Combinatorial Graph Creation and Navigation for Blind People

Suzanne Prem Balik

February 4, 2011

Abstract

Blind individuals have been deprived of the use of diagrams as a form of knowledge representation and an aid to problem solving as well as a means of communication. Our focus is on providing them with access to a specific type of diagram – the node-link diagram, also known as a combinatorial graph. Graphs figure prominently in computer science and software engineering as well as chemistry and other fields. Our goal is to provide blind students and professionals with graph representations that approach the computational equivalence of those available to sighted people. A number of applications have been developed specifically and exclusively to give blind users access to graphs, but separate applications are often not equal. We have worked instead to include them in our graph-based application known as ProofChecker which is intended for universal use.

In this paper, we provide examples of general diagram accessibility efforts on behalf of blind people, a discussion of important issues in conveying graphs nonvisually, and a review of the work to date in this area. We also describe how ProofChecker has been made accessible to blind users and how these techniques were applied to a commercial graph-based application. Lastly, we propose the development of an universally accessible graph sketching tool. This tool would allow blind and sighted people to work alone or together using the same interface to create and/or examine graphs and use them for design, problem solving, and teaching/learning.

1 Introduction

Diagrams present a rich source of knowledge and enhance problem solving [1], yet they are often inaccessible to blind people. A 2003 study commissioned by Microsoft Corporation found that 27% of working age adults[1] in the United States suffer from a visual difficulty or impairment. This includes 27.4 million individuals with a mild visual difficulty/impairment and 18.5 million whose visual difficulty/impairment is severe [2]. Addressing the needs of visually impaired computer users makes sense both economically and ethically and in some cases is required by law (See Table 1).

[1] 18 - 64 year olds.

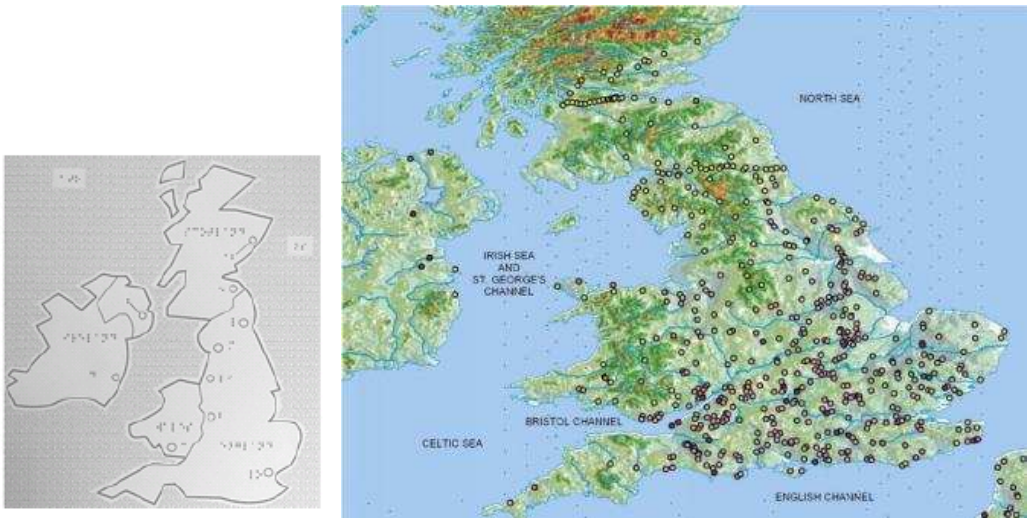

Figure 1: Tactile vs. BATS representation of Roman Britain [13]

Visual impairments that affect computer use include color blindness, low vision, and blindness. There is no clear dividing line between low vision and blindness. In the U.S., a person whose corrected vision is 20/200 or less is considered "legally blind".[2] In terms of computer use, Bergman and Johnson define a blind person as anyone "who does not use a visual display at all." [3]

Blind computer users generally rely on the keyboard for input and receive output via a screen reader in the form of synthesized speech or Braille sent to a refreshable Braille display. Commercial screen readers for the Windows platform include JAWS [4], Window-Eyes [5], and the ZoomText Magnifier/Reader [6], with JAWS being the most widely used. Open source screen readers for the Gnome environment include Orca [7] and the Linux Screen Reader (LSR) [8]. The VoiceOver screen reading technology is provided as part of Mac OS X [9]. Other input aids for the blind include the use of Braille or simulated Braille keyboards, predictive dictionaries, and speech recognition sofware provided by the operating system or a commercial product such as Dragon NaturallySpeaking [10]. Computer output may also be directed to a Braille printer.

1.1 Diagram Accessibility

Diagrams in general have typically been made accessible to blind people via *static tactile representations*. The Tactile Graphics Assistant (TGA) was developed to partially automate the long and arduous process of creating these representations [11, 12].

Combining *tactile input* with *dynamic audio output* provides the user with a much greater level of detail over traditional Braille-labeled tactile diagrams as illustated by the Blind Audio Tactile Mapping System (BATS) [13] shown in Figure 1. The BATS interface also

[2] A person with 20/200 vision can see at 20 feet what a person with normal vision can see at 200 feet.

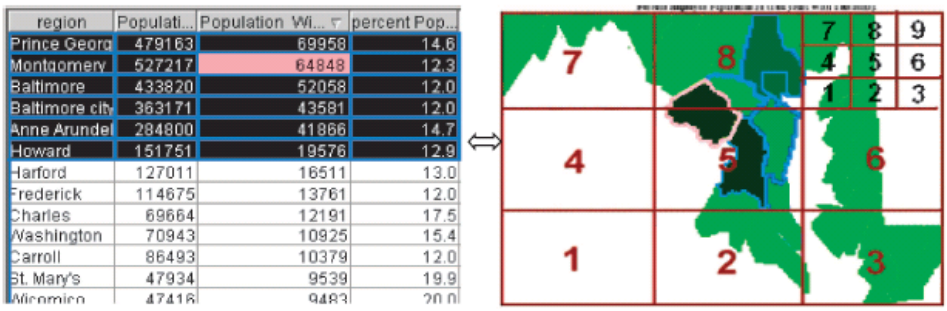

Figure 2: Population statistics for the counties of Maryland conveyed via the iSonic coordinated data table/map interface. [14]

incorporates *haptic feedback* in the form of bumps and textures that communicate geographic or political boundaries to the user when using a joystick or tactile trackball to examine a map. *Auditory icons* such as the sound of crashing waves, chirping birds, and traffic noise are used by BATS to signify oceans, forests, and cities respectively.

The iSonic tool was developed to help blind users explore coordinated choropleth maps[3] and has the advantage of not requiring any input devices beyond the standard keyboard and numeric keypad [14]. The iSonic interface uses a 3x3 grid[4] to recursively divide a map into regions thereby allowing the user to zoom in on a specific portion of the map (See Figure 2). The iSonic tool also uses *musical notes* in various pitches and timbres together with *stereo panning effects* to convey the data values associated with the regions of the map. Another application known as AudioGraph used *musical notes* to convey shapes and bar charts [17, 18, 19]. Music in the form of short musical phrases, know as *earcons* [20], was incorporated into the AudioGraph interface to denote various controls.

1.2 Accessible Graph-based Applications

A combinatorial graph, also known as a node-link diagram, is formally defined as G = (V,E), where V represents a set of vertices or nodes and E is the set of edges or links connecting pairs of vertices. This type of graph differs from graphs in the coordinate plane and is the type of graph to which this paper refers.

Graphs are important in many areas. Chemical molecules consist of atoms linked to other atoms with bonds. Software engineers use UML diagrams when designing computer applications. The study of computer science is replete with a variety of graphs such as entity-relationship diagrams, data structures, and automata, as well as graph-related algorithms such as breadth/depth first search and tree traversals. Real world applications sometimes include graphs as part of the interface. For example, business intelligence and predictive analytics software company SAS Institute uses process flow diagrams in three

[3] A choropleth map is one in which regions are colored or shaded based on the value of a statistical variable, for example, population or per-capita income.

[4] This 3x3 grid was inspired by the Integrated Communication 2 Draw (IC2D) system developed to allow blind users to create and examine drawings [15, 16].

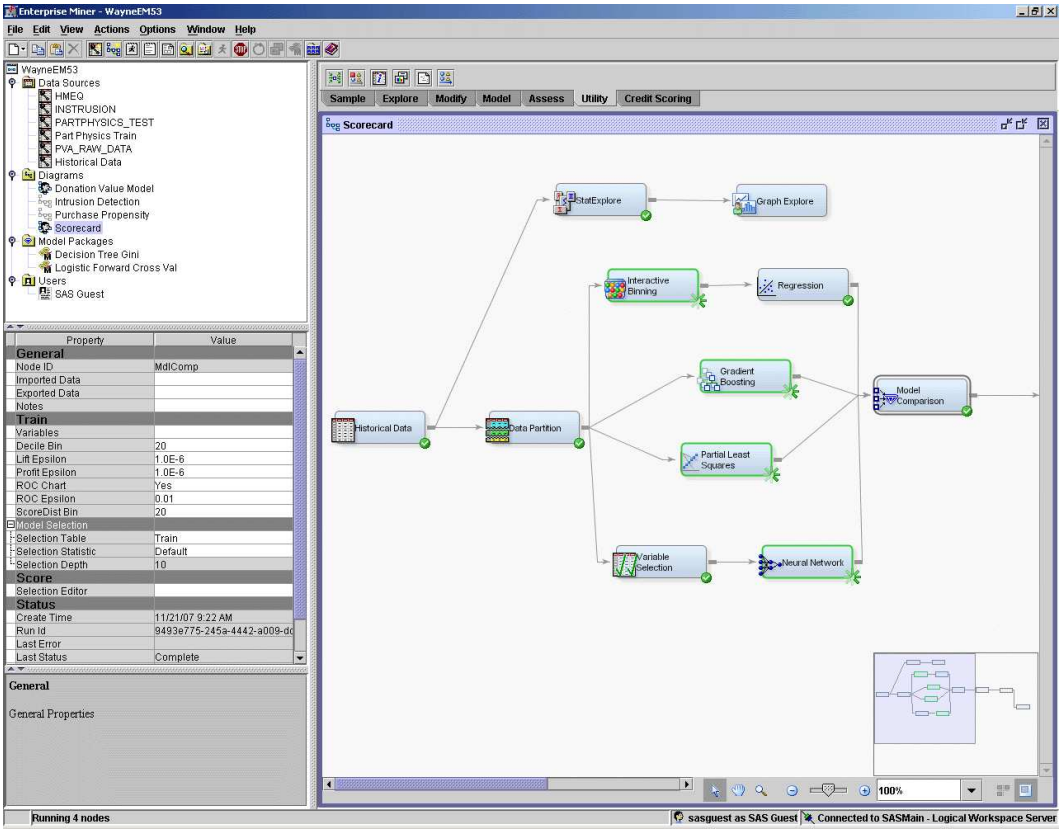

Figure 3: SAS® Enterprise Miner interface with process flow diagram. [21]

of its applications – Enterprise Guide, Data Integration Studio, and Enterprise Miner (See Figure 3). In order for blind people to function in these areas, they must be able to access and create graph structures.

