CogentHelp: A Tool for Authoring Dynamically Generated Help for Java GUIs

David E. Caldwell
Michael White
CoGenTex, Inc.
{ted,mike@cogentex.com}

Abstract

CogentHelp is a prototype tool for authoring dynamically generated on-line help for applications whose graphical user interfaces (GUIs) are built with the Java Abstract Windowing Toolkit (AWT). In this paper, we describe some of the techniques used in CogentHelp to facilitate the authoring, maintenance and customization of high-quality help systems. These include the use of (1) a "single-source" methodology for developing program code and help text; (2) small-grained, reusable "snippets" of help text instead of monolithic topics; and (3) a lightweight, extensible framework for planning and generating help topics from "snippets".

Introduction

We begin by discussing the design goals for CogentHelp, with reference to examples of previous work. We then give a brief overview of the CogentHelp system, and discuss some of the techniques used to support these goals.

Design Goals

In designing CogentHelp, we set out to achieve three main goals. The first goal was to effectively assist authors in creating high-quality on-line help documents. The second goal was to support the *maintenance* of help documents as the documented application evolves; this goal is closely related to the first, of course, since the quality of the help documents would diminish if they became inconsistent, either internally or with respect to the application. The final goal was to facilitate use of the technology deployed to achieve the first two goals.

Quality in help documents was taken to include the following aspects:

- *Consistency* — the grouping of material into help pages, the use of formatting devices such as headings, bullets, and graphics, and the general writing style should be consistent throughout the help system;
- *Navigability* — the use of grouping and formatting should make it easier to find information about a particular GUI component in the help system;
- *Completeness* — all GUI components should be documented;
- *Relevance* — information should be limited to that which is likely to be of current relevance, given the current GUI state;
- *Conciseness* — redundancy should be avoided;
- *Coherence* — information about GUI components should be presented in a logical and contextually appropriate fashion.

Secondly, effective maintenance of the quality of help documents was taken to imply:

- *Fidelity* — the help author should be assisted in producing complete and up-to-date descriptions of GUI components;
- *Reuse* — wherever possible, the help author should not have to write the same text twice.

Finally, we assumed that the benefits of the system must be made available at a reasonable cost in terms of the understanding and effort required of the help author and the developer, so that they could exercise their creativity in order to provide customized, appropriate help solutions.

SIGDOC *97 Snowbird Utah USA*

Related Work

The main idea of CogentHelp is to have developers or help authors write the reference-oriented part of an application's help system[1] in small pieces (or "snippets"), indexed to the GUI components themselves, instead of in a relatively unstructured document or set of documents. CogentHelp then dynamically assembles the snippets into a set of well-structured help pages for the end user to browse.

This general approach (storing many small document components in a structured database, and using them to generate complex and/or varied documents) has two main advantages over a "manual" approach to document creation. First, it makes it easier to maintain documents, since the dependencies between the snippets and the objects they describe are represented more explicitly, and consistency only needs to be checked "locally". Secondly, having a document generator frees the author to concentrate on writing accurate content for the snippets, rather than the drudgery of applying consistent formatting and managing a complex hypertext network.

Elements of this approach have been used in many systems for software-related documentation in the *literate programming* tradition (Knuth, 1992), such as the *javadoc* utility distributed with Sun Microsystems' Java Developers Kit (Friendly, 1995), and several systems presented at the last SIGDOC conference (Priestly et al., 1996; Carlton & Harmsen 1996; Roposh & Schoenrock, 1996; Korgen, 1996). The approach also draws on work in the natural language generation community (for example, Rambow & Korelsky, 1992; Reiter & Mellish, 1992; Johnson & Erdem, 1995; Knott et al., 1996; Milosavljevic et al., 1996; Paris & Vander Linden, 1996).

[1] By *the reference-oriented part* of a help system, we mean the part in which the user can navigate according to the structure of the GUI to view descriptions of individual windows, widgets, etc. The remainder of the help system will usually describe more generally what the application does, how to accomplish specific tasks, etc. CogentHelp does not specifically support creation of this part of the help system, though the architecture and *Exemplars* framework (see "Generating Topics with Exemplars") does facilitate the creation of application-specific, dynamically generated context-sensitive help here too.

