navigation (token database, Web pages etc.)

3 Future Developments and Conclusions

We have presented a powerful tool to browse source code on the Web, for documentation, maintenance and development purposes. Our tool is simple to install and use, does not require commercial components, can run almost everywhere being written in PERL, employs modified GNU compilers for the parsing process. Our tool does not provide, by design, Web navigation of other

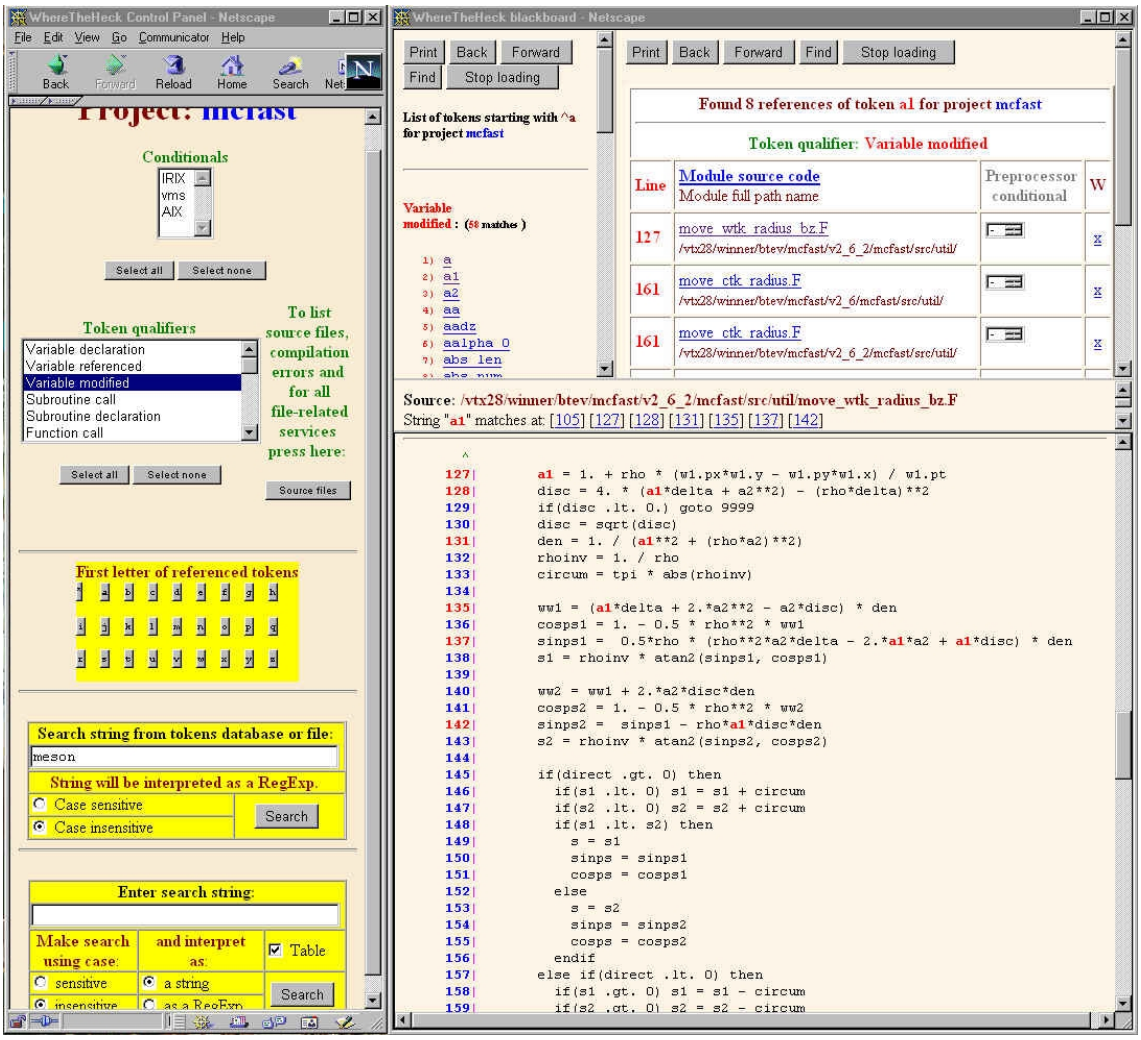

Figure 1: An Example of WhereTheHeck session (this is a color picture)

informations, like manuals or object diagrams; there exist packages trying to do that (as LIGHT [4]) which, in our opinion, require more effort for installation and use and are not easily portable. We decided to develop a less ambitious tool with high degree of portability and user-friendliness. The current version of WTH has been used in test mode by several collaborations (BTeV, CHORUS, FOCUS) and the user base is growing. We plan to complete the language coverage, which is now very limited for the C++, and to improve the configurability of the whole package. The dependence from Javascript could be removed too, but at the moment we do not have a schedule for that. We intend to distribute a production version of WTH during the year in order to make it publicly available.

References

1 http://www.roma1.infn.it/ winner
2 J.Friedl, Mastering Regular Expressions, O'Reilly (1997).
3 http://www.gnu.org/
4 http://light.cern.ch