1.2.1 Graph Accessibility Issues

In any graphical user interface, whether or not it contains a graph, the notion of programmatic *focus* is very important for blind computer users.[5] While sighted users can easily rest their gaze on any item on the screen, blind users must use the keyboard or another device to move from item to item. In a graph-based application, the node or edge with focus is generally designated as the *current* or *selected* node/edge. The user may need to

[5]Programmatic focus is crucial for any computer user unable to use a pointing device and limited to the keyboard, a switch, or voice for input. U.S. Sec. 508 Guideline §1194.21 (c) states "A well-defined on-screen indication of the current focus shall be provided that moves among interactive interface elements as the input focus changes. The focus shall be programmatically exposed so that assistive technology can track focus and focus changes." [22]

move focus from this node/edge to associated information or another part of the interface and then back again.

Connection- and order-based as well hierarchical *navigation* of the graph itself have been incorporated in the applications to be discussed. Some applications provide the ability to go "back" and "forward" to previously visited nodes similarly to the way the world wide web is navigated. The ability to "bookmark" nodes also aids in navigation. Some applications mark nodes as "visited" as well. Other important issues include:

- Overview - With just a glance, a sighted user can get an idea of a graph's topology, relative size/complexity and features such as cycles and parallel paths. Efforts have been made to provide blind users with graph overviews as well.
- Verbosity Level - Because most information is conveyed to the blind user via speech which can sometimes be quite lengthy and time-consuming, applications often provide the ability to select one of several verbosity levels depending on the level of detail desired by the user.
- Details On Demand - Another method for dealing with an overwhelming amount of spoken information is by providing details only when requested by the user. This idea is part of the visual information-seeking mantra "overview first, zoom, filter, then details-on-demand" proposed by Schneiderman [23] that formed the basis of the Auditory Information-Seeking Actions (ASIA) developed by the iSonic creators [14].
- Sound - Non-speech sounds are used to offload the heavy demand on conveying information via speech and to make concurrent use of the auditory channel.
- Orientation - When blindly navigating a graph or interface, it is easy to get lost. Users sometimes need to ask "Where am I?" or reset the focus to a specific node in the graph such as the root or start node.
- Grouping - Miller's classic paper on *The Magical Number Seven* reports that only about seven (plus or minus two) pieces of information can be stored in short term memory at one time [24]. The process of recoding – grouping and naming chunks of information – increases the amount of information that can be retained. Deprived of the external memory provided by visual representations, blind users are especially reliant on their own internal mental models. Applying names to groups of nodes and subgraphs makes their internal graph representations more manageable.
- Annotation - Allowing the user to annotate nodes, edges, and subgraphs also serves as a memory aid.
- Search - We are all blind when it comes to navigating the gigantic graph known as the world wide web and regularly rely on search engines to find the information we need. At times, searching for and going directly to the desired node of a graph is more efficient for blind users than navigating to it via the edges.
- Create/edit - While some applications simply provide access to existing graphs, others provide blind people with the means to create and edit graphs. In order to share these graphs with sighted users, the use of graph layout algorithms or some other means must be employed.

- Positional Mental Model - Some applications provide the blind user with a means to build up a positional mental model of a specific physical layout for a graph.

- Universal Design

> ***Universal design is the design of products and environments to be usable by all people, to the greatest extent possible, without the need for adaptation or specialized design.*** **–Ron Mace**

Disabled architect, Ron Mace,[6] coined the term "universal design" and developed a set of universal design principles (see Table 3) that were originally applied to making physical environments accessible [25, 26]. Designing software that accommodates a wide range of users, including those with visual or other impairments, is equally important and beneficial. Universal design principles espouse the development of products that are equitable in use *without* segregating users. All of the graph-based applications reviewed in Section 2 were developed *separately* and *specifically* to accommodate blind users. While these efforts are very worthwhile, we feel that the development of applications that are inherently *universally accessible* is preferable. In Section 3, we describe our ProofChecker application which includes blind individuals in a graph-based application intended for universal use [27].

2 Related Work

The aforementioned issues have been addressed in a variety of ways in the following research efforts to make graphs accessible to blind computer users.

2.1 AudioGraf

AudioGraf was an early attempt to make graph-like diagrams accessible by means of a touch panel and an auditory display [28]. The diagrams consisted of three types of graphic elements – frames, texts, and connections. Graphic elements could be linked to each other; for example, a text might be associated with a frame. The user explored the diagram by touching the panel with varying degrees of pressure using one of four sizes of *focus*, square areas of the diagram surrounding the working point (user's finger). The focus sizes were zero (0 mm), small (4 mm), medium (16 mm), and large (36 mm). Touching the panel with a small amount of pressure activated *counter mode* which caused the number and types of elements in focus to be read aloud. A greater amount of pressure activated *element view*, causing important attributes of the elements in focus to be read and/or expressed with a non-speech *sound* such as a plucked string for a connection. An even greater amount of pressure switched the interface to *attribute mode* which auditorially displayed all attributes of the elements in focus. In this way, the AudioGraf prototype provided the user with three

[6] After receiving a degree in architecture from North Carolina State University, Ron Mace returned to found the Center for Accessible Housing which was later renamed The Center for Universal Design [25]. He was instrumental in creating the nation's first building code for accessibility as well as the passage of the Fair Housing Amendments Act of 1988 and The Americans with Disabilities Act of 1990.

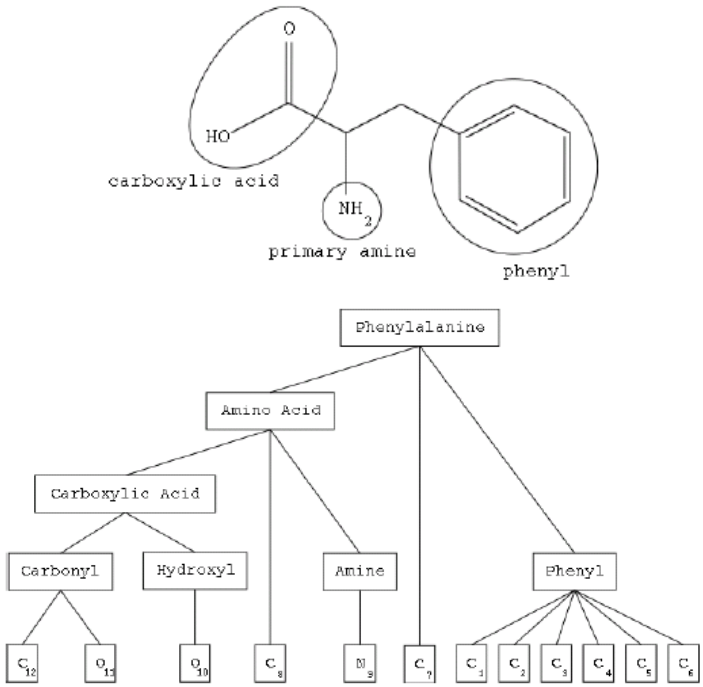

Figure 4: Hierarchical view of Phenylalanine. [30]

verbosity levels. It also allowed the user to build up a *positional mental model* of a simple graph. Usability tests found that users could correctly describe diagrams containing two frames and one connection, identifying the name, position, and size of the elements. They also determined that only the small and medium focus sizes were effective.

2.2 Kekulé

Kekulé was developed to enable blind students to examine the structure of chemical molecules — essentially graphs in which nodes represent atoms and edges represent the bonds between them [29, 30]. Molecules encoded as publicly available Chemical Markup Language (CML) files are used as input to Kekulé. Input from the user comes solely from the keyboard and output is provided in the form of speech via the Java-based FreeTTS [31].

Grouping was quite naturally incorporated into Kekulé by identifying the functional groups of a molecule using Ullmann's algorithm for subgraph isomorphism [32]. The functional groups were used to present a molecule to the user in a hierarchical fashion as shown in Figure 4. Both *hierarchical* and *connection-based navigation* are provided by Kekulé. *Hierarchical navigation* provides the user with the means to zoom in and out in order to examine the molecule at higher and lower levels of detail. *Connection-based navigation* allows the user to examine the functional groups and/or atoms that are connected via bonds to a given group or atom. *Focus* changes to a bonded atom/group in a "move as you

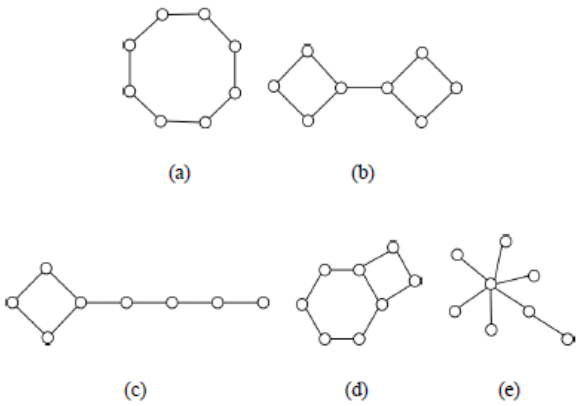

Figure 5: Undirected graphs conveyed via an auditory glance using Kekulé. [33]

hear" fashion. Pressing RETURN provides *details on demand* about the currently selected connection, atom, or functional group.

Evaluations found that users were reluctant to switch between the two modes of *navigation*. They also expressed a need for help with *orientation* because they sometimes had difficulty keeping track of their current position in a molecule and were confused as to which atom within a functional group had *focus*. They also wanted to find out what atoms and/or groups were connected to a given atom/group without necessarily changing their *focus* as dictated by Kekulé's "move as you hear" implementation of navigation.

The Kekulé platform was also used to investigate providing an "auditory glance" (*overview*) of an undirected graph to enable blind users to quickly get a general sense of the size, complexity, topology, and features of the graph in the same way that sighted users do [33]. Graphs such as the those in Figure 5 were sonified by playing a *tone* for the left most node followed by tones for all of the nodes connected to it, subsequently followed by tones for all nodes connected to them, and so on in a breadth-first manner. Left/right stereo effects were used to indicate a node's relative distance above or below the horizontal axis, respectively. A cycle was indicated by an ambient *sound* played during the tones for its nodes. This approach was tested using sighted users who were generally able to match the auditory glance to the correct graph in a group.

