Interactive Audio Documents

T. V. Raman*
Cambridge Research Lab, Digital Equipment Corp.
E-mail: ⟨raman@crl.dec.com⟩

David Gries
Computer Science, Cornell University
E-mail: ⟨gries@cs.cornell.edu⟩

Abstract

Communicating technical material orally is often hindered by the relentless linearity of audio; information flows *actively* past a passive listener. This is in stark contrast to communication through the printed medium, where we can actively peruse the visual display to access relevant information.

AsTeR is an interactive computing system for *audio formatting* electronic documents (presently, documents written in (LA)TeX) to produce audio documents. AsTeR can speak both literary texts and highly technical documents that contain complex mathematics. In fact, the effective speaking and interactive browsing of mathematics is a key goal of AsTeR. To this end, a listener can browse both complete documents and complex mathematical expressions. AsTeR thus enables *active* listening.

This paper describes the browsing component of AsTeR. The design and implementation of AsTeR is beyond the scope of this paper. Here, we will focus on the browser, and refer to other parts of the system in passing for the sake of completeness.

Keywords

Interactive Audio Renderings, Audio Browsing, Browsing Structure, In-context Rendering and Browsing, Spoken mathematics.

1 Introduction

AsTeR is a computing system that *audio formats* electronic documents to produce audio documents. AsTeR can speak both literary texts and highly technical documents (presently in (LA)TeX[4, 5, 6]) that contain complex mathematics. The first author has used it to listen to many texts and articles in computer science and mathematics, as well as Shakespeare. AsTeR far surpasses what can be done, say, by audio cassettes produced by the RFB (Recordings For the Blind), for three reasons.

- AsTeR can begin speaking a paper or book minutes after it has been received. Because of the number of books being produced and the trained manpower required, RFB often takes years to put a requested book on a cassette.
- Because of the precision of a voice synthesizer, AsTeR can use the parameters of the synthesizer (e.g., head size, pitch, stereo, tones) to reduce verbiage, thus shortening the audio rendering considerably and making it easier to comprehend. For example, subscripts may be spoken in a lower pitch, so that cues like "begin subscript" are unnecessary.
- AsTeR is interactive: the listener can stop the system while listening and browse the document.

A full description of AsTeR appears in [9]. In this paper, we focus on the part of AsTeR that makes listening interactive, the *browser*. The rest of this paper will focus on the browser, and mention other parts of AsTeR for the sake of completeness.

Note that the browsing techniques presented in this paper depend on the structure present in the information being conveyed. Research on browsing human speech [2, 1, 3] can be regarded as being orthogonal and complementary to the work presented here. The problem of traversing information organized as a directed asyclic graph (DAG) using telephonie interfaces has been addressed by [11, 14, 12]. Our approach is similar; the system described in [11], HyperVoice, regards the information as a series of linked lists, whereas we present our browser in terms of traversing a hierarchical tree structure.

*Supported by Digital Equipment Corp.

When perusing printed text, a reader can quickly skip portions of the document, reading only portions of interest. Typeset documents allow such structured browsing by using layout cues to present the underlying document structure; from here, the eye's ability to "randomly" access portions of the two-dimensional printed page appears to take over. The passive information in a printed document is accessed by an active reader capable of selectively perusing the text. Hence, visual documents themselves need not be interactive.

This passive-active relationship is reversed in traditional oral communication; the information flows *actively* past a passive listener, who has little control over what is heard. The problem is particularly severe when presenting structured information (e.g., complex mathematics) —a listener is likely to lose interest by the time the relevant information is presented. Hence, we need to enable active listening, i.e., enable the listener to determine what is heard. To be effective, audio renderings need to be interactive.

Our browser allows a user to interactively traverse the internal high-level representation of a document (described in [8, 10, 9]) and listen to portions of interest. The browser provides basic tree-traversal commands that can be composed to effectively browse the information structure. The browser is characterized by its simple command set; commands are of the form <*verb*> rather than the more complex <*verb*> + <*noun*>. The simplicity of the command set makes the browser flexible and extensible; the same set of commands are used to browse both general document structure (e.g., sections and subsections), as well as complex mathematical expressions.

This design is motivated by the conjecture that most of visual browsing actions are directed by the underlying structure present in the information. Thus, when we read a complex mathematical expression containing a fraction, we can quickly look at the numerator while reading the denominator. This single action of looking up at the numerator can be decomposed into a series of atomic tree traversal movements with respect to the structure of the expression. In the visual context, these actions happen extremely fast, leading to a feeling that the eye can access relevant portions of the visual display almost randomly. However, this notion of "randomness" disappears when we consider that such visual browsing becomes difficult in a badly formatted document where the underlying structure is not so apparent. Similarly, even when presented with a well-formatted document, a person unfamiliar with the subject matter finds it impossible to perform the same kind of visual browsing. Visual browsing thus depends on familiarity with the underlying structure and a clear rendering of this structure. ASTER parallels this functionality by building up a rich internal representation and providing a set of atomic actions to traverse this representation. The effectiveness with which a user can browse this representation is now a function of the user's familiarity with the structure in the subject matter being presented.

