Nonvisual Tool for Navigating Hierarchical Structures

Ann C. Smith
Department of Computer Science
Saint Mary's University
Winona, MN 55987
asmith@smumn.edu

Joan M. Francioni
Department of Computer Science
Winona State University
Winona, MN 55987
jfrancioni@winona.edu

Mohd Anwar
Department of Computer Science
University of Minnesota - Morris
Morris, MN 56267
anwar@mrs.umn.edu

Justin S. Cook
Department of Computer Science
Winona State University
Winona, MN 55987
jscook2345@webmail.winona.edu

Asif Hossain
Department of Computer Science
Winona State University
Winona, MN 55987
ahossain5127@webmail.winona.edu

Mohammed Rahman
Department of Computer Science
Winona State University
Winona, MN 55987
mfrahman7664@webmail.winona.edu

ABSTRACT
The hierarchical structure of a program can be quite complex. As such, many Integrated Development Environments (IDEs) provide graphical representations of program structure at different levels of abstraction. Such representations are not very accessible to non-sighted programmers, as screen readers are not able to portray the underlying hierarchical structure of the information. In this paper, we define a set of requirements for an accessible tree navigation strategy. An implementation of this strategy was developed as a plug-in to the Eclipse IDE and was tested by twelve student programmers. The evaluation of the tool shows the strategy to be an efficient and effective way for a non-sighted programmer to navigate hierarchical structures.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *auditory (non-speech) feedback, style guides.*

General Terms
Design, Experimentation, Human Factors.

Keywords
Navigation, hierarchical structures, Java programmers.

1. INTRODUCTION
The Computer Science Curriculum Accessibility Program (CSCAP) [5] was designed to develop strategies to remove or significantly reduce unnecessary barriers to learning computer science for students with visual disabilities. As part of this project, over the past three years we have worked with six students with visual disabilities, who are majoring in computer science, in developing both teaching methods and software tools to address their needs [10].

The students in CSCAP have been using different versions of *JavaSpeak* [8, 16] as their programming environment. The current version of *JavaSpeak* [5] is based on Eclipse [7] and is able to support the students writing their own programs quite well. However, as some of students in CSCAP started taking the Advanced Data Structures course, we began to see that navigating through code that they did not write was much harder for them than for the sighted students. Since time spent investigating existing source code is a significant portion of the time required to develop and maintain programs [14], this was recognized as a serious issue for the non-sighted students.

To deal with the problem of managing large code in general, whether it is the programmer's own code or someone else's, many current integrated development environments (IDEs) include graphical tools for presenting program structure at different levels of abstraction. Unfortunately, these strategies are not very accessible to blind and low-vision programmers who are dependent on screen readers. Although screen readers may be able to read the labels of graphical components, they are not inherently able to make the functionality of the hierarchical views available to non-sighted users.

This paper reports on the development of a tool that supports nonvisual navigation, via keyboard input and speech/sound output, of hierarchical representations (or "trees") of information. Specifically, the tool works as a plug-in to the Eclipse IDE and represents program structure information. However, the concepts are applicable to any tree structure, such as file systems, that a blind user needs to navigate.

For this research, we performed a usability study using hyperbolic [10] browsers to identify specific ways that sighted users move through fully expanded, unfamiliar trees. "Hyperbolic browsers" are browsers that employ a fisheye technique, whereby the user can see details of specific parts of the hierarchy at the same time as viewing the general content of the full hierarchy. In addition,

ASSETS'04, October 18–20, 2004, Atlanta, Georgia, USA.

the part of the hierarchy that is in focus can be changed simply by moving the mouse.

Based on our usability study of the hyperbolic browser and prior research, we defined a set of requirements for an accessible tree navigation strategy. A prototype was implemented and evaluated with two of the blind students in the program. A plug-in for Eclipse was then implemented and tested via a more extensive usability study. The following sections explain this process in more detail and present specific recommendations for a nonvisual tool to navigate tree structures.