There are currently several commercial development and help-authoring tools that embody one aspect of this approach by generating initial, "skeleton" help systems automatically, based on a scan of the existing GUI components in an application. And at least one such tool[2] is finally addressing the maintenance issue by performing consistency checks to make sure that each GUI component always has a corresponding help topic, and vice versa. However, to the best of our knowledge, CogentHelp is the first tool to combine the advantages of automatic consistency checking and automatic document generation in a tool for authoring documentation directed at end users.

System Overview

CogentHelp takes as input a set of human-written "help snippets" indexed to components of a GUI, and generates help pages describing components or groups of components. The generated help system incorporates various navigation aids, including an expandable table of contents tree and an automatically generated "thumbnail sketch" hypergraphic representing each GUI window. A sample help window is shown in Fig. 1.

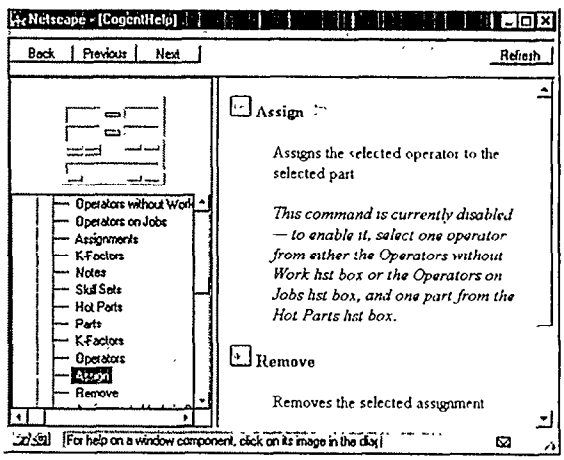

Figure 1: *A help window generated by CogentHelp*

CogentHelp's help-page generator is implemented using Java and HTML for platform independence, and with a client-server architecture in order to allow dynamic generation of help pages which reflect the current state of a GUI. Help topics are delivered by an HTTP server via the Java Servlet API, for display

[2] VBHelp, from ForeFront, Inc.

in a Web browser. The expandable table of contents and thumbnail hypergraphics are also implemented in Java, as applets hosted by the browser.

When a context-sensitive help request is initiated in a CogentHelp-enabled GUI, the Web browser is told to load a URL which points to the help server. This URL encodes most of the information needed to dynamically generate a topic describing the current window, in its current state — such as the locations, types and part-whole relations of the widgets on the window, and their current labels and states of enablement.

The other information that the help server needs in order to generate a topic is the help snippets themselves. These are stored separately from the GUI code, and consistency is maintained through the use of utilities which check for missing or surplus snippets.[3] The number and type of snippets created for each widget can be customized, but the default implementation has the following for each widget:

- A short description of the widget;
- An optional longer description of the widget's function;
- A list of references to other related help topics;
- Descriptions of the conditions under which the widget is disabled, and how to enable it.

The CogentHelp server functions not only to generate help topics for end users to view, but also to generate HTML forms which constitute the help author's interface to the system. The forms generated in authoring mode (see Fig. 2) generally resemble the generated topics as closely as possible, except that the help snippets are editable, and all snippets for a given widget are shown (whereas in generated topics, some of them will not occur in certain situations). The forms can also furnish syntactic "frames" which help authors to write snippets in a consistent format, in order to create opportunities for text reuse (the author might want to present the same text in multiple contexts, or simply modify a single context without having to rewrite all of the snippets that go into it). For example, in the default implementation the Short Description for a widget should fit grammatically in the frame "*This widget _____.*"— therefore, an appropriate snippet might be "opens the Assignments window".

