CONTEXT HELP SEARCHING FOR wxHASKELL GRAPHIC LIBRARY

Ján KOLLÁR and Emília PIETRIKOVÁ
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice,
Letná 9, 042 00 Košice, Slovak Republic, tel.: +421 55 602 2577,
e-mail: {jan.kollar, emilia.pietrikova}@tuke.sk

ABSTRACT
This paper deals with two different approaches in programming sphere, and joins them in one entity through its practical part. One approach is context searching that is applied on the second approach, functional graphic library wxHaskell that is designed and implemented within Eclipse development environment through the plug-in extension. As searching object lies at the wxHaskell documentation, it is lexically analyzed, what forms a hashing table determined to searching for records.

Keywords: *Eclipse, hashing, Haskell, context help, support, wxHaskell*

1. INTRODUCTION

The task of searching is one of the most frequent operations in computer programming. Context searching refers to proactive capturing of user's information need through automatic augmenting user's query with information extracted from the search context. There exist several basic variations of the searching theme, and many different algorithms have been developed on this subject, while the basic assumption is that the collection of data, among which a given element is to be searched, is fixed [11].

Searching in explicitly stored amount of data include many algorithms that use a variety of search data structures, such as the simple linear search, binary search, and hash tables. Each hash algorithm is based on a simple item field, called associative array, and hash table as well. Hash table, also entitled as a transformation table, is a special data structure using hash function to efficiently map certain identifiers to associated values [11].

A pair of different keys with the same hash values is a hash collision. Ideally the hash function should map each possible key to a different index. However, this ideal is rarely achievable in practice, so most hash table designs assume that hash collisions are normal occurrences and must be accommodated in some way [9].

In this paper, main object of the context searching is a functional graphic library wxHaskell. Central to functional programming is the idea of a function that computes a result that depends on the values of its inputs [3]. wxHaskell is a graphical user interface (GUI) library for Haskell that is built on wxWidgets: a free industrial strength GUI library for C++ that has been ported to all major platforms, that retains the native look-and-feel of each particular platform [7].

wxHaskell consists of two libraries, WXCore and WX. The WXCore library provides the core interface to wxWidgets functionality. It exposes about 2800 methods and more than 500 classes of wxWidgets [2]. Using this library is just like programming wxWidgets in C++ and provides the raw functionality of wxWidgets. For connection with the original library, WXCore uses standard Foreign Function Interface (FFI) depicted on Fig. 1 [8]. The WX library is implemented on top of WXCore and provides many useful functional abstractions to make the raw wxWidgets interface easier to use.

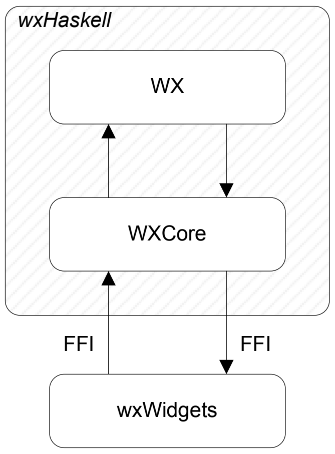

Fig. 1 Interface of wxHaskell graphic library above wxWidgets

2. DOCUMENTATION AND HADDOCK TOOL

Documentation of wxHaskell library is generated through the Haddock tool that is primarily intended for documenting hierarchical Haskell libraries with annotated source code. Haddock allows writing documentation annotations next to the definitions of functions and types in the source code, in a simple way that is an assumption for automatic generation of a comprendious documentation. The documentation generated in this manner is fully hyperlinked.

The wxHaskell documentation consists of three types of pages with the exact structure, as shown in Fig. 2:

- Main page with general description of the library, that links to the Index page, Module pages, and other useful pages. Module pages are specified with part of the HTML code: `STYLE=padding-left: 1.25em; width: 44em,` and separated from foot of the page with: `CLASS=s15`. These code specifications are used as part of lexical analysis applied in the design and implementation Section 3.