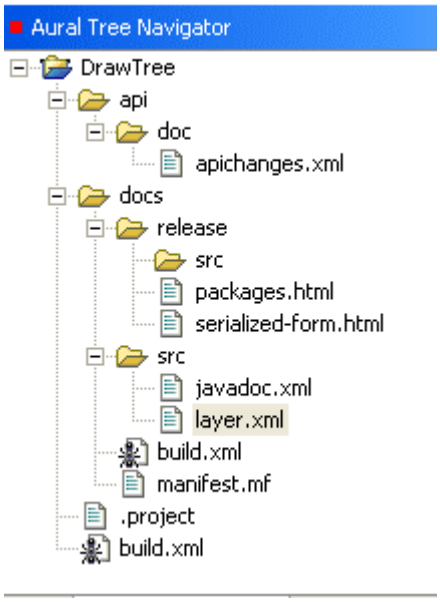

Figure 1. Eclipse View of Program Structure

2. PRIOR RESEARCH

Screen readers, which have been available for over twenty years, are computer applications that provide spoken feedback to the user. During the 1980's screen readers relied on the character representation of the content on the screen in order to produce a spoken translation of that content [12]. The advent of bitmap displays and GUIs led to a breakdown of this approach and spurred significant research attention to solve a new and more difficult problem – providing ways to translate GUIs to meaningful auditory user interfaces (AUIs).

AUIs have the capability of rendering both speech and non-speech feedback. Mynatt [11], Thatcher [18] and Raman [13] contributed seminal work to the area of speech access to GUIs. An early, formalized concept for translating visual icons to sound was the *earcon*. Earcons were defined to be "non-verbal audio messages that are used in the computer/user interface to provide information to the user about some computer object, operation or interaction" [2]. Brewster [3] experimented with using earcons specifically to provide navigational cues in a menu hierarchy. His results strongly support the idea that users can learn different earcons relatively quickly and then use them to navigate within a hierarchical structure. The concept of earcons has since been extended with the concept of 3-dimensional sound objects called *hearcons* [6].

IDEs have allowed for the integration of information hierarchies such as class libraries, source code repositories, and data dependency graphs. For even medium-size programs, these hierarchies can become relatively complex. Human cognition, however, can quickly become challenged by this complexity. For sighted users, research in the area of information visualization is providing visual aids for navigating and comprehending these complex hierarchies [17]. A complementary approach to solving this problem is to present different levels of informational abstraction to the user. The phrase *focus + context* has been used to describe the idea of displaying most information compactly with lesser detail (context part), but some information with greater detail (focus part) [10].

Donker, Klante and Gorny [6] list four general barriers that cause difficulties for blind users when interacting with graphical user interfaces: the pixel barrier, the mouse barrier, the graphics barrier, and the layout barrier. The one relevant to this paper is the layout barrier, which occurs when semantics that are encoded in the layout of graphical interfaces are not relayed during screen reader translation.

3. SYSTEM REQUIREMENTS

3.1 Tree Navigation Strategy for Non-Sighted Users

At the most basic level, an accessible tree navigation strategy for non-sighted users must support all navigation tasks with only keystroke commands and be based only on sound and speech output. At a more desirable level, however, "good" accessibility means the users have appropriate commands and output that assist them in parsing and navigating a tree effectively and efficiently.

Limitations of Existing Screen Readers.
Commercial screen readers have improved tremendously over the past five years in providing effective and efficient GUI navigation strategies for non-sighted users. Their main limitation, however, is that "a screen reader speaks *what* is on the screen without conveying *why* it is there [13]." In the case of hierarchical structures, a node's location in the tree is important in the context of its parent and sibling nodes, not in the context of which other nodes are drawn nearby.

