

[In *Proceedings of 5th International Workshop on Engineering Hypertext Functionality into Future Information Systems (HTF5)*, Kyoto, Japan, April 1998.]

Utilising Hypertext Functionality in Programming Environments

Jun Han , Peninsula School of Computing and Information Technology, Monash University, McMahons Road, Frankston, Vic. 3199, Australia. jhan@monash.edu.au

Abstract

Proper management and presentation of software artifacts (including programs) is an important issue for software engineering tools and environments. In this paper, we introduce an approach to using hypertext functionality as the basis of structuring, managing and presenting programs in a programming environment, to achieve better support for program comprehension and manipulation. The major issues addressed include fine-grained structuring and organisation of programs, representation of program relationships, and program presentation and navigation. The approach is also applicable to other software artifacts such as analysis and design documents.

1 Introduction

Software development involves a large number of artifacts, including requirements documents, design documents, programs, testing suites, and so on. Proper management and presentation of these software artifacts is an important issue for software engineering tools and environments. It facilitates better software comprehension, and assists various software development activities. In this paper, we focus on the issue of achieving better management and presentation of programs in programming environments through the use of hypertext functionality.

Traditionally, the programs for a software system are organised in the operation system's file structure. Most programming tools and environments also rely on this file-based management of programs, with additional presentation, editing and analysis capabilities. The structuring of programs into different files is very often influenced by the programming language and the compiler used, and is determined by the developer usually according to his/her own judgement. For example, the reason for a file to contain one or more classes in C++ programming is often not clear in a project. In addition, the logical relationships between the various program files are often implicit. That is, these relationships are embedded in the file body through language mechanisms such as the use of a class in one file by another file, and/or are codified in separate configuration management files such as the *makefiles* in the Unix environment. Relationships captured and managed in such a manner can not be easily used by the developers in assisting their programming activities, such as finding out all the places in all files that use a given class. Besides, the navigation between program segments (within same files or between different files) is often through browsing and searching. The exploitation of explicitly recorded logical relationships between program segments for navigation and other analytical purposes such as change propagation would greatly assist the developers' activities.

Recent programming languages and related compilers and tools provide improved support for program manipulation and management. An example is the clear class-file correspondence in the Java programming language/system. In general, however, there is still much to be desired for. In this

paper, we investigate the use of hypertext functionality to achieve better structuring, management, presentation, navigation and analysis of programs and their relationships. In the next section we present some of the major considerations in providing hypertext support for programming. In section 3, we outline our effort on a hypertext-based programming environment for C++.

2 Hypertext support for programming

Hypertext provides a natural way of managing and presenting inter-related information. Realising the need to support inter-related software artifacts, a number of efforts have investigated how to incorporate hypertext functionality into software engineering tools and environments. These efforts have resulted in systems like Dynamic Design [Bigelow88], DIF/Ishys [Garg90, Garg89], HyperCASE [Cybulski92] and HyperWeb [Ferrans92]. These systems mainly focus on the coarse-grained management of software artifacts. In this section, we show that hypertext functionality can also provide better support for fine-grained management of software artifacts, in particular, programs. The major issues to be discussed include logical structuring and organisation of program segments, explicit capture and management of program relationships, and program presentation and navigation support.

Program structuring and organisation. Programs have a logical structure based on the various program units supported by the programming language and the development methodology. For example, the programs for a software system may be organised into subsystems, modules, classes, functions at various abstraction levels. The program developers construct, modify and comprehend the programs with the help of their logical structures based on these program units and their structural and logical relationships.

Ideally, the structuring and organisation of programs in a programming environment should directly reflect the logical program units and their relationships to aid the program developers. However, the traditional way of organising programs into files with the program segments (in the files) being streams of text does not reflect the logical structure of the programs. Consequently it hinders the program developers' activities. For example, a developer may need to browse a number of files to locate a class definition, and may resort to other means (such as documentation in a separate file) to record the relationships between the program segments in the various files.

The use of relevant hypertext functionality can significantly improve the structuring and organisation of programs, to directly reflect the logical program units and their relationships and to aid the program developers' activities. Specifically, logical program units at different abstraction levels can be captured in (atomic or composite) hypertext nodes, so that these units are easily identifiable in the programming environment. Besides, these units/nodes are hierarchically related through hypertext links as their structural relationships dictate (for the capture of the semantic relationships, see the section on ``program relationships" below). As such, the direct correspondence between program units and hypertext nodes and the explicit capture of the compositional relationships facilitate easy comprehension of the programs.