We present the browser as follows. Section 2 motivates the need for a browser by analyzing how visual browsing works. Based on this, we derive a corresponding model for audio browsing and identify a set of atomic browsing actions that enable general browsing. Section 3 describes how a user can traverse the high-level representation of a document. This section introduces the concept of a *current selection* and describes how the user is unobtrusively cued to the nature of the current selection. Section 4 describes how the listener can execute actions after setting the current selection. These actions include listening to the current selection, rendering it relative to its parent, and listening to the rest of the document. Section 5 discusses an important component of technical documents: cross references. The ASTER mechanism for cross references allows listeners to annotate cross-referenceable objects with meaningful labels that can be used to refer to such objects in later cross references, thus making cross references easier to listen to and comprehend. Section 4 also describes how places of interest in a document can be marked using a bookmark facility.

We conclude this introduction with two remarks. First, ASTER is written in CLOS, which is an object-oriented extension of Lisp, and is referenced from an EMACS window. A few concepts and notations from CLOS and EMACS are used in this paper, but we attempt to write so that the reader who has no knowledge of CLOS can understand the paper. See [13, 15] for a description of CLOS. Second, this paper suffers from being a printed (passive) document that describes audio (inter*active*) documents! A full appreciation of ASTER can be obtained only by using ASTER to listen to a document.

2 How does browsing work?

Visual versus audio communication

As a first step toward developing an effective audio analogue, let us examine communication using the printed page. The printed page is passive: it is a two-dimensional visual display with marks on it. Readers are active: they can scan the material linearly or browse through parts of the document, and the power of the printed medium lies in the eye's ability to browse. For example, we are able to skim the text, focus on an interesting paragraph or formula, and quickly skip to a footnote and back again.

Visual layout (the way the marks appear on paper) aids browsing. Line-breaks, paragraphs, section headings, indentation, and displayed formulas help the reader perceive *chunks* of printed matter and to read portions of the text selectively.

In contrast, in the audio setting, the document is active —scrolling away in a linear fashion— while the ear is passive. This makes the goal of achieving an audio analogue to the printed page seemingly difficult.

Structure aids browsing

The eye is capable of moving to any point on a page extremely rapidly. Yet, browsing is not random movement. Typically, the eye moves to the next word, line, paragraph, or section heading. The eye seems to infer some structure that it uses to move effectively around the printed document. Since each action is being performed extremely rapidly, owing to the eye's inherent scanning ability, these atomic actions are difficult to pinpoint.

We can, however, conjecture that:

> *A well-formatted document presents inherent logical structure, which the eye is capable of perceiving. Visual browsing actions can be characterized as movements directed by this structure.*

For example, consider the layout tree for a formatted page. The layout structure consists of a root node, which is the page, and several children, which are the paragraphs on the page. At the next level of this tree, we have the lines, each line is further broken up into words, and words themselves are broken up into characters. Consider the following atomic (simple) tree-traversal movements:

1. Move to the next sibling.
2. Move to the previous sibling.
3. Move to the parent.
4. Move to the leftmost child.
5. Move to the rightmost child.
6. Mark the current node.
7. Return to the marked node.

All browsing actions can be defined as combinations of these atomic actions. For example, on encountering a reference to a footnote, we:

1. Mark the current node.
2. Move to the parent (this gets us out of the current paragraph).
3. Move across siblings until the footnote is located.
4. Read the footnote.
5. Return to the marked node.

As a more complex example, consider the following expression as read by a person familiar with mathematical notation:

$$D_x^n w = \sum_{0\le j\le n} \sum_{\substack{k_1+k_2+\cdots+k_n=j \\ k_1+2k_2+\cdots+nk_n=n \\ k_1,k_2,\ldots,k_n\ge 0}} D_u^j w \frac{n!(D_x^1 u)^{k_1}\cdots(D_x^n u)^{k_n}}{k_1!(1!)^{k_1}\cdots k_n!(n!)^{k_n}} \tag{1}$$

The experienced reader is able to quickly scan this expression and, while perusing the denominator, access the numerator. This ability is a consequence of internalizing the underlying structure conveyed by the visual layout and using it to traverse the information. The atomic actions in accessing the numerator are:

1. Mark the current node.
2. Read the previous sibling.
3. Return to the marked node.

We enable audio browsing by giving the listener the ability to perform the same kind of traversals. ASTER internalizes a sufficiently rich structure to permit all of these browsing actions.

3 Traversing structure

Our internal representation for document structure is an attributed tree. Tree structures are easy to traverse, and they provide a uniform way of browsing structure present in both plain text as well as mathematical formulae. This section outlines our approach to enabling such browsing actions.

All browsing actions are defined with respect to the *current selection* (a node in the internal tree representation of the document). The current selection is initially the top of the document —the root of the internal representation. The current selection can be changed in two ways:

- Interrupt the rendering of the current selection by executing command stop[1]. Commands stop and quit are described in detail in Section 4.
- Move the selection when no rendering is in progress. Typically, a single key stroke is used for each of the commands listed in the following paragraphs.