For example, JAWS [9] provides information about the level and number of siblings of a node, which is good. However, the navigation strategy is inconsistent with moving *logically* through a tree structure. This is because it follows the picture of the tree, not the logic of the tree hierarchy. Looking at the tree in Figure 1, if the current position of the cursor is on layer.xml and the user hits an up-arrow, the cursor will move to the sibling node javadoc.xml. Issuing the same up-arrow command again from that point will have a very different logical meaning, as it will move the cursor to the parent node src. This functionality of the up-arrow only makes sense if you can see the picture. Additionally, when the picture changes, the functionality of a command may also change. Given the tree expanded as shown, if the cursor is still on the same src node and the user hits the up-arrow, the cursor will move to serialized-form.html, which is a child of one of src's siblings. However, if the release node was not expanded, the same command would move the cursor from src to it's sibling release. Again, the functionality is matching the picture, not the logic that the picture is conveying.

A tool like Emacspeak [12] provides a fully functioning audio desktop specifically designed for non-sighted users. However, it does not provide a mechanism for navigating a tree structure within a specific IDE, such as Eclipse. We are particularly interested in a strategy that is portable to IDEs in general.

General Criteria for Accessible Tree Navigation Strategy.
Specifically, we define the following general criteria for a tree navigation strategy (TNS) to be considered accessible.

- The TNS must be accessible to non-sighted users.
- The TNS must match the hierarchical structure of the tree.
- The TNS must support navigating large, complex trees representing program structure. While any particular program structure tree may be relatively small, the navigation strategy must not be limited to the kinds of hierarchies represented by most menu systems.
- The TNS must support effective navigation through unknown tree structures as well as familiar trees.

Meeting these criteria alone, however, does not ensure that the user interface is effective and efficient. Independent of navigating a tree structure, an effective and efficient interface is one that

- is consistent, easy to learn, and easy to remember,
- leads to few errors, and
- the user feels provides useful interactions [1].

In Section 5, we will evaluate a specific accessible tree navigation strategy tool in terms of these seven criteria.

3.2 Functional Requirements of an Accessible Tree Navigation Tool

Usability Study of Hyperbolic Browser.
As part of a graduate class project [15], a formal usability study was performed on Inxight Corporation's Star-Tree 1.1™ hyperbolic viewer [19]. Although the focus of the study for the class was on how well sighted users liked the hyperbolic browser compared to more traditional browsers, the study was designed to also elicit information about the functionality users needed to navigate through a tree structure. In particular, the hyperbolic viewer presents trees as fully expanded at all times. The observations made during this study were used to define an initial set of functional requirements that were consistent with the accessible tree navigation strategy criteria. The requirements included navigation commands and speech and sound output.

Prototype of Accessible Tree Navigation Program.
A stand-alone program was written as a prototype for the functional requirements described above. The program used the CloudGarden [4] implementation of the Java Speech API for the speech output. A small tree of 14 nodes over 4 levels was hard-coded into the program for testing purposes.

Two students in the CSCAP project volunteered to test the prototype. Both students are completely blind, and both are experienced JAWS for Windows users. One of the students had taken Computer Science courses through Advanced Data Structures (the third programming class in the major), and the other one had taken through the second programming class. They were both familiar with the concepts of stacks and trees, and both had experience navigating tree structures, such as Explorer, in a Windows environment.

As an introduction to the test, the students were told what the test was generally about and then they were given information about each of the keyboard commands. They were then asked to use the commands to explore the test tree. No information was given to the students up-front about the output of the program that they would be hearing.

After the students said they were comfortable with the tree navigation commands, they were asked a set of questions that required them to move through the tree to be able to answer. Following the task questions, the students were then asked a set of general questions about the ease of use and usefulness of the program's output. The test concluded with a general discussion session where the students offered their opinions and suggestions of things to change or add to the program.

Functional Requirements of an Accessible Tree Navigation Tool.
Based on the hyperbolic browser usability study and the evaluation of the prototype described above, ten functional requirements for an accessible tree navigation tool were defined as follows:

1) *All nodes of the tree are considered to be "visible" from the start.*

2) *The spatial orientation of the tree is sideways, with the children of a node to the right of the node, the parent to the left, and the siblings above and below the node.*

   To establish directions that made sense for moving in the tree, it was necessary to define a specific orientation of the tree. Consistent with the presentation of hierarchies in Eclipse, a sideways orientation was chosen.