- Index page of keywords, that might be searched, and links to Module pages containing the keywords. This page is not consideranted in the design and implementation part.
- Module page with general description of a module, and detailed specifications of each definition, argument, and other components.
  Each specification is listed in a separate block. Statement blocks bound with code `CLASS=s15` and definitions begin with characters `CLASS=decl`, while the first word is always treated as a keyword dedicated for the searching process.

These auxiliary codes are cut-out of the HTML tags referring to particular pages of the wxHaskell documentation. Lexical analysis considerates only the code sections, not the whole tags.

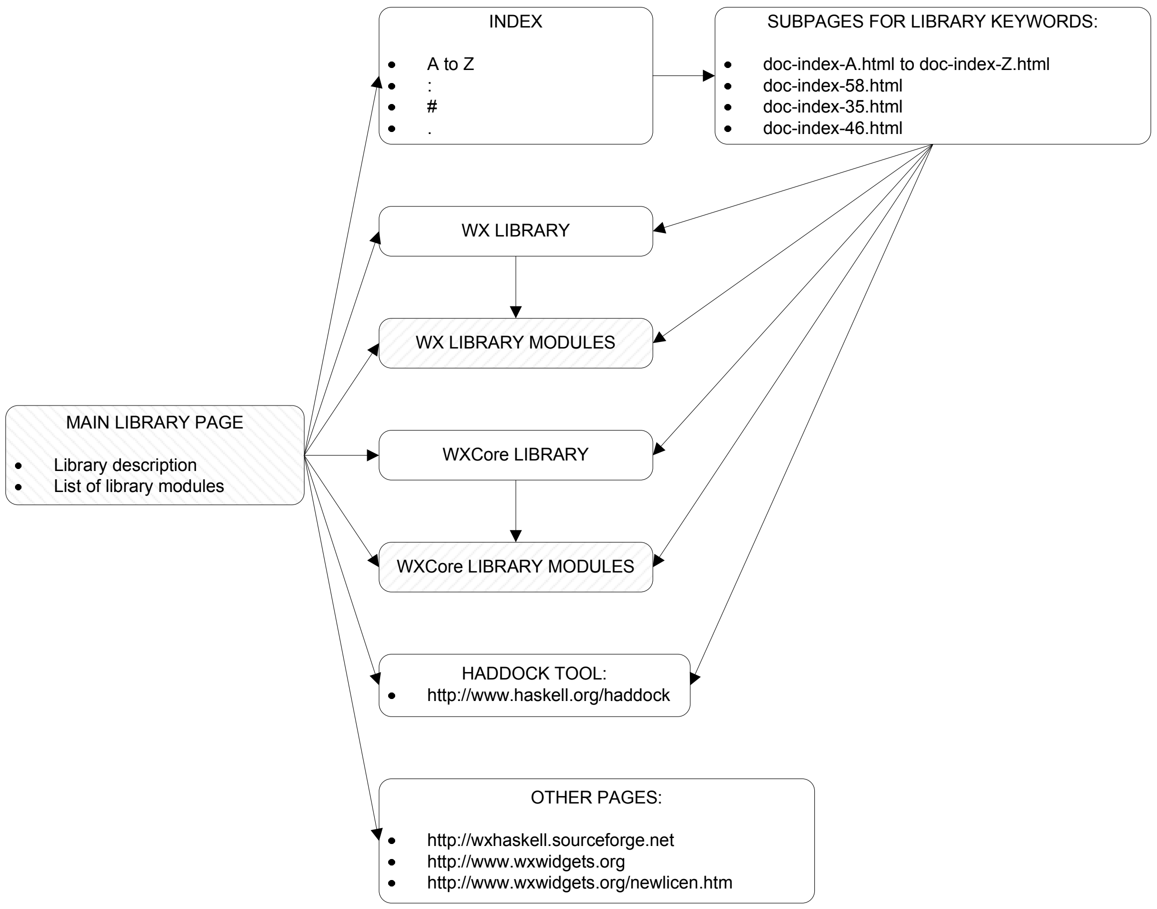

Fig. 2 References contained in main page of the wxHaskell documentation

3. PLUG-IN DESIGN

Context searching in the wxHaskell library was designed as a plug-in extending Eclipse development environment called *SearchFP*, using the Haddock documentation, Google, and user's auxiliary code database.

