Bridging Javadoc and design documentation via UML diagram image maps

Asko Soukka, Tuukka Hastrup, Tuomas J. Lukka and Benja Fallenstein
Hyperstructure Group
Agora Center, University of Jyväskylä
P.O. Box 35, FIN 40014
humppake@iki.fi, Tuukka@iki.fi, lukka@iki.fi, b.fallenstein@gmx.de

ABSTRACT

We present a navigational aid for documentation used in software development. Based on using human-made UML diagrams as multi-ended links, we hypertextually connect two distinct areas of documentation: design documents and documentation embedded in the program code, such as Javadoc.

Categories and Subject Descriptors

H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia—*Navigation*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Evolutionary prototyping*

General Terms

Design,Documentation,Human Factors

Keywords

imagemap,maps,bi-directional linking,link authoring,UML,spatial hypertext

1. INTRODUCTION

Software projects manage a large base of evolving documentation, whose parts are interrelated on many levels. Design documentation shows architectural views on a more general level, while the source code usually contains embedded documentation, such as Javadoc [3], giving details on the concrete classes and methods. The generation of easily navigable hypertext from the embedded documentation is essentially a solved problem for several languages [2, 3, 6]. Similarly, requirements tracing — managing relations between requirements and design decisions — is relatively well developed [4].

Often missing are the connections between the design documentation and the embedded documentation. This can be daunting to a newcomer; consider the javadoc in Sun's Java2 SDK 1.4.1 for the interface `java.security.PublicKey`. While the class is easily found through the package and class lists when looking for a class representing cryptographical public keys, the javadoc is not terribly useful: it lacks the information on how to create objects that implement this interface. The only relevant links are on an accompanying `Use` page, without any context: the methods for creating a `PublicKey` are listed alongside the accessor methods that return a `PublicKey` contained in another, unrelated object.

Far more helpful would be a link to the design documentation, even just a diagram showing the intended uses or lifespans of related objects and how one creates the objects. These architectural diagrams are certain to *already exist* for all properly managed medium-to-large scale software projects — the problem is getting them to where it matters without having to manually insert the links into the low-level documentation. In this article, we introduce Navidoc: a system that, given design documentation with marked-up UML diagrams, inserts the UML diagrams into the classes' javadocs to function as spatial menus.

2. UML DIAGRAMS AS SPATIAL MENUS

The Unified Modeling Language (UML) is the standard way to visually describe software architectures and constructs in diagrams [1]. It is a natural part of software design documentation because it functions as a common language for communication within a software development team. Each UML diagram is meant to expose a particular aspect of the design. Therefore a single program element often appears on several separate diagrams that together create a multicontext view for the element. Using the UML diagrams created for the design documentation as spatial menus to jump around the documents for the different program elements is attractive exactly because of this: each diagram in which an element appears gives some of the necessary context for that element. Some screenshots illustrating this navigation paradigm are shown in Figure 1.

Navidoc, our implementation, is a light-weight tool built on top of existing Free Software tools. Navidoc converts each UML diagram into a set of imagemaps — one for each target document since the current document is highlighted in the images and inserts them into the relevant documents. The diagrams allow easy traversal through all the participant nodes, and also contain a backlink to the design document page containing the original diagram. The automatically inserted images are scaled to half size to avoid cluttering the Javadoc pages.

3. EXPERIENCES

We have found navidoc useful in our free software project family, the most important aspect being the value added to the Javadoc pages. Anecdotally, reaching the right documentation and under-