3) *An "error" sound is generated if a user tries to make an illegal move in the tree.*
   An example of an illegal move would be to try to move to the parent of the root of the tree.

4) *The list of siblings is presented as a circular list.*
   For example, moving to the "next" sibling from the last one in the list will move the cursor to the first sibling of the list.

5) *Commands are supported for moving directly to the nearest cousin nodes.*
   We observed that users frequently visited neighboring cousin nodes without going "up" to a grandparent, then back down to the cousins. They explained they did this when they felt like they were in the right "area" of the tree to find what they were looking for.

6) *Information about relative size of the sub-tree rooted at the current node is available.*
   The size of a sub-tree rooted at a particular node is considered to be the total number of descendants of that node.

7) *A "where am I" command is supported to give users information about the current node.*
   The "current node" is defined to be the node where the cursor is positioned.

8) *A command is supported to give information about the distance of the current node from the root of the tree.*

9) *It is possible to know if a node has previously been "visited." It is also possible to clear all visited marks at any given time.*
A node is marked as visited after all of its children have been the "current node" at least once. In this sense, the node has actively been in focus and its list of children has been identified.

10) *"Anchor points" are supported, whereby the position of the cursor can be saved to a stack, to be returned to at a later time.*

4. IMPLEMENTATION OF AURAL TREE NAVIGATOR

A specific implementation of an accessible tree navigation strategy has been written as an Eclipse plug-in. The implementation presents a customized view in Eclipse, called Aural Tree Navigator, that meets all the requirements defined in Section 3.2. As in the prototype tool, CloudGarden's JSAPI implementation was used for the speech output. Using this implementation, a formal usability study was done to test the effectiveness and efficiency of the tool. Information about this implementation and the usability study are given in this section.

4.1 Aural Tree Navigator Commands

Table 1 shows the user commands and their effect in the Aural Tree Navigator.

Table 1. Aural Tree Navigator Commands

Command	Action and Response
arrow keys	Move cursor to new node. Speak node's name and which sibling it is (e.g. "src, sibling 2 of 4").
left-arrow right-arrow	- from node to its parent - from node to one of its children; will move to most recent child visited.
up-arrow down-arrow	- from node to previous sibling - from node to next sibling (circular list of siblings assumed)
cntl-arrow keys	Move cursor to nearest cousin. Speak node's and node's parent's names (e.g., "release; parent docs").
cntl-up-arrow cntl-down-arrow	- move from node to previous cousin - move from node to next cousin
(subtree size)	Geiger Counter sound output. A continuous, background clicking sound, whose frequency is relatively high when current node has many descendants and correspondingly low when current node has with few descendants, is used.
i	Where-am-I? Speaks current node's name and number of children (e.g., "current node release; children 3; parent docs").
r	Distance to root? Speaks number of hops to root (e.g., "two hops to the root").
visited commands	Node visited?
v	Speaks whether node has been visited (e.g. "release has not been visited").
cntl-v	Marks all nodes as unvisited and speaks "all nodes unvisited".
anchor points	Set or jump to anchor point.
p	Push current node onto anchor stack; speak node affected and action (e.g., "an anchor point has been set at src").
cntl-p	Pop node from anchor stack and move to that node; speak action (e.g. "moved to anchor point src").
s	Stops current sound/speech output, including Geiger counter sound.

4.2 Usability Study

A usability study of the Aural Tree Navigator tool was administered to twelve participants. Two of the participants are blind; the ten sighted participants took the tests without being able to see the screen. All of the participants are computer science students and all signed consent forms to take the test. (They were thanked with cookies.)

As the participants all had Eclipse on their laptops, they were given a plug-in for the Aural Tree Navigator and shown how to install it on their machines. They were then given instructions about how to use the tool and allowed to practice with it overnight. For this practice only, the sighted participants were allowed to look at the screen.