3.1. Requirement specification

Basic assumption for the plug-in design consists of searching purpose and expected search result, what divided searching in three areas: *Documentation Search* for a particular keyword description, auxiliary *Code Search* pertaining to a keyword, and *Google Search* through the Internet.

The main requirement was *time saving* in the course of manual searching in the documentation webpage what implies two requirements: *understandability* and *reusability*. Additional two requisitions form *actual information* and *open perspective* for other libraries, that could be provided with a standard configuration `.ini` file. The file would keep url address (addresses) of the library (libraries), and literals necessary for lexical analysis. As one of the plug-in preconditions is Code Search, the last requirement is a possibility to form user's *own* auxiliary code *database* through `.xml` file.

3.2. Use-cases

On the basis of mentioned requirements, a simple use-case diagram was designed (see Fig. 3). Each use-case scenario starts in the Eclipse environment, as the user uses editor of the Eclipse perspective. User is able to mark a specific word, having three possibilities for searching: searching in the documentation, local code database, or searching via Internet Google search engine. Search results can be displayed in the perspective of view, new file in the editor, or web browser of Eclipse environment.

Special case was designed for adding new code to the local database, aimed for potential search event. User is capable of selecting a sequence of code words and adding them to a particular `.xml` file. After saving the code, it is possible to search for it immediately.

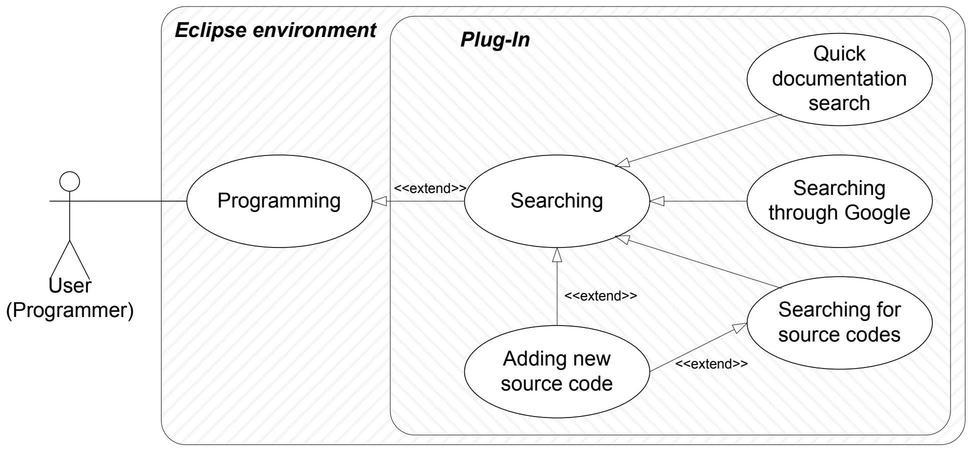

Fig. 3 Use-Case diagram for the SearchFP Plug-In

3.3. Plug-In development support in Eclipse environment

Eclipse is Java application providing functionalities of a loader. It is a simple program surrounded with amount (hundreds to thousands) of plug-ins.

In comparison with common Java applications, plug-in development is different. Plug-in is another Java application that extends basic functionalities of the environment in a certain way. Each plug-in can utilize services of other plug-ins and it can provide its own functionality for the remainder. All the plug-ins are loaded during runtime upon user's selection.

Eclipse environment has opened platform, what means that it is designed in a way that its interfaces enable a user to extend it.

3.4. Classes design

Classes of the SearchFP plug-in were designed according to simplified class diagram shown in Fig. 4. Classes of the `searchfp` package were generated automatically, providing direct connection to mechanisms that permit extension of the environment. These classes implement plug-in maintenance, and life cycle.

Other classes were implemented separately. Key part of the implementation lies at `HtmlParser` class designed for lexical analyzer, parsing main page of the wxHaskell documentation and a module page. Lexical analysis is based on documentation structure mentioned in Section 2.