2.3 PLUMB

PLUMB (exPLoring graphs at University of Massachusetts Boston) was developed to help blind students comprehend graphs and data structures [35, 36, 37, 38]. Like AudioGraf, PLUMB allowed the user to build up a *positional mental model* of a graph. Initial work focused on auditorially presenting simple undirected graphs such as airline flight paths and organizational charts using a tablet PC. The stylus (pen) was used to *navigate* from vertex to vertex along an edge. The edge was conveyed by a continuous *musical tone* with a vibrato effect that increased in intensity at the ends. The tone's loudness decreased as the user moved farther from the central axis of the edge. When a vertex was entered, a *sound*

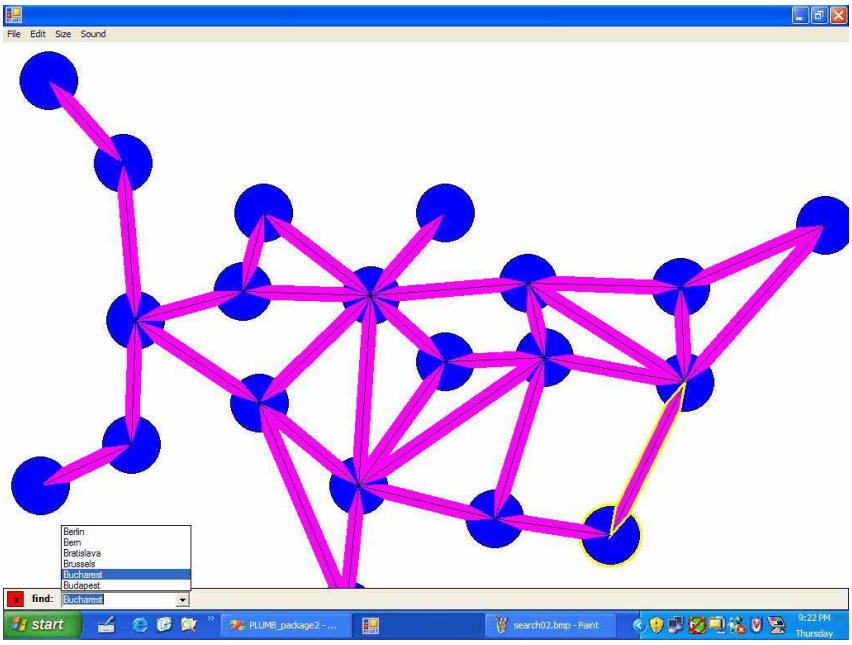

Figure 6: PLUMB representation of a map of Europe. [34]

was played and its name and sometimes a brief description were spoken. *Details on demand* about the element in *focus* could be requested via a right mouse click generated with the pen or by a mouse that had been partially disabled by removing the track ball. When no element was in *focus*, general information about the graph was provided instead.

A later version of PLUMB incorporated *keyboard navigation* in addition to *pen-based navigation*. The adjacency list of the current vertex was examined using the arrow keys. Pressing ENTER selected an edge and placed the *focus* on its opposite vertex. Pressing v or e provided *details on demand* about the current vertex or edge. Auto-complete *search* was added to allow users to choose a vertex as their starting point in graph exploration. Graphical editing tools and a command line interface allowed both sighted and non-sighted users to create or modify graphs, but how this was accomplished was not mentioned in the literature.

PLUMB was extended to PLUMB EXTRA^3 (EXploring data sTRures using Audible Algorithm Animation) in an effort to help blind students understand graph algorithms [34]. Students could choose a data structure and an operation to perform on it and use keyboard shortcuts to select the next or previous step in the algorithm. Descriptive text associated with the current step was spoken and the data structure could be examined at any point. Operations on linked lists and heaps were implemented.

Both applications were written in C# using the .NET platform. Graphs were rendered as Graph eXchange Language (GXL) files [39]. The files contained layout information as well as the information necessary to provide the appropriate speech and sound. These were generated using the Microsoft Speech API [40], DirectX [41], and the MIDI for .NET v2.0.4

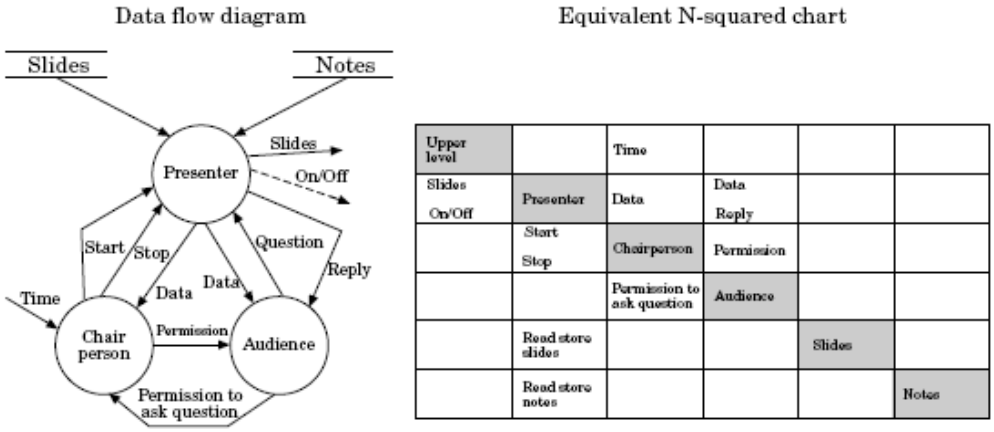

Figure 7: Kevin system mapping of a Data Flow Diagram (DFD) to an N^2 Chart. [42]

library.

2.4 Kevin

The Kevin system aimed to make data flow diagrams (DFDs) accessible to blind software engineers and computer science students [42]. A DFD consists of nodes known as "transformations" and labeled directed edges known as "data flows." The DFDs were based on the Ward-Mellor real-time structured analysis (RTSA) model [43] and as such were hierarchical in nature. That is, a given transformation could be decomposed into a lower level DFD.

The graphical representation of a DFD was mapped to an N^2 chart as illustrated in Figure 7. The transformations occupy the cells along the main diagonal from the top left to the bottom right. Data flow(s) leaving the transformation in row a and entering the transformation in column b occupy the cell in row a, column b.[7] The N^2 chart was conveyed to a blind user as part of a "talking tactile diagram" created by covering a touch sensitive tablet with a tactile overlay. When the user touched and applied pressure to a cell of the diagram, its contents were spoken aloud. In addition to the N^2 chart, the tactile overlay contained control buttons whose functions were designated by "hapticons" (tactile icons) as shown in Figure 8. Using this prototype together with the occasional use of the keyboard to name transformations and data flows, a blind user was able to read, edit, and create DFDs. Because Kevin can exchange DFDs with other Computer-Aided Software Engineering (CASE) tools through the CASE data interchange format (CDIF), blind users were able to read and modify DFDs created by sighted users. However, due to the lack of associated layout information, sighted users needed to rearrange the transformations and data flows in order to view DFDs created by blind users.

Kevin was informally evaluated by two blind users. One of the users, who was proficient in Braille, used the tactile interface very effectively while the other user, who was not, strug-

[7] Note that this representation does not allow for "self-loops" on transformations.

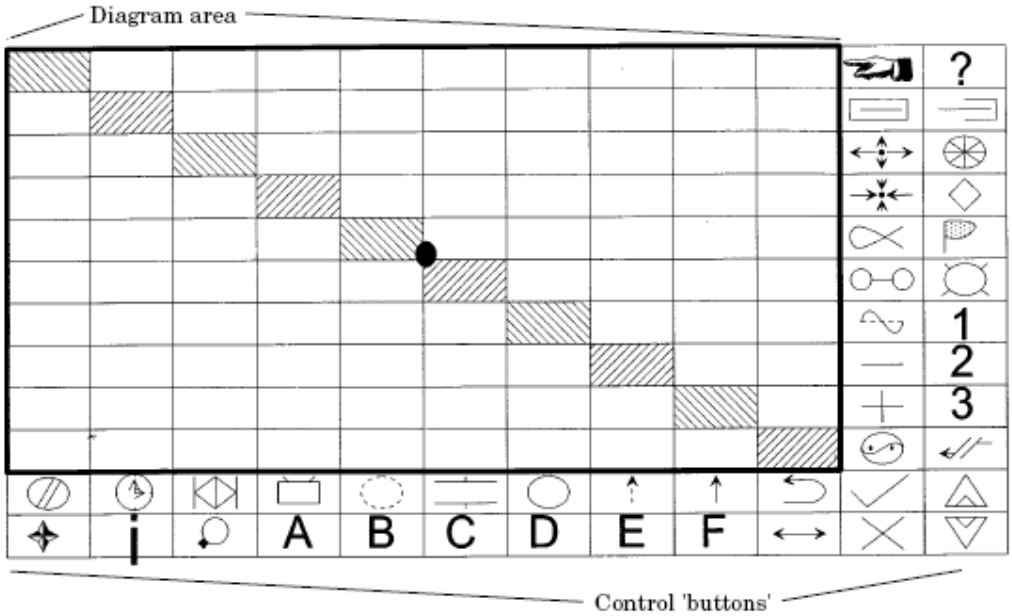

Figure 8: Kevin system "talking tactile diagram". [42]

gled with the evaluation tasks and expressed a preference for keyboard input. Comparisons of Kevin with static tactile DFD representations found that the latter lent themselves more readily to the task of determining "the complete set of data flows from one transformation to another." Based on that observation, a *query mode* that provided the path(s) between two transformations was added to Kevin.

Besides providing a tactile interface for blind users, Kevin is a fully functioning CASE tool. Its creators suggested that with the increasing use of the Windows Object Linking and Embedding (OLE) interface by commercially available CASE tools (in the late 1990s), a better approach would be to provide an accessible interface to these tools rather than recreating them.

2.5 TeDUB

The Technical Drawings Understanding for the Blind (TeDUB) project involved the Kevin system researchers, Blenkhorn and Evans, along with others [45, 44, 46, 47, 48, 49]. TeDUB strived to make existing Unified Modeling Language (UML) (*class, state, sequence, and use case*) and other types of diagrams accessible to the blind using the keyboard, joystick, and screen reader. UML diagrams were input in the form of XML Metadata Interchange (XMI) files exported from UML tools like Rational Rose [50] and ArgoUML [51]. These were converted to TeDUB's internal format using Extensible Stylesheet Language (XSL) transformations. The Microsoft DirectX 8 API provided the interface to the Microsoft Sidewinder and Saitek Cyborg 3D joysticks that were used.