For the formal part of the study, the participants were asked to take three tests using three different trees. The first two tests were designed to study the usefulness of the commands in navigating the trees. The third test was designed to study how well the tool supported understanding of the underlying program structure. The participants were able to reference the list of commands as needed during the tests.

For Test 1, participants were given a program tree of medium complexity. The tree had seven nodes at the first level and a total of 203 nodes. The longest path of any one leaf node was 13. The average path length of leaf nodes was 4. The participants were asked questions regarding the existence of specific files, given their full path name; and regarding information about certain nodes, such as number of children, names of cousins, and size of subtree. Some of the path names of the files they looked for had common beginnings.

For Test 2, participants were given the program tree shown in Figure 1 and asked to draw this tree. The blind participants used Excel spreadsheets to draw their trees.

In Test 3, participants were given a very large tree representing the package structure of the NetBeans program. This program includes over a thousand nodes. The height of the tree is greater than 20. Although quite large, there is a logic to the names used for the program's package structure. The participants were asked to find six different nodes in this tree. The location of these nodes ranged from a depth of 3 to 6 in the tree. However, the names of the nodes gave hints as to their logical location in the tree. For example, participants were asked to find node openCookie.java, located at NetBeansTree\org\openIDE\cookies\openCookie.java. They were also asked to identify which features helped them if they found the node, or contributed to them not finding the node.

After all the tests, the participants were asked to write down their general comments and to comment specifically on the use of each of the commands and speech/sound output. All of the participants were questioned informally after the test. Four of the participants were also interviewed formally by the examiners.

5. OBSERVATIONS AND EVALUATION

Section 3.1 outlined four criteria for an accessible tree navigation strategy and three criteria for an effective and efficient interface. In this section we evaluate the Aural Tree Navigator in terms of both its navigation strategy and its interface.

1) Accessible to non-sighted users.

All participants were able to use the tool without sight. Some of the sighted participants did complain about the pronunciations of some of the nodes, but this was not a significant barrier to navigating the system.

2) Navigation strategy related to hierarchical structure of information.

By design, the navigation strategy allowed users to move through the tree consistent with the hierarchical relationships among the nodes. Significantly, the sighted participants reported that being able to look at the tree when they were practicing with the tool was not helpful. When it came time to use the tool without sight, they needed to rethink how to visualize the tree based on how they were arrowing through the nodes. There was general agreement that this did not take long, but that it had to be done without looking.

3) Supports navigating large and complex trees.

The participants were able to find the six nodes in Test 3 with 90% accuracy. For those times when the node was found, the most common reason given was "found a pattern," followed by "lucky guess." When asked about the lucky guesses during the interviews, however, three of the four participants explained that they had a good idea where to look based on where they had already been and what they were catching on to as far as the package structure of NetBeans.

Participants reported that the most useful commands during Test 3, other than the arrow keys, were the "where am I" and "visited" commands.

For those times when the node was not found, the contributing factors were listed as pronunciation, and getting lost.

4) Supports navigation through unknown and familiar trees.

All of the test trees were previously unknown to the participants. Test 2 was designed specifically to see if participants could form an accurate mental image of the tree they were navigating. In this test, two participants tried to draw the tree of Test 1 by mistake. They both quit, after a while, because it was such a large tree. But the parts they drew were correct. Of the other ten participants, six drew completely accurate trees, three had one mistake, and one had two mistakes (like leaving out one of the children).

For the sighted participants, most of them drew their trees sideways. Two participants, however, drew their trees top-down, similar to how trees are drawn in computer science classes. Nonetheless, the hierarchical information in these trees was correct.

5) Consistent, easy to learn, easy to remember.

The participants all reported that the navigation commands were easy to learn and remember. During the test, it was observed that the participants referred to their command-lists early in the test, but then used them very infrequently after about 20 minutes.

6) Leads to few errors.