Method `parIndex()` acquires url addresses of library subpages (modules), that would be stored in a string vector, through distinguishing of auxiliary keywords as lexical symbols, marked by bold font. If a lexical symbol has an attribute value, it is displayed within angular brackets [5,6].

Auxiliary keywords of the file are defined by equations (1).

$$\begin{aligned} \mathscr{K}\left[\!\!\left[\begin{array}{c} <\mathtt{TD\ STYLE} = \backslash\text{"}\mathtt{padding-left:} \\ \mathtt{1.25em;width:44em}\backslash\text{"}> \end{array}\right]\!\!\right] & \\ &= \mathbf{key3} \\ \mathscr{K}\ [\![\ \mathtt{HREF}\ ]\!] &= \mathbf{key4} \\ \mathscr{K}\ [\![<\mathtt{TD\ CLASS} = \backslash\text{"}\mathtt{s15}\backslash\text{"}>]\!] &= \mathbf{key5} \end{aligned} \tag{1}$$

Syntax of recognized url address value *Sval* is defined by grammar (2).

$$\begin{array}{lcl} Sval & \to & Letter\ \{\ Letter \mid Digit\ \}\ Suffix \\ Suffix & \to & \mathtt{.html} \\ Letter & \to & \mathtt{a}\ \ldots\ \mathtt{z} \mid \mathtt{A}\ \ldots\ \mathtt{Z} \\ Digit & \to & \mathtt{0}\ \ldots\ \mathtt{9} \end{array} \tag{2}$$

The semantics of the transformation of url reference to url symbol is given by equation (3).

$$\mathscr{U}rl\ [\![<\mathtt{A\ HREF} = \text{"}Sval\text{"}>]\!] = \mathbf{url}\langle Sval\rangle \tag{3}$$

where *Sval* is a string representing particular url address, according grammar (2).

At each module page, `parse()` method acquires every keyword and its description, and stores them in a hash table. One keyword can match one or more descriptions. The method distinguishes auxiliary keywords, searched keywords, and the descriptions as lexical symbols [4,6]:

Auxiliary keywords of the file are defined by equations (4).

$$\begin{aligned} \mathscr{K}\ [\![<\mathtt{TD\ CLASS} = \backslash\text{"}\mathtt{s15}\backslash\text{"}>]\!] &= \mathbf{key1} \\ \mathscr{K}\ [\![\ \mathtt{decl}\ ]\!] &= \mathbf{key2} \end{aligned} \tag{4}$$

Syntax of recognized keyword value *Kval* intended for searching *Tag* is defined by grammar (5).

$$\begin{array}{lll} Tag & \rightarrow & < \ Text \ > \\ Text & \rightarrow & Printable\_Char \ \{ \ Printable\_Char \ \} \\ Kval & \rightarrow & (\ Letter \mid Digit \ ) \ \{ \ Letter \mid Digit \ \} \\ Letter & \rightarrow & \texttt{a...z} \mid \texttt{A...Z} \\ Digit & \rightarrow & \texttt{0...9} \end{array} \qquad (5)$$

where *Printable_Char* is arbitrary printable character except characters $<$ and $>$.

The transformation is defined by equations (6) and (7).

$$\begin{array}{lll} \mathscr{K}eyword \ [\![\ Kval \ \{ \ Text \ \} \ ]\!] & = & \mathbf{keyword}\langle Kval \rangle \\ \mathscr{K}eyword \ [\![\ Tag \ ]\!] & = & \varepsilon \end{array} \qquad (6)$$

where *Kval* is a string representing first word in the text without html tags, and ε is an empty symbol.

$$\begin{array}{l} \mathscr{H}tml \ [\![\ < \texttt{A HREF} = Text > \{ \ Text \ \} < /\texttt{a} > \ ]\!] \\ \quad = \mathbf{html}\langle < i > \{ \ Text \ \} < /i > \rangle \\ \mathscr{H}tml \ [\![\ < \texttt{IMG} \ Text > \ ]\!] \ = \ \varepsilon \end{array} \qquad (7)$$

where image tags are removed and hyperlink tags are replaced with font tags. The attribute value of **html** symbol represents the description of recognized keyword, and ε is an empty symbol.