In structuring and organising C++ programs, for example, we may choose to represent as hypertext nodes the following types of program units: system, subsystem, class, function, and main function. The system node is composed of handles (anchors) to the subsystems (nodes). A subsystem node contains handles to the classes (nodes) forming the subsystem. A class node contains the content of a class definition, except that the function definitions are represented by handles to the corresponding function nodes. Finally, a function node contains the definition of a function and the main function node contains the main function. This structuring and organisation of programs directly capture the logical program units and results in hypertext nodes of reasonable size for comprehension. The inter-node relationships captured through the ``handle" concept reflects the hierarchical composition structure of the programs.

Program relationships. In addition to the compositional relationships between program units, there are many other relationships between program units or between parts of a single program unit. For example, there are relationships between the uses and the definition of a class. It would be very helpful to the program developers if the programming environment directly captures such relationships and allows simple navigation from a use of a class to the class's definition (instead of browsing or searching). There may also be relationships that may not be syntactically implied by the program code but are useful when explicitly captured. An example would be the relationships among a cluster of classes that represent the use of a design pattern. Furthermore, relationships may be introduced for program management purposes.

In general, the relationships between program segments will provide great assistance to the program developers when they are *explicitly* captured and can be *directly* used by the program developers. However, this has not been the case in existing programming environments. In most cases, the developers have to rely on browsing and searching to locate related program segments.

With hypertext support, the various relationships between or within program units can be explicitly captured or represented with hypertext links. As such, the developers may directly use these links to locate related program information. Relationships that are implied by textual syntax (e.g., class definition/use) can be introduced and managed by purposely built analytical tools. Other relationships have to be maintained by the developers.

With C++ programs, for example, we may capture the variable definition/use, class definition/use, class inheritance and other developer-introduced (e.g., pattern-related class cluster) relationships. The developers can use these relationships to locate related information and navigate around the "program space" with semantic understanding.

Presentation and navigation. As discussed above, the program units captured in hypertext nodes can be presented directly to the developers for browsing and editing, and the relationships captured by hypertext links can be used for the developers to navigate around the program space. Besides, additional support in terms of program presentation and navigation would be desirable to the developers. One type of such support is program views that are dynamically generated from the basic program units and their relationships based on certain criteria. For example, one possible type of view for C++ programs is a view that shows the inheritance hierarchy rooted at a given class. Another view would be a view that contains a class and all its friends (functions and classes). Such groupings (*i.e.*, program views) of program information are dynamically generated (rather than being static program units) because they are usually collections of existing program units and reflect particular viewpoints of the program space. As such, there is no need to maintain duplicate information and manage its consistency.

The program views can be achieved in a hypertext-based programming environment through the use of such hypertext mechanisms as attributes and queries. Conforming to the intent of program views, such dynamically generated hypertext nodes are not static nodes. Therefore, we do not need to maintain duplicate information and deal with its consistency in the light of modification. Related issues include view-based relationships and navigation, and view-based update of program content.

3 A hypertext-based C++ programming environment

Based on the approach proposed in the previous section, we have been developing a prototype hypertext-based programming environment for C++. The prototype uses the HyperWave system as the underlying implementation platform. The range of C++ program units and their relationships are identified and are implemented as outlined above. Typical C++ program views are also identified and supported. We are currently working on view-based updates and additional semantic program analysis capabilities.

4 Conclusions

In this paper, we have proposed an approach to using hypertext functionality as the basis of structuring, managing and presenting programs in a programming environment. With this approach, the logical structure and relationships of the programs are explicitly captured and additional program views can be identified and generated, which are all for the direct use by the program developers. As such, it provides better support for program comprehension and manipulation. According to the approach, a hypertext-based C++ programming environment has been prototyped. During our research effort, certain shortcomings of hypertext have also been uncovered in terms of supporting programming in particular and software engineering in general [Han95].

References

[Bigelow88] J. Bigelow. Hypertext and CASE. *IEEE Software*, 5(2):23--27, March 1988.

[Cybulski92] J.L. Cybulski and K. Reed. A hypertext based software-engineering environment. *IEEE Software*, 9(2):62--68, March 1992.

[Ferrans92] J.C. Ferrans, D.W. Hurst, *et al.* HyperWeb: A framework for hypermedia-based environments. In H. Weber, editor, *SIGSOFT92, Proceedings of 5th ACM SIGSOFT Symposium on Software Development Environments*, pages 1--10, December 1992. Appeared as *Software Engineering Notes*, 17(5), 1992.

[Garg89] P.K. Garg and W. Scacchi. Ishys: Designing an intelligent software hypertext system. *IEEE Expert*, pages 52--63, FALL 1989.

[Garg90] P.K. Garg and W. Scacchi. A hypertext system to manage software life-cycle documents. *IEEE Software*, 7(3):90--98, May 1990.

[Han95] J. Han. The role of hypertext in CASE environments. *Journal of Computing and Information*, pages 947--961, 1995.