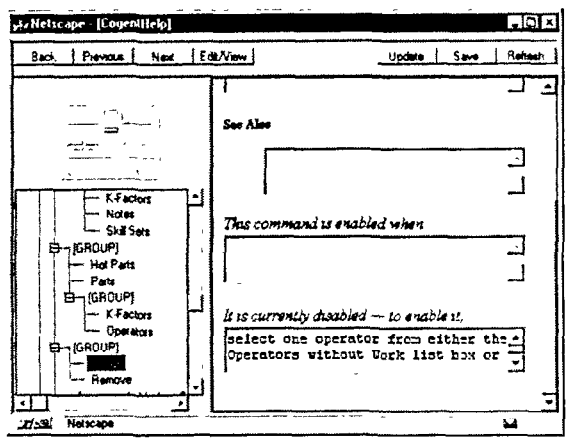

Figure 2: *CogentHelp in authoring mode*

The steps required to enable a Java GUI to use the CogentHelp framework involve both the help author and the developer. The developer "drops in" a Java package which handles communications with the Web browser and the help server, and implements a help button (or similar device) on each GUI window. Some code must also be added to specify which widgets on each window will be documented in help topics — this amounts to about one line of code per widget, and the effort involved is comparable to that of assigning unique help IDs to widgets in conventional help systems.

The help author, meanwhile, decides on the number and form of the help snippets to be assigned to each widget, and customizes the help exemplars (see the next section), if desired, to produce the appropriate structure and layout of the help system. The author may also collaborate with the developer to customize the information that is passed from the GUI and used in generating topics which reflect the dynamic state of the GUI — for example, giving messages about widgets which are disabled, or whose labels change in certain situations.

Generating Topics with Exemplars

To generate help documents, CogentHelp uses *Exemplars*, a lightweight Java-based text planning framework developed in connection with a variety of projects at CoGenTex (see also Lavoie et al. 1997; White & Caldwell 1997). Exemplars are designed to flexibly support a range of different text generation methodologies — from designs incorporating deep linguistic models via CoGenTex's RealPro syntactic

[3] As long as we provide such tools, we have found a "virtual single-source" approach to be adequate for storing snippets — it is often not convenient, or even possible, to physically store them with program code or resource files.

realizer (Lavoie & Rambow 1997), to approaches closer to traditional template-based document generation.

Exemplars are so called because they are meant to capture an *exemplary* (or expert) way of achieving a communicative goal in a given communicative context; as such, exemplars are more like mail-merge templates (or CGI scripts) than rules for first-principles planning. What distinguishes exemplars from templates is that they are *object-oriented* and recursive.

Object orientation in this case chiefly means that exemplars are arranged in a specialization hierarchy, where more specialized exemplars can inherit properties and methods from more general ones. For example, the help author might create an exemplar called *DescribePopupMenu*, which inherits from another called *DescribeMenu*, and which generates the same help text as *DescribeMenu*, but adds some material only relevant to pop-up menus (or, on the other hand, it could ignore its parent and generate its own text from scratch). The choice of which exemplar to use to describe a given widget is determined by constraints that the author attaches to each exemplar — in this case, *DescribeMenu* would be specified to apply to menus, and *DescribePopupMenu* only to pop-up menus (assuming that any pop-up menu is also a menu). As part of the help topic generation process, the text planner will automatically select the most specific exemplar whose constraints are satisfied by a given input.

Exemplars are recursive in that they can be embedded within one another — unlike templates, which are typically "flat". For example, a help author could create a *MakeStandardTopic* exemplar that builds a shell containing the features common to all help topics, such as background color, standard hypertext links, etc. Some of the holes in this shell, such as the name of the widget being described in the topic, could be filled in in the usual way, by simply substituting the value of a variable. But others could be filled in by calling other exemplars — say, one called *DescribeWidget* which generated a paragraph describing a widget. By itself, the recursive quality of exemplars allows for a more modular and compact specification of a help system, since the decisions made by *DescribeWidget* about how to generate a particular paragraph could be isolated from other decisions, and *DescribeWidget* could be reused by other exemplars. But recursivity provides the greatest benefit when it interacts with the specialization hierarchy — if, say, *DescribeWidget* had subclasses *DescribeRadioButtonGroup* and *DescribeTextField*, which generated appropriate descriptions of two different types of widget. In this case, when *DescribeWidget* was called to fill in a hole in *MakeStandardTopic*, the text planner would select whichever of these more-specific exemplars was appropriate for the given widget. By contrast, with flat templates the author might be forced to create a *MakeRadioButtonTopic* template and a *MakeTextFieldTopic* template, which would each specify the standard topic features redundantly, making it more difficult to modify them consistently later, should the need arise.

