Versioned Hypermedia Can Improve Software Document Management*

Tien Nguyen
Dept. of EECS
Univ. of Wisconsin, Milwaukee
tien@cs.uwm.edu

Satish Chandra Gupta
Dept. of EECS
Univ. of Wisconsin, Milwaukee
scgupta@cs.uwm.edu

Ethan V. Munson
Dept. of EECS
Univ. of Wisconsin, Milwaukee
munson@cs.uwm.edu

ABSTRACT

The Software Concordance project is addressing the software document management problem by providing a fine-grained version control model for software documents and their relationships using hypermedia versioning. A set of tools needed to maintain, visualize and analyze software documents is being constructed. This short paper presents research issues, initial results and a scheme for using hypermedia versioning and time stamps to automate detection of possible semantic non-conformance among software artifacts.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments; H.5.4 [**Information Interfaces and Presentation**]: Hypertext / Hypermedia

General Terms

Documentation, Management

Keywords

hypermedia, version control, software engineering

1. INTRODUCTION

Software engineering is a discipline addressing the application of a systematic, quantifiable approach to the development, operation, and maintenance of software. During the process of implementing and designing large-scale software systems, software engineers produce a large variety of heterogeneous software artifacts such as requirements documents, feasibility studies, design specifications, source code, module documentation, test plans, bug reports and sales plans. One challenging task for software engineers is to manage this collection so that the information can be quickly accessed and consistently organized. One important aspect of this task is to capture and manage the logical relationships among software documents.

*This research was supported by the U. S. Department of Defense and by NSF CAREER award CCR-9734102.

HT'02, June 11-15, 2002, College Park, Maryland, USA.

In the context of a large-scale software system with hundreds of artifacts and thousands of implicit and explicit relationships, that all evolve over the time, the task of keeping all of them semantically consistent and conformant with each other is not easy. The lack of powerful tools for document and relationship management forces developers to use ad hoc methods, including paper notes or memorization, which can lead to cognitive overhead and errors. This problem hinders developers from having a full understanding of the system and reduces their effectiveness.

The Software Concordance (SC) is a software development environment that is attempting to address the relationship management problem by providing a fine-grained revision control model for both software artifacts and their relationships, an extensible set of types of relationships and a set of tools needed to analyze, retrieve, visualize and maintain them. A SC prototype has been constructed that uses a uniform document model to integrate program analysis and hypermedia services for both program source and documentation [6]. Current research on SC is exploring the use of link and anchor versioning to automate analysis of document conformance. This short paper presents the initial results of this research and the design plan for the next prototype.

2. RELATED WORK

A wide variety of research has attempted to manage software documents and their logical relationships in a software development environment. Examples of this work include SODOS [8], DIF [5], HyperPro [9], DynamicDesign [1] and CHIME [3]. These systems have a number of important limitations: they do not address the conformance of the semantic content of different software documents; they do not support versioning in the presence of relationships; the set of document types and relationships that they support is fixed and pre-defined; and their versioning and anchoring are not arbitrarily fine-grained. Some of the systems have scalability problems due to centralized document storage.

Versioning is also an important research issue within the hypermedia community. A wide range of approaches have been used to address the issue. Hicks et al. [2] and Whitehead et al. [11] proposed general version support frameworks for open hypermedia systems. Haake [7] developed a version support environment using a task-based version model to reduce cognitive overhead. Durand [4] introduced a change-oriented version model for hypermedia systems in order to support collaboration. Vitali [10] provides a brief survey of this research, while Whitehead [12] presents a careful, formal analysis of the domain.

3. DETECTING NON-CONFORMANCE

Software systems evolve over time as the system is constructed from scratch and then as the system goes through a series of release

versions. Thus, any system which purports to help with the management of software documents must support this evolution. We are particularly interested in helping developers detect when different software documents become *non-conformant* as the system evolves. Our current solution to this problem uses typed links along with versions and timestamps for anchors and links.

3.1 Link types

In general in SC, links are semantically directional but can be navigated in any direction and may connect more than two anchors. They are divided into three broad categories and the category affects both directionality and the number of anchors linked. *Non-conformance links* are intended to support navigational relationships that have little or no relevance to determining agreement between documents, such as table of contents and index links. Non-conformance links are n-ary and their direction is user-specifiable. *Causal links* are used to represent semantic dependencies between entities where one document "causes" another. They have named types (like "requires," "specifies," "responds to,") and are always binary and directional. For example, item A in a design document may *require* the use of class B (A causes B) and function C may *respond to* bug report D (D causes C). Non-causal links are n-ary and non-directional. The distinction between causal and non-causal links is important because causality can provide clues that can help with detection of non-conformance.