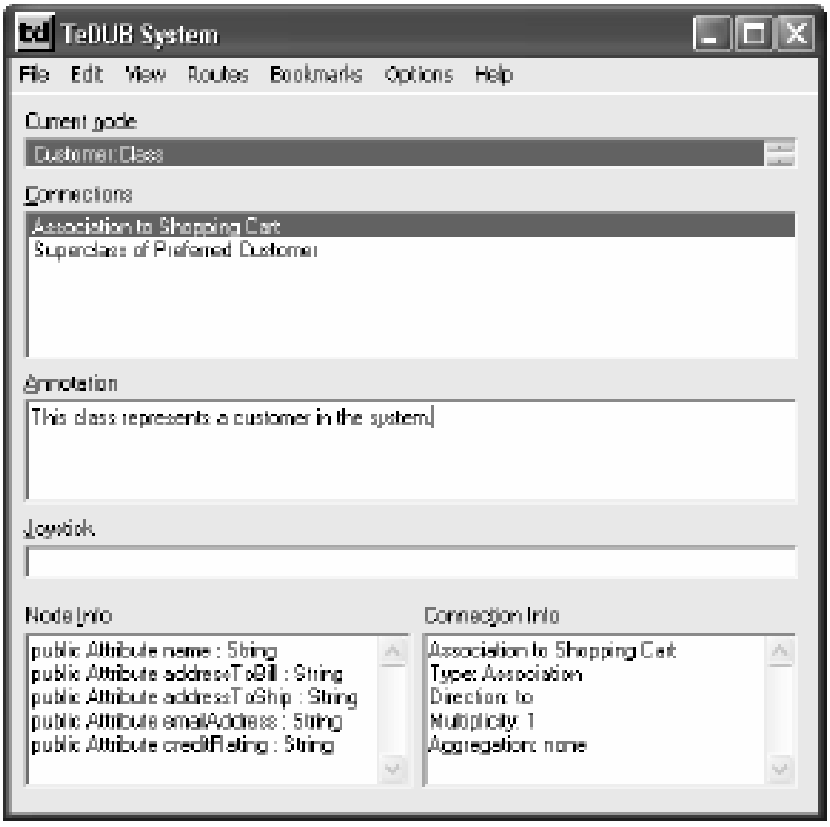

Figure 9: TeDUB interface. [44]

The underlying graph of a UML diagram was represented as a hierarchical tree containing individual nodes as well as "structural nodes" used for *grouping*. The structural nodes provided context or summary information about their component nodes. The tree structure was presented by a Windows Explorer style interface operated via the arrow keys thereby providing *hierarchical navigation*. Warning *sounds* alerted the user when trying to access a non-existent node. Nodes at the same level in the hierarchy were listed alphabetically with the exception of UML state charts whose start and end nodes were listed first and last respectively. This alphabetical listing facilitated the use of a letter key to cycle through all nodes beginning with a particular letter as is done in Windows-style lists. Users were able to "bookmark" nodes and retrace their path through the nodes using "back/forward" functionality. Help with *orientation* was provided via a keystroke that returned the user to the root node.

Additional information about the selected (current) node and its connections as well as the ability to *annotate* it were available through the TeDUB interface shown in Figure 9. A "find function," which operated similarly to Windows Notepad, allowed the user to seach

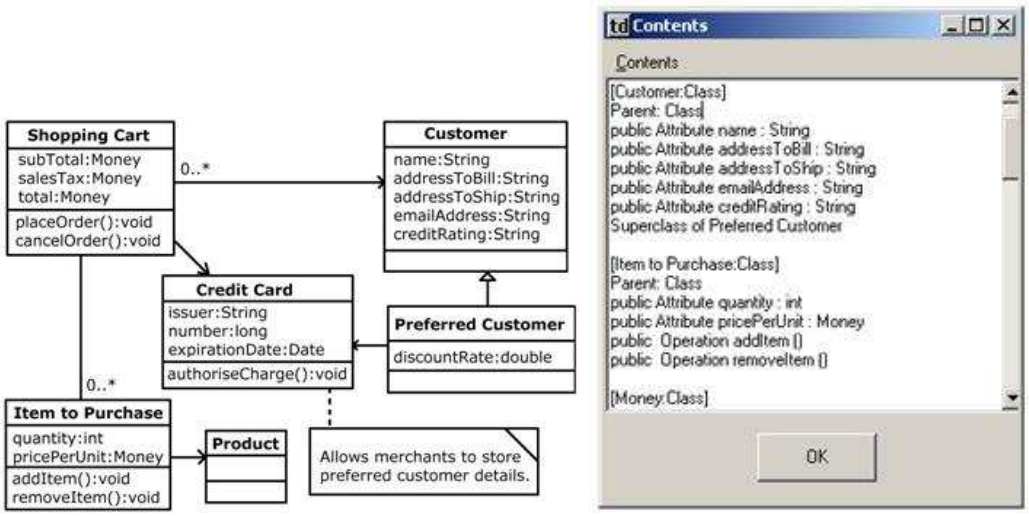

Figure 10: TeDUB UML class diagram represented via text. [44, 49]

for a node in the diagram as well as to seach the annotations. Selecting a connection in the TeDUB interface allowed the user to move to a new node thus providing *connection-based navigation*. Another option was to navigate the graph via a joystick. If a node was located in the current direction of the joystick, it was announced and its name appeared in the joystick textfield in the interface. Pulling the joystick trigger moved the focus to the indicated node. Keyboard users could similarly access nodes in 8 different directions from the current node by using the number pad. This method did not work for nodes with more than 8 connections. Neither the number pad nor the joystick worked when more than one node was located in the same direction from the current node. The TeDUB interface was augmented with a flattened Text View for UML diagrams as illustrated in Figure 10. This view provided information about the nodes but not their connections. Users were able to examine the Text View using standard keyboard and screen reader functions as well as a "find function" provided by the application.

Evaluations of the TeDUB project by blind users found that the use of a joystick to provide spatial information was helpful for maps, but not necessarily for other types of diagrams. Warning *sounds* were appreciated, but context *sounds*, for example, representing different countries in a map of Europe, were annoying and users turned them off. While they generally found TeDUB effective for examining UML diagrams, they expressed a strong desire to be able to create and examine them as well. Interestingly, Alasdair King, one of the TeDUB researchers, comments on his personal website [52] that "with hindsight, the best approach would be simply to convert a UML diagram (from an XMI file) into an HTML document with lots of internal linking and plain text. This would be simpler and fit with screenreader user's standard ability to read web pages . . . look at the XSL files in config to see how this might be accomplished." Considering that we are all "visually impaired" with respect to the world wide web, this may be a sensible approach.

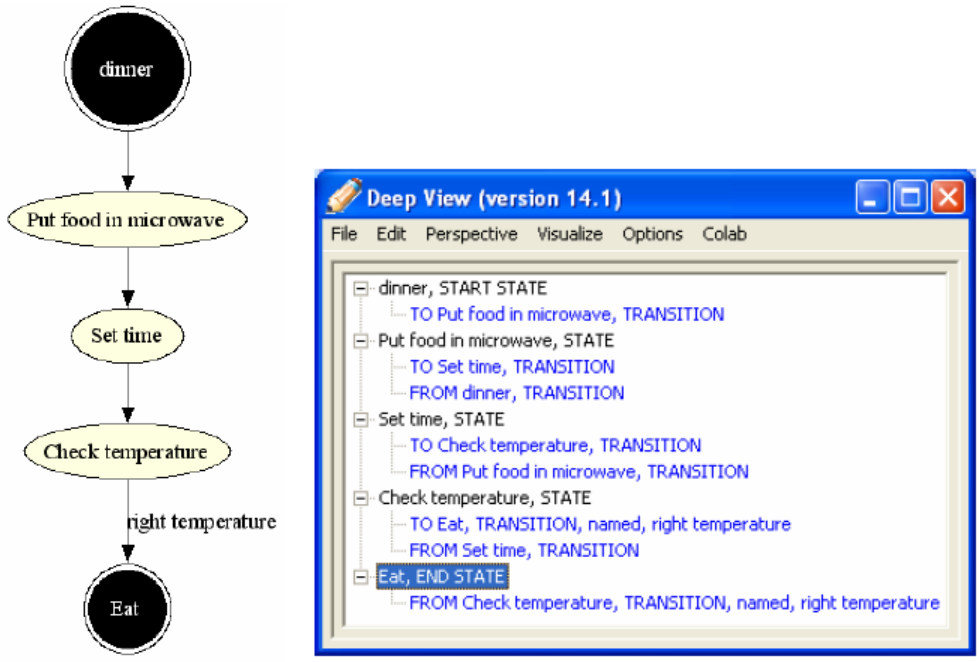

Figure 11: Representation of a state chart in the Deep View interface. [53]

2.6 Deep View

Deep View was developed to allow blind users to *create* and *edit* node-link diagrams, both alone and in collaboration with sighted users [53]. As a third-party plug-in, Deep View provides an accessible custom interface to commercial diagram drawing applications. The plug-in was incorporated by the researchers into Rational Rose [50] and Microsoft Visio [54] to provide access to state charts and Entity Relationship Diagrams (ERD) respectively. Deep View is written in Java using the Eclipse Standard Widget Toolkit (SWT) [55], whose widgets are accessible via a screen reader.

The Deep View interface and its representation of a simple state chart using a treeview are shown in Figure 11. Text describing nodes and links is customized for screen reader use by placing the unique information first, for example, "Check temperature, STATE" instead of "STATE, Check temperature." The underlying graph is *navigated* similarly to the world wide web. Selecting one of the links for the current node and pressing ENTER transfers focus to the other endpoint. Pressing BACKSPACE returns focus to the previous node. A visited node is noted by appending "visited" to its textual description. This scheme provides the user with *connection-based navigation*. Deep View's support of nested subdiagrams provides *hierarchical navigation* (see Figure 12).

The edit menu and/or keyboard shortcuts are used to bring up simple dialogs that allow the user to add nodes or links as well as edit the selected node/link as shown in Figure 13. The selected node/link is removed whenever DELETE is pressed. A *search* dialog allows the user to locate and transfer focus to a particular node. Another dialog provides the path(s)

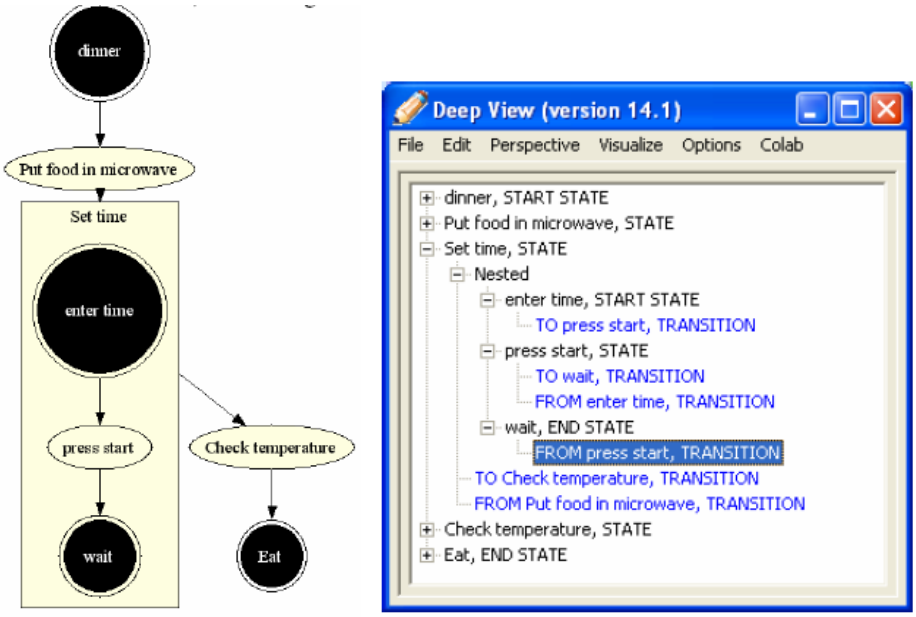

Figure 12: Deep View representation of a subdiagram. [53]

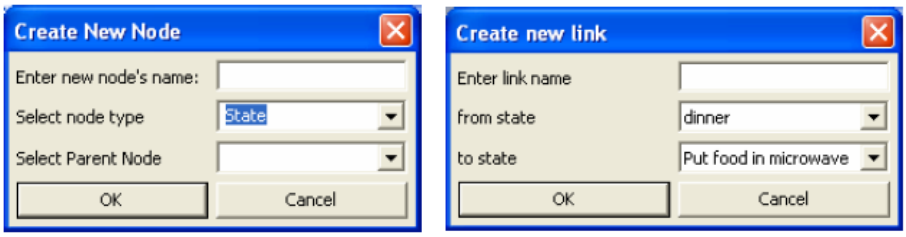

Figure 13: Deep View dialogs for creating new nodes and links. [53]

between two nodes. An *overview* of graph features such as parallel paths (see Figure 14) and cycles can be detected and presented to the user. Diagrams created in the Deep View standalone interface may be rendered as visual diagram images using a GraphViz web service [56, 57].