Test 3 was designed to test how this tool could be used to support program understanding. Although some participants were not able to find all of the nodes during this test, there was substantial evidence that all of them were able to understand the package structure of the program to some extent. As mentioned earlier, participants frequently reported that they had caught on to some of the naming logic of the packages, and they used that information to help guide their search.

Participants also started using anchor points much more in this test than they had earlier. This shows that they were catching on to which subtree they needed to be searching, in order to find their node.

Participants did report getting lost sometimes in the NetBeans tree. Their only recourse at these times was to go back to the root and start over. In a number of cases, the participants gave up at this point for that particular search. Interestingly, the two blind participants found all of the nodes.

7) Provides useful interactions.

The most used commands were the arrow keys. Participants reported that the sideways orientation of the tree was very easy to get used to and the arrows made sense to them in this context. There was also much praise for the list of siblings being a circular list.

Participants seemed to take for granted that moving to a set of children already visited positioned the cursor at the most recent child visited. For example, if the current node is the second of five children, and the user moves to the parent and then back to the same list of children, the current node would be the second child again instead of the first one. When asked directly about this later, however, they said that this allowed them to more easily "get back to where I was before."

The two blind participants felt that the tree being fully expanded was a useful feature of the tool. Also, one sighted student remarked that this helped him to figure out the tree easier than having to open and close folders.

The ability to move directly to neighboring cousins was not reported to be useful by most participants. A couple of people, however, said this feature was very helpful in finding certain

nodes: “I didn’t use the cousin feature till after I explored a bit, but (then) it was fairly useful.”

The Geiger Counter density feedback got very mixed reviews. About half of the participants found it completely distracting and continually stopped it. A few others said sometimes it was helpful, but most of the time they didn’t use it. And two people mentioned that it definitely helped them in their searches and it never bothered them. The consensus seems to be that it would be better to have this as an optional feature, rather than a default one.

Participants reported frequent use of the “where am I” command. There was a request for a way to get the full path name of a node in addition to just the node name.

Only sporadic use of the “distance to the root” command was reported. Those that did use it found it useful though.

The ability to check if a node had already been visited was reported as very useful at times, particularly when doing exhaustive searches of parts of an unfamiliar tree. Some participants also reported using this feature when they had to draw the tree in the Test 2.

Anchor points were generally considered to be a useful feature by almost all the participants. Participants reported using them during Test 3 and also in Test 1 when they had to search for nodes with prefix path names common to nodes they had previously searched for. One participant, however, reported that “I didn’t like anchor points because I would rather backtrack, which helped me remember where I had been.”

8) Additional Recommendations.
Additional functionality that participants requested included having a shortcut key to move directly to the root of the tree, being able to move to the first sibling in a list starting with a certain letter (since the siblings are listed in alphabetical order by type), and being able to access the full path name of a node. Also, as mentioned earlier, it was recommended that users have the ability to turn the Geiger Counter off completely.

Participants also recommended including a “search” command that would find a node directly. Such a command would be consistent with the recommendations made in [1] to support efficient search strategies.

6. Conclusions and Future Work

The accessible tree navigation tool described in this paper provides an effective and efficient strategy for non-sighted users to navigate within hierarchical representations of information. The tool offers users an interface that is consistent with the logical structure of trees. As such, users are able to develop an accurate mental model of the tree and can therefore navigate successfully within both familiar and unfamiliar trees of various sizes.

The tool also provides the user with an appropriate *focus + context* tool. The “context” features let the user know where they are located in the tree, relative to the tree’s overall structure. These features include giving the user location information about distance from the root, who the cousins are, and what the density of the subtree is, as well as providing a means for the user to change the context relative to a new anchor point. The strategy also includes “focus” features that let the user get detailed information about a specific node, including the parent, the number of children and siblings, and the past history of visiting the node.

Future Work.
Future work will include extending the implementation to provide a mechanism for users to build their own trees. This will be particularly useful for non-sighted students in computer science who can then “draw” trees and manipulate them as a way to understand concepts such as binary search trees.