3.2 Timestamps and versioning

Anchors in SC denote the regions of interest within a document and are versioned. But unlike some hypermedia systems, SC anchors are object-bound and view-independent. In version space, when the software documents evolve, the anchors might change as well, creating new versions. Anchors may also change when users redefine their positions in the documents, making the semantic contents of end-points of the links changed. Links in SC are also versioned, since a link is a set of anchors. Thus, each anchor or link has a version tree. Every addition, deletion or replacement of an anchor for a link causes a new version of the link created.

In SC, the representation of anchors and links uses standard concepts such as identifier, attributes and values. Both anchors and links also have version information including version ID, version creator, and a timestamp. In addition, links have a *validation timestamp* that records the time when the link version was last validated by human inspection. When a link gets a new version, the validation timestamp is copied from the previous version.

3.3 Conformance analysis

We have developed a scheme for detecting possible conformance problems in software document relationships based on the timestamps of anchors and links. This scheme produces a numeric rating of the likely seriousness of the problem on a scale from zero (likely conformance) to ten (likely non-conformance). However, these numbers are just heuristic values and have yet to be validated systematically.

3.3.1 Causal links

Causality implies a partial ordering in time. This partial order results from the fact that if A causes B, then A must have occurred before B. In the case of causal links, three timestamps are relevant: t_{src}, the version timestamp of the source anchor; t_{dest}, the version timestamp of the destination anchor; and t_{valid} the validation timestamp of the link. We have made a complete case analysis of the order of these timestamps, defining conformance ratings for each case. For example, if $t_{valid} > t_{dest} > t_{src}$ then the destination is newer than the source and the link relationship was validated by a human sometime after the last changes to both and the rating is zero. In contrast, if $t_{dest} > t_{valid} > t_{src}$ then then the destination has changed since the last validation and the rating is five (5).

3.3.2 Non-causal links

When causality is not available to provide clues, we turn to a more formulaic approach. Assume that a non-causal link connects N anchors and let t_i be the timestamp of the i^{th} anchor. Then, if there exists any t_i such that $t_i > t_{valid}$ then

$$D = \alpha + (n/N) \times (10 - \alpha)$$

Otherwise, $D = 0$. α is an adjustable parameter representing the minimum rating when the timestamps suggest that a problem exists. The idea is that this minimum rating may be affected by other factors such as the frequency of changes to the link's anchors.

4. CONCLUSION

The Software Concordance project is exploring new ways to use hypermedia technology to improve the management of software documents. Typed links, versioning, and timestamps have been found to be useful tools for detecting problems in the conformance of software documents.

5. REFERENCES

[1] J. Bigelow and V. Riley. Manipulating source code in DynamicDesign. In *Proceedings of the 1987 ACM Conference on Hypertext*, pages 397–408, 1987.

[2] David L. Hicks, John J. Leggett, Peter J. Nurnberg and John L. Schnase. A hypermedia version control framework. *ACM Transactions on Information Systems*, 16(2):127–160, April 1998.

[3] P. Devanbu, Y.-F. Chen, E. Gansner, H. Müller, and J. Martin. Chime: customizable hyperlink insertion and maintenance engine for software engineering environments. In *Proceedings of the 1999 International Conference on Software Engineering*, pages 473–482, 1999.

[4] David Durand. *Palimpsest: Change-oriented concurrency control for the support of collaborative applications*. PhD thesis, Boston University – Boston, 1999.

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

[6] Satish Chandra Gupta. Bringing multimedia to source code: A document interface to program analysis services. Master's thesis, University of Wisconsin – Milwaukee, December 2001.

[7] Anja Haake and David Hicks. VerSE: Towards hypertext versioning styles. In *Proceedings of the 1996 ACM conference on Hypertext*, pages 224–234, 1996.

[8] Ellis Horowitz and Ronald Williamson. SODOS: a software documentation support environment—its use. *IEEE Transactions on Software Engineering*, SE-12(11):1076–1087, November 1986.

[9] Kasper Østerbye and Kurt Nørmark. An interaction engine for rich hypertext. In *ECHT '94, Proceedings of the 1994 ACM European conference on Hypermedia technology*, pages 167–176, 1994.

[10] Fabio Vitali. Versioning hypermedia. *ACM Computing Surveys*, 31(4es), December 1999.

[11] E. James Whitehead, Jr., Kenneth M. Anderson, and Richard N. Taylor. A proposal for versioning support for the Chimera system. In *ECHT '94, Proceedings of the Workshop on Versioning in Hypertext Systems*, pages 45–54, 1994.

[12] E. James Whitehead, Jr. *An Analysis of the Hypertext Versioning Domain*. PhD thesis, University of California, Irvine, September 2000.