Deep View can also be used to allow blind and sighted users to collaborate on creating and working with a diagram. While a blind person uses the Deep View interface, a sighted person uses the graphical interface provided by Rational Rose. The Sync subsystem, which maintains a consistent shared data model, is used for communication between the interfaces as well as concurrency control [58]. Changes made by a user are immediately apparent in the other user's interface. Nodes created by a blind user are placed in rows across the top of the sighted user's drawing canvas. Short *audio icons* alert the blind user that an addition, deletion, or change has been made by the sighted user. A keyboard shortcut

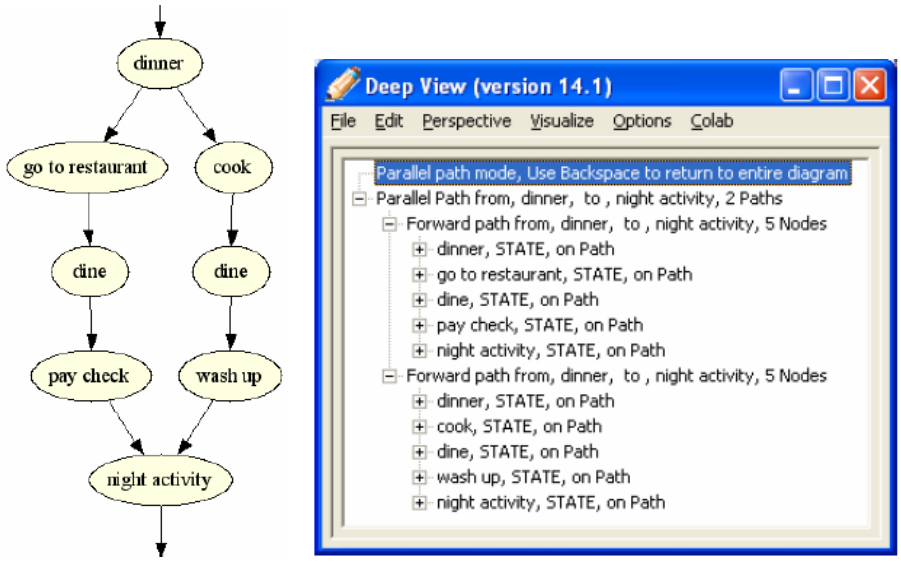

Figure 14: Deep View representation of parallel paths. [53]

can be used to display a summary of recent edits to the diagram. Working together is enhanced through a "semantic pointing" mechanism which allows nodes under discussion to be highlighted by changing their background color in the visual interface with an option in the Deep View interface to list only the highlighted nodes. The blind user's currently focused node is indicated by making its font larger and bolder in the visual interface thus allowing the sighted user to follow the blind user's navigation of the graph, a notion termed "follow-me pointing."

Deep View was evaluated by five blind participants and four blind/sighted pairs who were able to use the interface to create and edit node-link diagrams. An interesting finding was that the blind participants functioned by memorizing large portions of the diagrams.

2.7 Summary

A summary of the features included in the reviewed graph-based applications is provided in Table 2. The table also includes ProofChecker, our application which is discussed in the next section.

3 Work to Date

One of our goals is to include blind people in applications intended for general use, thereby adhering to universal design principles as discussed in Section 1. Another important goal is to provide them with graph representations that are "computationally equivalent" to those employed by sighted people and not simply "informationally equivalent." According to Larkin and Simon [1]:

δ	a	b
q_0	q_2	q_3
$*q_1$	q_0	q_1
q_2	q_0	q_1
q_3	q_2	q_3

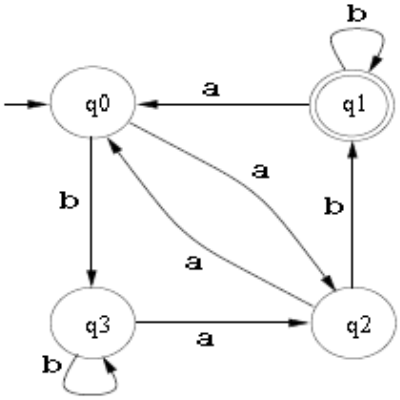

Figure 15: Transition table and bubble diagram representations of a DFA.

> Two representations are *informationally equivalent* if all of the information in the one is also inferable from the other, and vice versa. Each could be constructed from the information in the other. Two representations are *computationally equivalent* if they are informationally equivalent and, in addition, any inference that can be drawn easily and quickly from the information given explicitly in the one can be drawn easily and quickly from the information given explicitly in the other, and vice versa.

For example, in the context of automata theory, the same deterministic finite automaton (DFA),[8] may be represented by a transition table or a bubble diagram as shown in Figure 15. These representations are informationally equivalent, but when trying to determine if a string such as *abbabab* is in the language of the DFA, it is most likely quicker and easier to carry out this task using the diagram rather than the table. Thus, they are not computationally equivalent. Whether or not the applications reviewed in the previous section provide blind people with graph representations that are computationally equivalent to those available to sighted people is unclear. However, all of them segregate blind users to a separate interface and thus do not follow universal design principles.

Our efforts to provide computationally equivalent access to graphs include work on two graph-based applications intended for universal use, ProofChecker [27] and SAS® Enterprise Miner [21]. Both are Java applications that make use of the Java Accessibility API [59] and are intended for use with the JAWS screen reader [4]. In order for a Windows-based assistive technology such JAWS to interact with the Java Accessibility API, the Java Access Bridge [60] must be installed. Examples and guidelines for making Java applications accessible may be found in [61] and on our website located at `http://research.csc.ncsu.edu/accessibility/`.

[8] A DFA is a type of directed graph.

3.1 ProofChecker

ProofChecker is a graphical program based on the notion of formal correctness proofs that allows students to draw a DFA and determine whether or not it correctly recognizes a given language. Because the states of a DFA partition the language over its alphabet into equivalence classes, each state has a language associated with it. Conditions that describe the language of each state are entered by the student in the form of conditional expressions with function calls and/or regular expressions. A brute-force approach is then used to check that each state's condition correctly describes all of the strings in its language and that none of the strings in the state's language meet the condition for another state. Feedback is provided that either confirms that the DFA correctly meets the given conditions or alerts the student to a mismatch between the conditions and the DFA. A student's DFA can be saved in an XML file and reopened in ProofChecker and/or submitted for grading. An automated checking tool, known as ProofGrader, can be used by instructors to determine if a student's DFA and the correct DFA for a given language are equivalent.

We originally designed ProofChecker to be a very visual application as shown in Figure 16. Like most graphical applications, its point-and-click interface relied heavily on the mouse to draw and manipulate DFAs. When faced with the challenge of making ProofChecker accessible to a blind automata theory student, our first approach was to have him use an editor to modify an XML file which recorded a DFA and its state conditions and then have ProofChecker tell him whether or not the new DFA was correct. While this approach was informationally equivalent, it was very tedious and certainly not computationally equivalent. Fortunately, without changing the sighted interface nor adding an awkward extension to this interface, we were able to make it accessible to him and other blind students. This was accomplished by mapping keystrokes to actions and setting the accessible descriptions provided through the Java Accessibility API appropriately. Because the original sighted interface is intact, blind and sighted individuals can use ProofChecker to simultaneously access the same graph. What follows is a detailed description of the use of ProofChecker in the context of automata theory correctness proofs as well as its original and accessible interface.

3.1.1 Overview of ProofChecker

The ProofChecker Graphical User Interface (GUI) has three main panels. The largest panel is a canvas where the DFA is drawn. To the left of the canvas is a scrolling panel with a text area for each state in which a condition describing its language can be entered. The top left corner contains a panel where the student can enter comments about the DFA, for example, a description of its language or its homework problem number.

The GUI also contains two tool bars, a general one which will be used for other formalisms as well, and one specific to manipulating DFAs. A menu bar is available for working with files, checking the DFA, and accessing Help.

This section describes how a sighted user would typically interact with ProofChecker using the mouse, keyboard, and graphical components. The next section describes how a blind user can do the same things using only the keyboard and the JAWS screen reader.

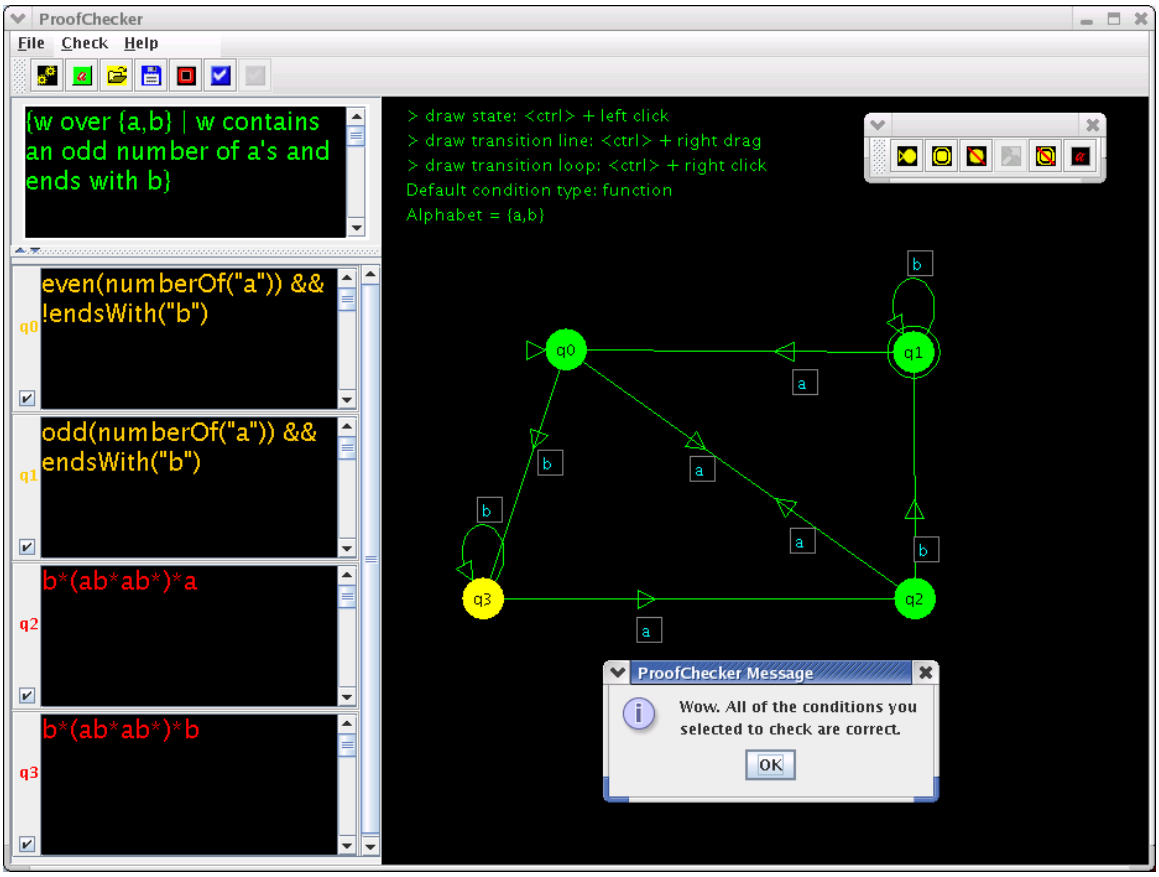

Figure 16: ProofChecker Graphical User Interface. [27]

Creating and Editing a DFA A new state is created by holding down the `CTRL` key and left-clicking with the mouse on the canvas area. The `CTRL` key plus a right-click on a state creates a self-transition. Right-dragging from one state to another while holding down the `CTRL` key creates a transition between them. Each transition has an associated text box where its alphabet symbol(s) can be entered. Clicking on the text box for an existing transition allows for editing of its symbols.