The Exemplars framework is implemented in Java, in order both to exploit aspects of Java's object orientation, and to facilitate customization by developers and help authors. Each exemplar is actually a Java class, which inherits from the base *Exemplar* class and overrides various members as appropriate, in order to specify:

- the input to the exemplar (typically a reference to a *Widget* object with a type, a label, etc.);
- the conditions in which the exemplar can be used;
- the output text (HTML) of the exemplar (possibly built recursively by calls to other exemplars).

The inheritance structure of exemplars is simply that given by the corresponding Java classes.[4] The output of an exemplar is built up as a tree representing an HTML document, using an API designed to make this not much harder in practice (and easier, in some respects) than editing raw HTML.

Customizing exemplars is easier in Java than it would be in some other languages, mainly because of Java's simpler compiling and linking procedures. And we believe that writing Java, at least within a constrained framework, will not daunt today's help author, who has lived to tell of RTF, HTML, JavaScript, Perl, etc. We have also developed a tool which minimizes the help author's exposure to Java code — the Exemplar Definition Wizard lets the author create a "starter" HTML file using the visual editor of his/her choice, then graphically specify the attributes of a new exemplar based on this HTML.

[4] The inheritance structure of a set of exemplars is not to be confused with another hierarchical structure, a *goal structure*, which is what is built up when exemplars call each other recursively in order to generate a specific topic.

The bulk of the Java code is then generated automatically, with comments indicating what details the author needs to fill in.

Customizing the exemplars typically means creating a new set of exemplar classes to generate a desired help system (or simply modifying an existing set, such as the default exemplars provided with CogentHelp). But authors, in collaboration with developers, can also customize more "deeply" by changing the nature of the information passed from an application to the help-topic generator. For example, one might want to generate messages which explain to the user not just why a button is disabled, but why the database they are browsing is in a certain state, with reference to underlying business rules. This type of customization is also facilitated by the fact that exemplars are written in the same language as the application they are documenting, and therefore can access any information in the application, in principle.[5]

Conclusion

In summary, each of the various ideas we have assembled in our approach to authoring and generating help contributes to the quality and maintainability of the finished product, in the aspects mentioned earlier. First, viewing the help system as "one topic per widget", plus a structure to be imposed on these topics, allows the author to focus on the accuracy of individual topics (*fidelity*) and the global presentation format of the help system (*consistency*) independently, without one of these interfering with the other. The utilities which monitor the correspondence of widgets to topics help to ensure *completeness*.

Secondly, the further step of assigning a set of help snippets, rather than just a block of text, to each widget gives the author the flexibility to tailor different help messages appropriate to different contexts, without necessarily writing each one from scratch (*conciseness*, *reuse*). Designing a set of exemplars also gives the author a framework for thinking about what help information should be presented where (*coherence*).

Finally, the ability to use runtime information in generating dynamic help topics can enhance their *relevance*, and the automatically generated thumbnails and table of contents improve the *navigability* of a help system, while relieving the help author of some exacting chores.

Future Plans

CogentHelp in its current form incorporates several features developed as a result of our work with a trial user group at Raytheon — especially in the area of authoring support and visual navigation aids. By the time of the conference, which coincides with the end of our Rome Laboratory–sponsored software documentation SBIR project, we hope to have completed the transition of CogentHelp from a research prototype to a user-friendly, configurable system suitable for use by a wider range of Java GUI development teams. Part of this transition will involve simplifying the process of customizing the exemplars even further — in particular, we plan to create an exemplar definition language, from which the exemplar classes will be generated automatically. As with the current Definition Wizard, this language will hide much of the complexity of implementing exemplars as Java classes, while additionally allowing the author to freely mix annotated HTML with Java expressions and statements for information access, conditionalization, loops, etc.