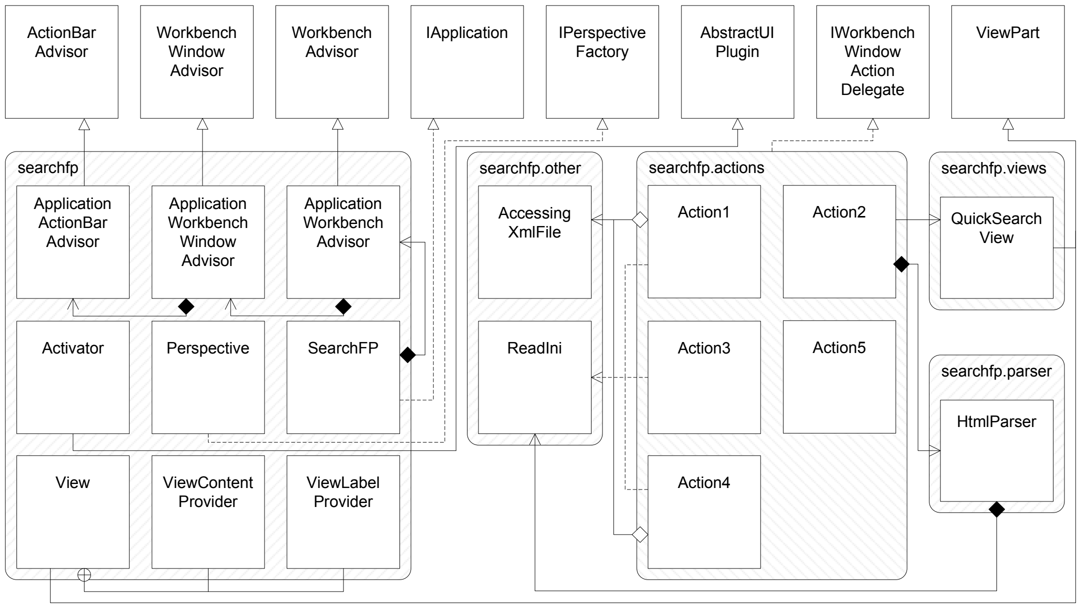

Fig. 4 Class diagram for the SearchFP Plug-In

3.5. Code database

Database dedicated for storing of auxiliary codes represents a simple `.xml` file, while name of the file is stored in the configuration file with predefined name *in.xml*.
The structure of the database file is as follows

- `<examples>, </examples>` ...tags representing the beginning and end of the file.
- `<codes>, </codes>` ...tags representing one record with content consisting of one keyword and one sequence of source code.
- `<keyword>, </keyword>`...tags representing keyword dedicated to code search event.
- `<code>, </code>`...tags representing source code pertaining given keyword. The code would be displayed as a searching result in a newly-formed temporary file with `.hs` extension.

Searching for the code is preceded by parsing of the `.xml` file, that results in creation of a new hash table with keywords and codes. Each source code is stored in a vector of strings, and each keyword matches one code vector only.

4. EVALUATION

Main advantages of the Eclipse SearchFP Plug-In v1.0.0 are as follows:

1. Fast and effective search directly in the environment.
2. Transparent form of the results view.
3. General usage possibility concerning other libraries and languages.

On the other hand, the drawbacks are:

1. Slow initialization and parsing.
2. Necessity of Internet connection.
3. Need for saving sample codes in order to search for them.

5. CONCLUSION

The current SearchFP Plug-In for Eclipse includes searching for the keyword description in wxHaskell documentation and sample codes database, additional web-based search, and new sample code storing availability as well.

Lexical analysis considers the first word in each separate block of documentation as a keyword, that is described by the rest of the block. Each description is stored by the hash table in the original html format, so search results could be displayed in a transparent form, not differentiating from the initial documentation form significantly. Each search event is used within the Eclipse environment, what makes searching fast and effective.