A state and its transitions can be moved by dragging the state to a new location on the canvas. Tool bar buttons are used to mark the selected state as the start and/or an accepting state as well as to remove it and its transitions from the DFA. A button to remove a single transition is also available.

Entering State Conditions Each state has an associated text area where a condition describing its language may be entered. The default for expressing conditions is Function Mode in which the strings in a state's language are described in the form of a conditional expression using one or more ProofChecker functions. Right-clicking on the text area brings

up a popup menu containing the available function choices as well as the option to switch to Regular Expression Mode.

Checking the DFA for Correctness When the **Check** button is pressed or **Check > State Conditions** is selected from the menu, the DFA is checked for correctness. The student is first alerted if the DFA is not well-formed, i.e., does not contain a start state, does not have a transition from each state on every alphabet symbol, etc. If it is well-formed, then the state conditions the student has selected will be checked. This is done by generating all strings over the alphabet up to an arbitrary length ℓ. For each string, the program checks that (a) it meets the condition of the state in which it ends up and (b) that it meets the condition of *only* that state. Setting the value of ℓ to $n + 2$, where n is the number of states, allows the checking to be done in a reasonable amount of time and is generally successful in ferreting out student errors. After many semesters of using ProofChecker, only one counterexample has been found in which a student's conditions were correct for strings of length up to $n + 2$, but the DFA was not. If the state conditions do not correctly match the DFA, an error message is provided in a popup dialog box.

3.1.2 The Accessible Interface

The same functionality described in the previous section is provided to blind students through keyboard shortcuts and audible feedback.

Navigation While sighted students can visually examine the GUI and select states, transitions, menu options, and text areas with the mouse, blind students must rely on audible feedback and the keyboard to do the same.

The up/down arrow keys are used to move the programmatic *focus* from state i/i+1 to state i+1/i in a DFA thereby providing *order-based navigation*. The `ENTER` key is used to toggle between the selected state and its transitions. When a state's transitions have *focus*, the up/down arrow keys are used to move between them. Thus a state or transition can be selected analogous to clicking on it with the mouse.

Each time a state is selected, its name and whether or not it is the start state and/or an accepting state is voiced by the JAWS screen reader. We cause the voicing to occur by resetting the Java accessible description for the canvas panel each time the focus and/or the DFA changes. The voicing of information about a state's "to" and "from" transitions may be turned on/off using `ALT-T` and `ALT-F` respectively. This provision of several *verbosity levels* allows the DFA to be audibly examined in the most efficient way for the task at hand.

The menu bar and menus are accessed through the use of mnemonics, with `ALT-F`, `ALT-C`, and `ALT-H` being used to select the **File**, **Check**, and **Help** menus respectively. Pressing `F1` brings up a list of all available keyboard shortcuts as shown in Figure 17. This list is automatically read aloud by the screen reader.

Creating and Editing a DFA `CTRL-N` is used to create a new state. States are drawn on the canvas in rows of 5 each with state qi being drawn at row $i/5$, column i mod 5. This can make for quite a jumble of states and transitions which is of absolutely no consequence to a blind student. By dragging with the mouse, a sighted fellow student or instructor can move the states to make the DFA more visible to them.

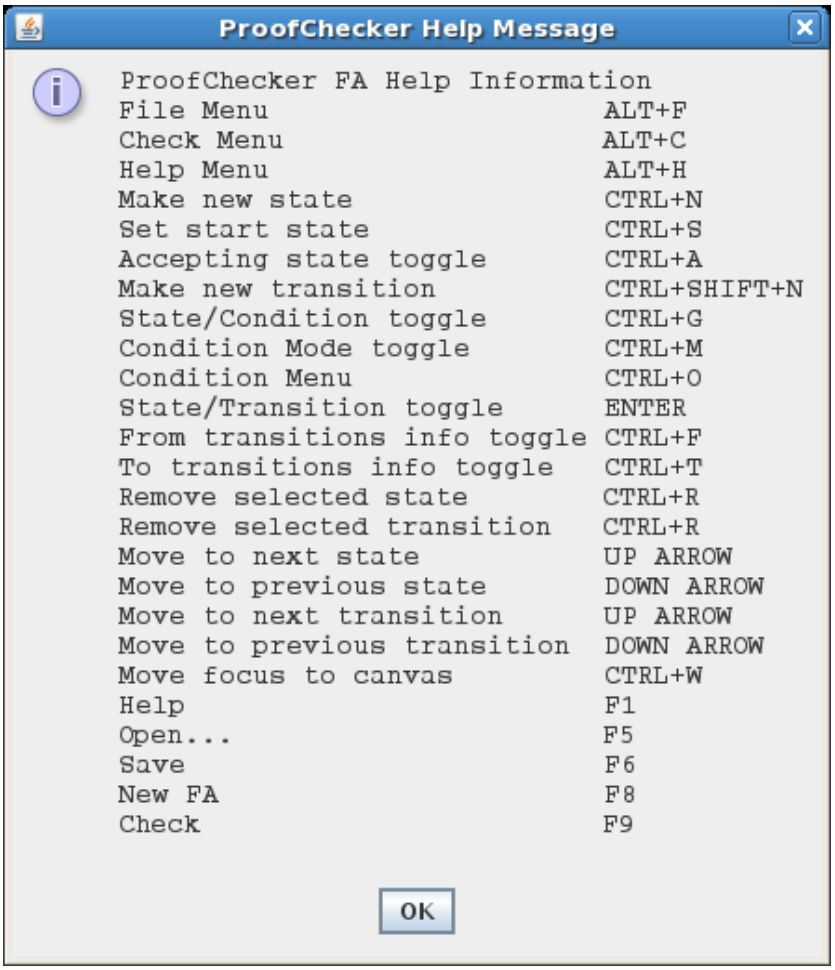

Figure 17: ProofChecker Help Message. [27]

A selected state is marked as the start state with `CTRL-S` and an accepting state with `CTRL-A`. A transition from state qi to qj is created by first moving to state qi, entering `CTRL-SHIFT-N`, then moving to state qj and entering `CTRL-SHIFT-N` to complete the transition. A self-transition on a selected state is created by entering `CTRL-SHIFT-N` twice. `CTRL-R` is used to remove a state and its transitions or to remove a single transition. Table 4 gives an example of using the keyboard to create part of the DFA in Figure 16 together with the audible feedback voiced by JAWS.

Entering State Conditions `CTRL-G` is used to move from the selected state and its condition and vice-versa. `CTRL-O` is used to popup the function choices; the up/down arrow keys are used to move between them and the space bar is used to select one of them. `CTRL-M` toggles between Function Mode and Regular Expression Mode.

Checking the DFA for Correctness Entering `ALT-C` followed by `ALT-S` initiates checking of the DFA. The contents of the popup dialog box that alerts students to errors or success is read aloud by JAWS.

3.2 SAS® Enterprise Miner

Based on our work with ProofChecker, the author was invited to work at SAS Institute to improve the accessibility of Enterprise Miner, a data mining application. The layout of its interface, as shown in Figure 3, is very similar to that of ProofChecker. The Diagram Workspace panel on the right contains a process flow diagram (PFD). Information associated with the currently selected node or link is displayed in an editable Property Sheet Table to the left of the Diagram Workspace. Due to Sec. 508 regulations, it is important that Enterprise Miner be accessible via the keyboard and JAWS screen reader. The following enhancements were made.

Process Flow Diagram The PFD nodes and links were navigable via the keyboard, but the selected node/link was not voiced by JAWS. This was remedied by resetting the accessible description of the PFD panel each time a Java ItemEvent occurred which signaled a move to and the selection of a different node/link. This caused JAWS to voice the name of the newly selected node/link.

Interface Navigation In order to examine and edit the Property Sheet Table associated with the selected node/link, a blind person must be able to transfer focus to the table using the F6 key.[9] Pressing F6, however, caused the node/link to be deselected and its Property Sheet to be removed from the interface. This was remedied by trapping and consuming the F6 keypress and transferring focus directly to the Property Sheet.

Property Sheet Table Whenever a cell of a Property Sheet Table was selected, its content was not read aloud by JAWS. This was remedied by resetting the accessible description of the table whenever a Java ListSelectionEvent occurred which signaled the move to a new table cell. This caused JAWS to voice the content of the newly selected table cell.

3.3 Summary

ProofChecker was successfully used to create DFAs by the blind student in question who was also involved in the accessibility efforts. While he still preferred designing DFAs on paper with the help of a sighted scribe, he felt that ProofChecker had great potential for making DFAs and graphs in general more accessible to blind students. He was also the subject of a user study of ProofChecker and Enterprise Miner conducted at SAS Institute by the author. During the ProofChecker study, we observed that detecting mistakes in a DFA that are readily apparent to a sighted user took quite a bit longer for him. Finding ways to "level the playing field" between sighted and blind users is an interesting challenge. The results of the Enterprise Miner study are company confidential.

[9] The F6 key is used to navigate among the internal windows in a multiple document interface.

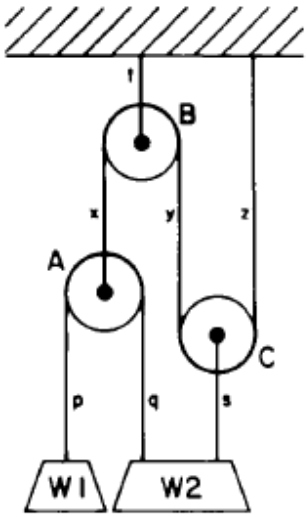

Figure 1. Diagram of the pulley problem.