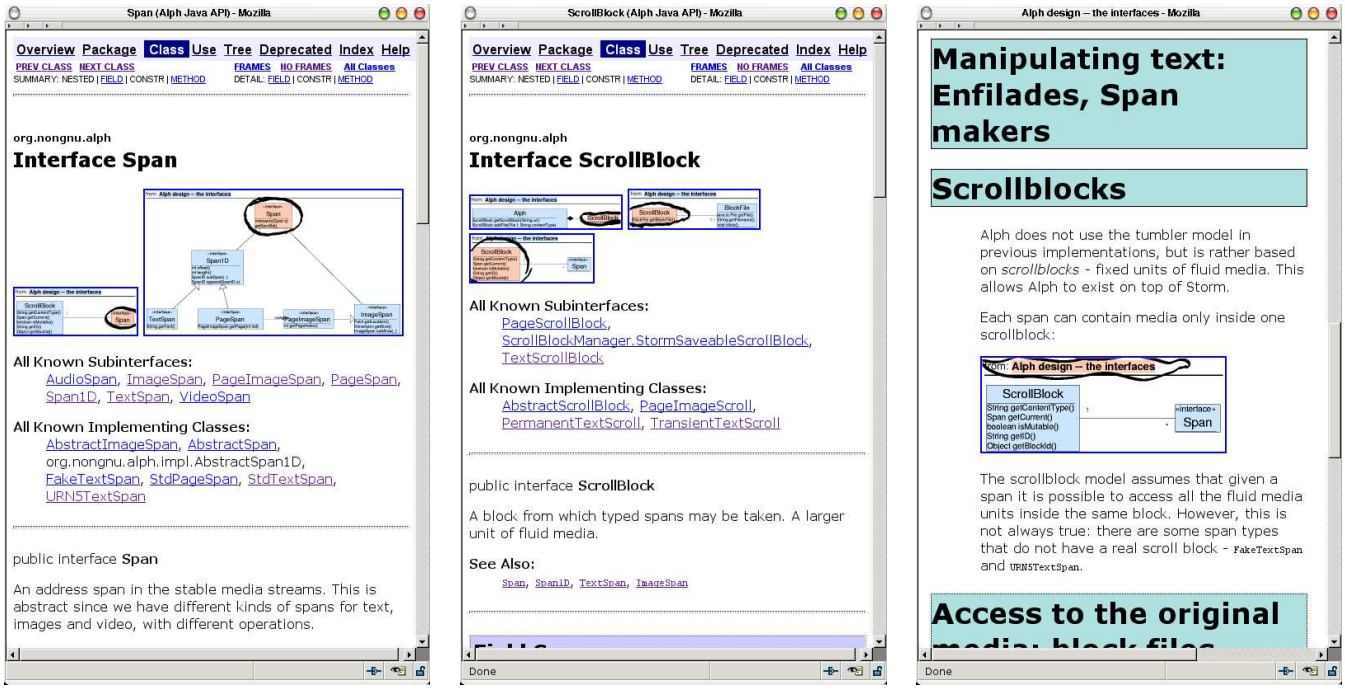

Figure 1: Three HTML pages modified by Navidoc. The left and center pages are Javadocs particular interfaces and the right-hand page is a part of the design documentation for those interfaces, which contains a UML diagram on which the two interfaces described by the Javadoc pages appear. Reduced copies of this diagram have been added to both Javadoc pages. The elements in the diagrams are active: clicking on the Scrollblock element in any of the diagrams takes the user to the Scrollblock javadoc. Traversing to the design document works by clicking the heading at the top of the diagram. In each version of the UML diagram, the currently active node is highlighted both with color and a distinctive thick, irregular line to catch the viewer's eye.

standing the context and interrelations between classes is easier.

Statistics from the WWW server of our project show that the UML diagrams have been used to traverse between javadoc and design documentation (approx. 10% of all page loads). Unfortunately, the referrer logs did not allow us to identify the use of UML diagrams for navigation within javadoc pages or within design documents, because there are other links besides the UML diagrams between those pages — this is a serious problem since anecdotally the UML diagrams are used to traverse between Javadoc pages more often than to design documentation: a common navigation technique appears to be to flick quickly between the Javadoc pages and then possibly read the deeper data in the design documentation.

Also, we are unable to say how much of the traffic came from testing and debugging Navidoc itself and how much from real use.

4. CONCLUSION

We have presented a navigational aid which hypertextually connects two distinct areas of documentation using human-authored UML diagrams as spatial menus. Now, it is not a new idea to use UML diagrams as spatial menus — for example, Doxygen [6] automatically generates class inheritance trees for navigation within the embedded documentation. Also, there has been work on connecting design documentation to code as well, e.g. through textual analysis [5].

However, automatically generated diagrams and indices often include too much irrelevant information; human-made UML diagrams are quite different from automatically generated ones since they try to *express* a meaningful part of the system's semantics. What is new in our approach is the use of the human-made UML diagrams from the design documentation as spatial menus on the relevant Javadoc or other embedded documentation pages, allowing traversal between the design documentation and the Javadoc pages as well as giving the user an idea of the context of the current class.

5. REFERENCES

[1] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language User Guide*. Addison-Wesley Object Technology Series. Addison-Wesley, 1998.

[2] P. T. Devanbu, Y.-F. Chen, E. R. Gansner, H. Muller, and J. Martin. CHIME: Customizable hyperlink insertion and maintenance engine for software engineering environments. In *ICSE*, pages 473–482, 1999.

[3] L. Friendly. The design of distributed hyperlinked programming documentation, 1995. Presented at the International Workshop on Hypermedia Design '95.

[4] O. C. Z. Gotel and A. C. W. Finkelstein. An analysis of the requirements traceability problem. *Proceedings of the First International Conference on Requirements Engineering*, pages 94–101, 1994.

[5] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *ICSE*, pages 125–137, 2003.

[6] D. van Heesch. Doxygen - a documentation system. http://www.doxygen.org/, 2003.