Plug-in usage requires Internet connection with consequence of slow initialization consisting of downloading and parsing the documentation. This could be solved by modification of the standard configuration file `searchfp.ini`, by replacement of library url `urls` with current documentation address localized on user's local system. However, this step disables the Google Search option.

Due to analysis focused on Haddock documentation, not wxHaskell documentation itself, utilization of the SearchFP Plug-In is more general, so it could be used for searching in any other library processed by the Haddock tool. As lexical units are stored in the configuration file as well, user is able to modify them according to his current needs.

By far the greatest need for research is for further empirical studies of features, that would make the initialization faster, actual, and Internet independent at the same time.

Preliminary experiments show pros and cons of context search based on lexical units and web-page structures. Constructed product forms a technical support for more sophisticated and refined applications out of consideration to automation of static knowledge acquisition [1] or online applications reporting phishing suspects [10].

ACKNOWLEDGEMENT

This work is the result of the project implementation: Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120020) supported by the Research & Development Operational Program funded by the ERDF.

REFERENCES

[1] AL-HASHIMY, A.S.H.: Lexical Units in Ontological Semantics, In: ISCIII '07 International Symposium on Computational Intelligence and Intelligent Informatics (2007).

[2] Functional Graphic Library wxHaskell: Documentation, http://wxhaskell.sourceforge.net/doc

[3] KOLLÁR, J.: Functional Programming (in Slovak), Elfa, Košice (2009).

[4] KOLLÁR, J.: Compilers (in Slovak), Elfa, Košice (2009).

[5] KOLLÁR, J. – PORUBÄN, J. – VÁCLAVÍ K, P.: Compiling the Process Functional Programs, In: Proceedings of the 3rd Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence, pp. 283–296 (2005).

[6] KOLLÁR, J. – FORGÁČ, M. – PORUBÄN, J.: Adaptiveness of Software Systems Using Reflection, In: Acta Electrotechnica et Informatica, pp. 53–57, Elfa, Košice (2007).

[7] LEIJEN, D.: wxHaskell: A Portable and Concise GUI Library for Haskell, In: Proceedings of the 2004 ACM SIGPLAN workshop on Haskell (2004).

[8] van NOORT, T.: Building GUIs in Haskell - Comparing Gtk2Hs and wxHaskell, In: Sofware Technology Colloquium, Utrecht University (2007).

[9] SKIENA, S.: The Algorithm Design Manual, Springer-Verlag, Heidelberg (2008).

[10] WENYIN, L. – GUANGLIN, H. et al.: Phishing Web page detection, In: Eighth International Conference on Document Analysis and Recognition (2005).

[11] WIRTH, N.: Algorithms and Data Structures, Prentice Hall, New Jersey (1985).

Received July 15, 2010, accepted November 24, 2010

BIOGRAPHIES

Ján Kollár is Full Professor of Informatics at Department of Computers and Informatics, Technical university of Košice, Slovakia. He received his M.Sc. summa cum laude in 1978 and his Ph.D. in Computer Science in 1991. In 1978-1981 he was with the Institute of Electrical Machines in Košice. In 1982-1991 he was with Institute of Computer Science at the P.J. Šafárik University in Košice. Since 1992 he is with the Department of Computer and Informatics at the Technical University of Košice. In 1985 he spent 3 months in the Joint Institute of Nuclear Research in Dubna, USSR. In 1990 he spent 2 months at the Department of Computer Science at Reading University, UK. He was involved in research projects dealing with real-time systems, the design of microprogramming languages, image processing and remote sensing, dataflow systems, implementation of programming languages, and high performance computing. He is the author of process functional programming paradigm. Currently his research area covers formal languages and automata, programming paradigms, implementation of programming languages, functional programming, and adaptive software and language evolution.

Emília Pietriková is PhD Student at Department of Computers and Informatics, Technical University of Košice, Slovakia. He received his MSc. in 2010. The subject of his research is metaprogramming, programming paradigms, and exploiting functional paradigm in systems evolution.