Figure 18: Pulley problem from Larkin and Simon. [1]

4 Future Directions

Retired Stanford University professor Dr. Jeff Ullman states that ideally a thesis topic should be selected based on the need of some "customer."[62] Certainly blind programmers and computer science students qualify as "customers" based on their need to access graphs, especially UML diagrams. Unfortunately blind programmers are increasingly left out of the design process due to the use of UML for object-oriented design. Despite the work to date, the need for accessible UML and simple graph sketching tools is still current as evidenced by recent email messages posted to the Programming Blind mailing list [63, 64, 65]. Packages such as Microsoft Office Visio, while somewhat accessible to screen reader users, are intended for *drawing* graphs and other types of diagrams, *not* for manipulating them and using them for problem solving. Based on our experience with ProofChecker and Enterprise Miner and drawing on ideas from the reviewed work, we propose the development of a universally accessible graph sketching tool.

A blind beginning programming student was recently observed using the Windows Notepad editor as a "computational tool" for evaluating a complicated arithmetic expression, such as **41 % 7 * 3 / 5 + 5 / 3 * 4.5**. He copied and pasted the expression into Notepad, found the leftmost operation of highest precedence, replaced it with its value, and continued in this manner until only one operand remained. This same student had difficulty in physics due to the inaccessibility of the diagrams. In a similar way to the Notepad example, an accessible graph sketching tool would provide him and others with a tool for working not only with graphs, but some physics and other types of problems. For example, the pulley problem shown in Figure 18 is used by Larkin and Simon as an example of how a diagram is more readily used for problem solving than a lengthy sentential form of the

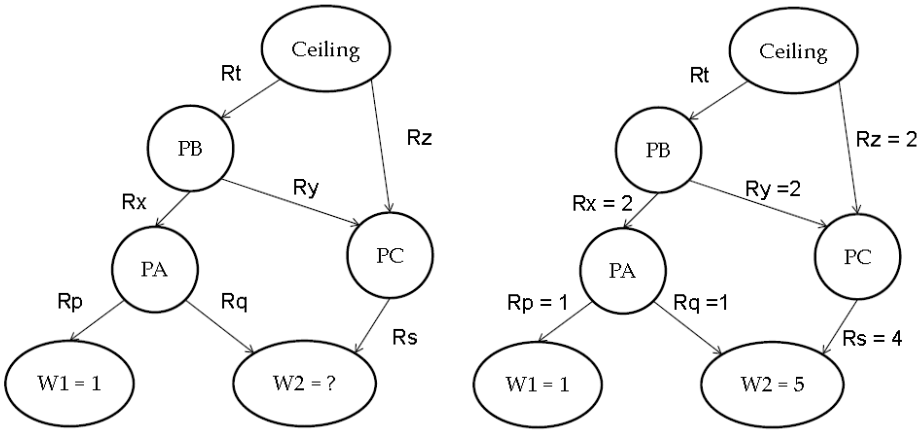

Figure 19: Pulley problem solved using graph representation.

same problem [1]. This problem involves determining weight W2 given weight W1. The same diagram could be drawn as a graph and subsequently solved as illustrated by Figure 19. With the ability to navigate the graph and modify the labels of the nodes and edges, a blind (or sighted) student could use physics principles to determine the tensions on the ropes and eventually the value of W2.

The development of a universally accessible graph sketching tool would allow blind and sighted people to work alone or together using the same interface to create and/or examine graphs and use them for design, problem solving, and teaching/learning. By providing animation capability the tool could be used for illustrating various graph algorithms such as breadth/depth first search, minimum spanning tree, shortest path, etc. as well as the solution of problems in other domains – see Figure 20 for an example [66]. If a sequence of graphs in an animation (or a single graph) were output to a Braille embosser, a blind student could study it in tactile form. An even more exciting possiblity would be the use of a full screen refreshable Braille display to provide an animation [67].

We plan to do the following initial work towards the development of a universally accessible graph sketching tool:

- Build a graph sketching prototype that allows users to create directed and undirected graphs using the mouse and/or keyboard.
- Identify graph creation and interaction methods for blind users which come as close as possible to providing them with a computationally equivalent experience to that of sighted users. The interaction methods should leverage the unique abilities and accommodational strategies of blind people such as superior item recall and serial memory, the tendency to encode spatial information in a sequential rather than global representation [68], and heightened auditory awareness.
- Develop graph layout techniques that allow graphs created by blind people to be shared with sighted people. Possibilities include automated graph layout algorithms, layout

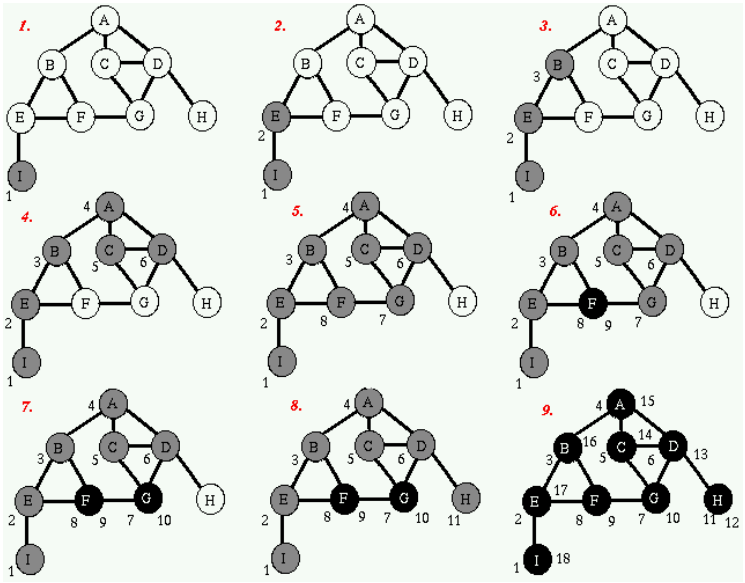

Figure 20: Depth-first Search Example. [66]

commands (North, South, etc.), predefined templates for various types of graphs (tree, free body diagram, etc.), and the use of a recursive 3x3 grid (See Figure 2).

Our continued work on a universally accessible graph sketching tool will provide groundbreaking work in an important research area as well as practical help for blind people who currently have very limited access to combinatorial graphs.

References

[1] Jill H. Larkin and Herbert A. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11(1):65–100, 1987. 1, 16, 23, 24

[2] Forrester Research Inc. The wide range of abilities and its impact on computer technology. Technical report, Microsoft Corp., 2003. Available at `http://www.microsoft.com/enable/download/default.aspx`. 1

[3] Eric Bergman and Earl Johnson. Towards accessible human-computer interaction. *Advances in human-computer interaction (vol. 5)*, pages 87–113, 1995. 2

[4] Jaws for windows.
Available at `http://www.freedomscientific.com/fs_products/software_jaws.asp`. 2, 17

[5] Window-Eyes.
Available at `http://www.gwmicro.com/Window-Eyes/`. 2

[6] ZoomText Magnifier/Reader.
Available at `http://www.aisquared.com/Products/ZoomTextMRD/index.cfm`. 2

[7] Orca.
Available at `http://live.gnome.org/Orca`. 2

[8] Linux screen reader.
Available at `http://live.gnome.org/LSR`. 2

[9] Voiceover.
Available at `http://www.apple.com/accessibility/voiceover/`. 2

[10] Dragon naturallyspeaking.
Available at `http://www.nuance.com/naturallyspeaking/`. 2

[11] Richard E. Ladner, Melody Y. Ivory, Rajesh Rao, Sheryl Burgstahler, Dan Comden, Sangyun Hahn, Matthew Renzelmann, Satria Krisnandi, Mahalakshmi Ramasamy, Beverly Slabosky, Andrew Martin, Amelia Lacenski, Stuart Olsen, and Dmitri Groce. Automating tactile graphics translation. In *Assets '05: Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, pages 150–157, New York, NY, USA, 2005. ACM. 2

[12] Chandrika Jayant, Matt Renzelmann, Dana Wen, Satria Krisnandi, Richard Ladner, and Dan Comden. Automated tactile graphics translation: in the field. In *Assets '07: Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*, pages 75–82, New York, NY, USA, 2007. ACM. 2

[13] P. Parente and G. Bishop. BATS: The Blind Audio Tactile Mapping System. *Proceedings of the ACM Southeast Regional Conference*, 2003. 2

[14] Haixia Zhao, Catherine Plaisant, Ben Shneiderman, and Jonathan Lazar. Data sonification for users with visual impairment: A case study with georeferenced data. *ACM Trans. Comput.-Hum. Interact.*, 15(1):1–28, 2008. 3, 5

[15] Hesham M. Kamel and James A. Landay. A study of blind drawing practice: creating graphical information without the visual channel. In *Assets '00: Proceedings of the fourth international ACM conference on Assistive technologies*, pages 34–41, New York, NY, USA, 2000. ACM Press. 3

[16] Hesham M. Kamel and James A. Landay. Sketching images eyes-free: a grid-based dynamic drawing tool for the blind. In *Assets '02: Proceedings of the fifth international ACM conference on Assistive technologies*, pages 33–40, New York, NY, USA, 2002. ACM Press. 3

[17] James L. Alty and Dimitrios I. Rigas. Communicating graphical information to blind users using music: the role of context. In *CHI '98: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 574–581, New York, NY, USA, 1998. ACM Press/Addison-Wesley Publishing Co. 3

[18] James L. Alty and Dimitrios Rigas. Exploring the use of structured musical stimuli to communicate simple diagrams: the role of context. *Int. J. Hum.-Comput. Stud.*, 62(1):21–40, 2005. 3

[19] Dimitrios Rigas and James Alty. The rising pitch metaphor: an empirical study. *Int. J. Hum.-Comput. Stud.*, 62(1):1–20, 2005. 3

[20] D. Sumikawa. Guidelines for the integration of audio cues into computer user interfaces. Technical Report UCRL 53656, Lawrence Livermore National Laboratory, 1985. 3

[21] Data mining with SAS® Enterprise Miner.
Available at `http://www.sas.com/technologies/analytics/datamining/miner/`. 4, 17

[22] Section 508 homepage: Electronic and information technology. Available at `http://www.access-board.gov/508.htm`. 4

[23] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *VL '96: Proceedings of the 1996 IEEE Symposium on Visual Languages*, page 336, Washington, DC, USA, 1996. IEEE Computer Society. 5

[24] G.A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for information processing. *Psychological Review*, 63(2):81–97, 1956. 5

[25] The Center for Universal Design, College of Design, North Carolina State University. Available at `http://www.design.ncsu.edu/cud/`. 6, 33

[26] Sheryl E. Burgstahler and Rebecca C. Cory, editors. *Universal Design in Higher Education From Principles to Practice*. Harvard Education Press, 2008. 6

[27] Matthias F. Stallmann, Suzanne P. Balik, Robert D. Rodman, Sina Bahram, Michael C. Grace, and Susan D. High. Proofchecker: an accessible environment for automata theory correctness proofs. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 48–52, New York, NY, USA, 2007. ACM Press. 6, 17, 19, 21, 34