7. ACKNOWLEDGMENTS

We would like to thank SueAnn Rodriquez, Dennis Schwab, and the students in the Spring 2004 CS 341 class at WSU for participating in the usability study of the Aural Tree Navigator.

This work is supported, in part, by the National Science Foundation, under grant number 9986689.

8. REFERENCES

[1] Barnicle, K., “Usability Testing with Screen Reading Technology in a Windows Environment,” in proceedings of the 2000 *Conference on Universal Usability*, Arlington, VA, November 2000.

[2] Blattner, M., Sumikawa, D., & Greenberg, R., “Earcons and Icons: Their Structure and Common Design Principles,” *Human Computer Interaction,* Vol. 4 (1), pp. 11-44, 1989.

[3] Brewster, S. A., “Using Nonspeech Sounds to Provide Navigation Cues,” *ACM Transactions on Computer-Human Interaction (TOCHI)*, Vol. 5 (3), pp. 224-259, Septemnber 1998.

[4] CloudGarden's TalkingJava SDK with Java Speech API Implementation, http://www.cloudgarden.com/JSAPI/index.html.

[5] Computer Science Curriculum Accessibility Program, http://cs.winona.edu/CSCAP.

[6] Donker, H., Klante, P., & Gorny, P., “The Design of Auditory User Interfaces for Blind Users,” presented at NordiCHI, October, 2002, available at http://www-cg-hci.informatik.uni-oldenburg.de/resources/zib_nordichi.pdf .

[7] Eclipse Platform, http://www.eclipse.org.

[8] Francioni, J. M., & Smith, A. C., “Computer Science Accessibility for Students with Visual Disabilities,” proceedings of *33rd SIGCSE Technical Symposium on Computer Science Education*, Cincinnati, Kentucky, February 2002. Available at http://cs.winona.edu/cscap/papers/sigcse2002.pdf.

[9] JAWS for Windows, Freedom Scientific, http://www.freedomscientific.com/.

[10] Lamping, J., Rao, R., & Pirolli, P., “A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies,” in proceedings of the *SIGCHI Conference on Human Factors in Computing Systems*, Denver, CO., 1995.

[11] Mynatt, E. D., *Auditory Presentation of Graphical User Interfaces,* Addison-Wesley, Reading, MA., 1994.

[12] Raman, T. V., “Emacspeak-A Speech Interface,” in proceedings of the SIGCHI conference on *Human Factors in*

Computing Systems, Vancover, British Columbia, Canada, pp. 66-71, 1996.

[13] Raman, T. V., *Auditory User Interfaces: Toward the Speaking Computer*, Kluwer Academic Publishers, Boston, 1997.

[14] Robitaille, S., Schauer, R., & Keller, R. K., "Bridging Program Comprehension Tools by Design Navigation," presented at the *International Conference on Software Maintenance,* San Jose, CA, October 2000.

[15] Smith, Ann C., "Usability Study of Hyperbolic Browser Star Tree Viewer 1.1," report for DCIS 720 Course, Nova University, July 2003. Available at http://csdept.smumn.edu/asmith/hyperbolicUStudy.pdf .

[16] Smith, A. C., Francioni, J. M., & Matzek, S. D., "A Java Programming Tool for Students with Visual Disabilities," in proceedings of the *Fourth International ACM Conference on Assistive Technologies (ASSETS)*, Arlington, Virginia, 2000. Available at http://cs.winona.edu/cscap/papers/assets2000.pdf.

[17] Spence, R., *Information Visualization*: ACM Press Addison-Wesley, 2001.

[18] Thatcher, J., "Screen reader/2: Access to OS/2 and the Graphical User Interface," in proceedings of the *First Annunal ACM Conference on Assistive Technologies (ASSETS)*, pp. 39-47, 1994.

[19] Star-Tree 1.1, *Inxight Corporation*, h*ttp://www.inxight.com*.