Acknowledgements

We gratefully acknowledge the helpful comments and advice of Ehud Reiter, Philip Resnik, Keith Vander Linden, Terri SooHoo, Marsha Nolan, Doug White, Colin Scott, Owen Rambow, Tanya Korelsky, Benoit Lavoie and Daryl McCullough. This work has been supported by SBIR award F30602-94-C-0124 from Rome Laboratory (USAF) and by the TRP/ROAD cooperative agreement F30602-95-2-0005 with the sponsorship of DARPA and Rome Laboratory.

References

1. Carlton, D. & Harmsen, M. (1996). Customizing tools to manage complex online help development. In *Proceedings of the 14th Annual International Conference on Computer Documentation (SIGDOC-96)*, Research Triangle Park, North Carolina, 29-34.

2. Friendly, L. (1995). The design of distributed hyperlinked programming documentation. In *International Workshop on Hypermedia Design.*

[5] This might call for another kind of architecture which we have experimented with, where the CogentHelp server actually runs in a separate thread within the application it is documenting, giving it direct access to any desired runtime information. We have so far only implemented this with Java applications, not applets.

3. Johnson, W.L. & Erdem, A. (1995). Interactive explanation of software systems. In *Proceedings of the Tenth Knowledge-Based Software Engineering Conference (KBSE-95),* 155-164.

4. Knott, A., Mellish, C., Oberlander, J. & O'Donnell, M. (1996). Sources of flexibility in dynamic hypertext generation. In *Proceedings of the Eighth International Natural Language Generation Workshop (INLG-96),* 151-160.

5. Knuth, D. E., editor (1992). *Literate Programming.* CSLI.

6. Korgen, S. (1996). Object-oriented, single-source, on-line documents that update themselves. In *Proceedings of the 14th Annual International Conference on Computer Documentation (SIGDOC-96)*, Research Triangle Park, North Carolina, 229-237.

7. Lavoie, B. & Rambow, O. (1997). A fast and portable realizer for text generation systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing,* Washington, D.C., 265-268.

8. Lavoie, B., Rambow, O. & Reiter, E. (1997). Customizable descriptions of object-oriented models. In *Proceedings of the Fifth Conference on Applied Natural Language Processing,* Washington, D.C., 253-256.

9. Milosavljevic, M., Tulloch, A. & Dale, R. (1996). Text generation in a dynamic hypertext environment. In *Proceedings of the 19th Australasian Computer Science Conference,* Melbourne, Australia, 229-238.

10. Paris, C. & Vander Linden, K. (1996). Drafter: An interactive support tool for writing. *IEEE Computer, Special Issue on Interactive Natural Language Processing,* July.

11. Priestly, M., Chamberland, L. & Jones, J. (1996). Rethinking the reference manual: Using database technology on the www to provide complete, high-volume reference information without overwhelming your readers. In *Proceedings of the 14th Annual International Conference on Computer Documentation (SIGDOC-96)*, Research Triangle Park, North Carolina, 23-28.

12. Rambow, O. & Korelsky, T. (1992). Applied text generation. In *Third Conference on Applied Natural Language Processing,* Trento, Italy, 40-47.

13. Reiter, E. & Mellish, C. (1992). Using classification to generate text. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics,* Newark, Delaware, 265-272.

14. Roposh, C. & Schoenrock, H. (1996). Developing single-source documentation for multiple formats. In *Proceedings of the 14th Annual International Conference on Computer Documentation (SIGDOC-96)*, Research Triangle Park, North Carolina, 205-212.

15. White, M. & Caldwell, D.E. (1997). CogentHelp: NLG meets SE in a tool for authoring dynamically generated on-line help. In *Proceedings of the Fifth Conference on Applied Natural Language Processing,* Washington, D.C., 257-264.