[28] Andrea R. Kennel. AudioGraf: a diagram-reader for the blind. In *Assets '96: Proceedings of the second annual ACM conference on Assistive technologies*, pages 51–56, New York, NY, USA, 1996. ACM Press. 6

[29] A. Brown, R. D. Stevens, and S. Pettifer. Issues in the non-visual presentation of graph based diagrams. In E. Banissi, K. Borner, C. Chen, M. Dastbaz G. Clapworthy, A. Faiola, E. Izquierdo, C. Maple, J. Roberts, C. Moore, A. Ursyn, and J. J. Zhang, editors, *Proceedings of the 8th International Conference on Information Visualisation*, pages 671–676. IEEE, July 2004. 7

[30] Andy Brown, Steve Pettifer, and Robert Stevens. Evaluation of a non-visual molecule browser. In *Assets '04: Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility*, pages 40–47, New York, NY, USA, 2004. ACM Press. 7

[31] FreeTTS. Available at `http://freetts.sourceforge.net/`. 7

[32] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976. 7

[33] A. Brown, R. D. Stevens, and S. Pettifer. Audio representation of graphs: a quick look. In *Proceedings of the 12th International Conference on Auditory Display*, June 2006. 8

[34] Matt Calder, Robert F. Cohen, Jessica Lanzoni, Neal Landry, and Joelle Skaffa. Teaching data structures to students who are blind. In *ITICSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, New York, NY, USA, 2007. ACM Press. 9

[35] Robert F. Cohen, Rui Yu, Arthur Meacham, and Joelle Skaff. Plumb: displaying graphs to the blind using an active auditory interface. In *Assets '05: Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, pages 182–183, New York, NY, USA, 2005. ACM Press. 8

[36] Robert F. Cohen, Valerie Haven, Jessica A. Lanzoni, Arthur Meacham, Joelle Skaff, and Michael Wissell. Using an audio interface to assist users who are visually impaired with steering tasks. In *Assets '06: Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*, pages 119–124, New York, NY, USA, 2006. ACM Press. 8

[37] Matt Calder, Robert F. Cohen, Jessica Lanzoni, and Yun Xu. PLUMB: an interface for users who are blind to display, create, and modify graphs. In *Assets '06: Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*, pages 263–264, New York, NY, USA, 2006. ACM Press. 8

[38] Robert F. Cohen, Arthur Meacham, and Joelle Skaff. Teaching graphs to visually impaired students using an active auditory interface. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 279–282, New York, NY, USA, 2006. ACM Press. 8

[39] Richard C. Holt, Andy Schürr, Susan Elliott Sim, and Andreas Winter. Gxl: A graph-based standard exchange format for reengineering. *Science of Computer Programming*, 60(2):149–170, 4 2006. 9

[40] Microsoft speech api (sapi) 5.3. Available at `http://msdn.microsoft.com/en-us/library/ms723627(VS.85).aspx`. 9

[41] DirectX 10. Available at `http://www.microsoft.com/games/en-US/aboutgfw/Pages/directx10.aspx`. 9

[42] P. Blenkhorn and D. G. Evans. Using speech and touch to enable blind people to access schematic diagrams. *J. Netw. Comput. Appl.*, 21(1):17–29, 1998. 10, 11

[43] P.T. Ward and S.J. Mellor. Structured development for real-time systems, volume 1-3. *New Jersey: Prence Hall*, 1985. 10

[44] A. King, P. Blenkhorn, D. Crombie, S. Dijkstra, G. Evans, and J. Wood. Presenting UML software engineering diagrams to blind people. *Proceedings of 9th International Conference on Computers Helping People with Special Needs,(Lecture Notes in Computer Science LNCS 3118)*, 2004. 11, 12, 13

[45] Helen Petrie, Christoph Schlieder, Paul Blenkhorn, David Gareth Evans, Alasdair King, Anne-Marie O'Neill, George T. Ioannidis, Blaithin Gallagher, David Crombie, Rolf Mager, and Maurizio Alafaci. TeDUB: A system for presenting and exploring technical drawings for blind people. In *ICCHP '02: Proceedings of the 8th International Conference on Computers Helping People with Special Needs*, pages 537–539, London, UK, 2002. Springer-Verlag. 11

[46] M. Horstmann, C. Hagen, A. King, S. Dijkstra, D. Crombie, D.G. Evans, G.T. Ioannidis, P. Blenkhorn, O. Herzog, and C. Schlieder. TeDUB: Automatic interpretation and presentation of technical diagrams for blind people. *CVHI 2004—Conference and Workshop on Assistive Technologies for Vision and Hearing Impairment*, 2004. 11

[47] Mike Födisch, David Crombie, and George Ioannidis. TeDUB: Providing access to technical drawings for print impaired people.
Available at `citeseer.ist.psu.edu/542183.html`. 11

[48] Technical drawings understanding for the blind.
Available at `http://www.tedub.org/tedubsystem_en.html`. 11

[49] Alasdair Robin King. *Re-presenting visual content for blind people*. PhD thesis, University of Manchester, Manchester, England, 2006.
Available at `http://www.alasdairking.me.uk/research/PhD.htm`. 11, 13

[50] Rational Rose product line. Available at
http://www-01.ibm.com/software/awdtools/developer/rose/. 11, 14

[51] ArgoUML. Available at http://argouml.tigris.org/. 11

[52] Alasdair King. TeDUB and accessible UML.
Available at http://www.alasdairking.me.uk/tedub/index.htm. 13

[53] Dorian Miller. *Can we work together?* PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 2009.
Available at http://search.lib.unc.edu/search?R=UNCb5970444. 14, 15, 16

[54] Microsoft Office Visio.
Available at http://office.microsoft.com/en-us/visio/default.aspx. 14

[55] SWT: The standard widget toolkit. Available at http://www.eclipse.org/swt/. 14

[56] E.R. Gansner and S.C. North. An open graph visualization system and its applications to software engineering. *Software Practice and Experience*, 30(11):1203–1233, 2000. 15

[57] Graphviz - graph visualization software. Available at http://www.graphviz.org/. 15

[58] Jonathan P. Munson and Prasun Dewan. Sync: A java framework for mobile collaborative applications. *Computer*, 30(6):59–66, 1997. 15

[59] Package javax.accessibility. Available at http://java.sun.com/j2se/1.5.0/docs/api/javax/accessibility/package-summary.html. 17

[60] Sun Developer Network. Java access bridge. Available at
http://java.sun.com/products/accessbridge/. 17

[61] Robert F. Cohen, Alexander V. Fairley, David Gerry, and Gustavo R. Lima. Accessibility in introductory computer science. In *Proc. 36th SIGCSE Tech. Symp. on Computer Science Education*, pages 17–21, 2005. 17

[62] Jeffrey D. Ullman. Viewpoint advising students for success. *Commun. ACM*, 52(3):34–37, 2009. 23

[63] Programming blind email list.
Available at http://www.freelists.org/list/programmingblind. 23

[64] Accessibility of software for degree email thread, March 2010. Available at
http://www.freelists.org/post/programmingblind/Accessibility-of-software-for-degree. 23

[65] Creating simple graphics email thread, June 2010. Available at
http://www.freelists.org/post/programmingblind/Creating-simple-graphics,1. 23

[66] McGill university: School of computer science winter 1997 class notes for 308-251b data structures and algorithms topic #26: Depth-first search. Available at
http://www.cs.mcgill.ca/ cs251/OldCourses/1997/topic26/. 24, 25

[67] N. Di Spigna, P. Chakraborti, D. Winick, P. Yang, T. Ghosh, and P. Franzon. The integration of novel eap-based braille cells for use in a refreshable tactile display. volume 7642, page 76420A. SPIE, 2010. 24

[68] Noa Raz, Ella Striem, Golan Pundak, Tanya Orlov, and Ehud Zohary. Superior serial memory in the blind: A case of cognitive compensatory adjustment. *Current Biology*, 17(13):1129 – 1133, 2007. 24

Table 1: Laws Governing Accessibility in the United States

Rehabilitation Act of 1973, as amended in 1998	
Section 504	-requires universities receiving federal assistance to provide equal access to students with disabilities.
Section 508	-requires federal agencies to make their electronic and information technology (EIT) accessible to members of the public with disabilities. -mandates that most purchases of EIT by the federal government be accessible to disabled federal employees.
Americans with Disabilities Act of 1990 (ADA)	
	-outlaws discrimination in the private sector.
Telecommunications Act of 1996	
Section 255	-requires manufacturers of telecommunications products and services to make them accessible to people with disabilities.

Table 2: Summary of Accessible Graph-based Applications

Feature	AudioGraf	Kekulé	PLUMB	Kevin	TeDUB	Deep View	ProofChecker
Connection-based Navigation	√	√	√		√	√	
Order-based Navigation							√
Hierarchical Navigation		√		√	√	√	
Overview		√				√	
Verbosity Level	√						√
Details on Demand	√	√	√				
Sound	√	√	√		√	√	
Orientation					√		
Grouping		√		√	√	√	
Annotation					√		
Search			√		√	√	
Create/edit			?	√		√	√
Positional Mental Model	√		√		√		
Universal Design							√

Table 3: Universal Design Principles [25]

Principle	Definition
1. Equitable Use	The design is useful and marketable to people with diverse abilities.
2. Flexibility in Use	The design accommodates a wide range of individual preferences and abilities.
3. Simple and Intuitive Use	Use of the design is easy to understand, regardless of the user's experience, knowledge, language skills, or current concentration level.
4. Perceptible Information	The design communicates necessary information effectively to the user, regardless of ambient conditions or the user's sensory abilities.
5. Tolerance for Error	The design minimizes hazards and the adverse consequences of accidental or unintended actions.
6. Low Physical Effort	The design can be used efficiently and comfortably and with a minimum of fatigue.
7. Size and Space for Approach and Use	Appropriate size and space is provided for approach, reach, manipulation, and use regardless of user's body size, posture, or mobility.

Keystroke	Comment	Accessible description
`CTRL-n`	q0 drawn at position (0,0)	“State q0 is selected.”
`CTRL-s`	Incoming arrow drawn to q0	“State q0 is selected. It is the start state”
`CTRL-n`	q1 drawn at position (0,1)	“State q1 is selected.”
`CTRL-a`	Ring drawn around q1	“State q1 is selected. It is an accepting state”
`CTRL-shift-n`	Starts drawing transition from q1	“”
`CTRL-shift-n`	Self-transition drawn to q1; focus in textfield	“q1 self-loops on the symbols:”
`b`	b entered in transition textfield	“q1 self-loops on the symbols: b.”
`enter`	q1 is highlighted	“State q1 is selected. It is an accepting state”
`CTRL-shift-n`	Starts drawing transition from q1	“”
↓	q0 is highlighted	“State q0 is selected. It is the start state. Currently creating a transition starting from state q1.”
`CTRL-shift-n`	Transition drawn from q1 to q0. focus in textfield	“q1 transitions to state q0 on the symbols: ”
`a`	a entered in transition textfield	“q1 transitions to state q0 on the symbols: a.”

Table 4: Accessible Interface Example. [27]