The following browser commands can be executed when no rendering is in progress. Our key-mapping for these commands is inspired principally by the UNIX VI editor. These key-bindings are shown only as an example; it is trivial under Emacs to rebind these browser commands to other keys.

[1]Contrast this with command quit.

left-superscript	accent	superscript
$\nwarrow$	$\uparrow$	$\nearrow$
	math object	
$\swarrow$	$\downarrow$	$\searrow$
left-subscript	underbar	subscript

Figure 1: Math object with attributes (each attribute is also a math object)

- **t**: Move to the top of the document.
- **C-u t**: Move to the top of the current math expression.
- **h**: Move left: set current selection to previous sibling.
- **l**: Move right: set current selection to next sibling.
- **j**: Move down: set current selection to first child.
- **k**: Move up: set current selection to parent.

Next, these browser actions are augmented to enable the traversal of the attributed tree structure that represents the document. In this representation, all nodes have *content* and, optionally, *children* and *attributes* (subscript, superscript, accent, underbar, left subscript, and left superscript). These attributes are displayed in Figure 1 on page 4.

- **i**: Move to content: set current selection to the contents of the current selection.

The following actions move the selection to the various attributes. The parent of an attribute is defined to be the object being attributed. The result of moving to attributes can therefore be undone by moving back to the parent.

- **^**: Move to superscript.
- **_**: Move to subscript.
- *****: Move to accent.
- **#**: Move to underbar.
- **!**: Move to left subscript.
- **%**: Move to left superscript.

The above key-map[2] for traversing the attributes was arrived at as follows: The choice for superscript and subscript is automatic, since the key strokes match the symbols used by TEX to markup these attributes. Placing the fingers on the row of numerals on a standard keyboard, the actions necessary for typing ^ and _ are mimicked with the left hand to arrive at the key-bindings for the left superscript and subscript. The middle finger of each hand is used to get to the accent/underbar.

[2] A user of ASTER is free to re-map these actions to other keys.

Key	Action	Context	Type
		Right hand side is	Summation
j	first child	summand is	summation
j	first child	summand is	juxtaposition
j	first child	first term is	derivative
l	next sibling	second term is	fraction
j	first child	numerator is	product
l	next sibling	denominator is	product

Table 1: Traversing Faa De Bruno's formula. The context and type are spoken

Tables are the only objects in our internal representation that do not conform completely to the tree-traversal model. This is because each table element is linked to its parent as well as to its four neighbors. The left and right neighbors can be modeled as siblings, but we need extra links and hence extra actions to traverse the entries by columns.

- **a**: Move to element above.
- **d**: Move to element below.

Summarizing the selection

As pointed out earlier, browser commands are of the form <*verb*>. The effect of browser commands is determined by the current context, i.e., the current selection. When a browsing action is executed, ASTER summarizes (speaks a summary of) the resulting selection. These summaries are designed to be concise but informative enough to avoid the "lost in space" problem that sometimes happens when complex structures are traversed. A summary consists of:

- **Context**: Contextual information about where the current selection occurs with respect to the rest of the document.
- **Type**: The type of the current selection. This usually involves summarizing the current selection, which is described below.

Table 1 on page 4 shows a sample interaction resulting from moving down the right hand side of Faa de Bruno's formula shown in equation 2 on page 3.

Object Type	Summary
Article	Title
Sectional unit	Section title
Complex Math object	Operator appearing at the root
Math object (leaf node)	Render node

Table 2: Summarizing document objects.

In our CLOS implementation, each kind of tree node is an object type and each tree node is an object of its type. Summaries of objects are summarized by generic function summarize, so methods on this function can be defined for each object type in order to define the summary for that type. Hence, the listener who has enough knowledge and experience can customize summaries. Summaries are typically of the form "Context is Type". We show examples of summaries in Table 2 on page 5. Messages like these have been found to be sufficient to avoid the lost-in-space problem. In cases where insufficient information is available to generate a complete summary of an object instance, the type of that object is spoken.

Contextual information (e.g., what the children of math objects are called) is available to ASTER—e.g., children of an inference are called "premise" and "conclusion", and children of a fraction are called numerator and denominator. Such information is exploited in giving contextual information about the current selection.

Traversing the structure of mathematical expressions is particularly useful when used in conjunction with the variable-substitution[3] rendering style —described in [9, 7]. In fact, such traversal can be thought of as a dual to variable substitution. If an expression has been rendered once using variable substitution, then future traversals of that expression use the variable names generated in the substitution process when cueing the current selection. This proves to be a useful memory aid when understanding complex equations like Faa de Bruno's formula shown in equation 2 on page 3.

Traversing structure is also useful when listening to large documents, like textbooks. The actions of the browser enable the listener to move quickly through the document without having to listen to a lot of text. In conjunction with the ability to switch rendering styles, this enables the quick location of portions of interest. For instance, a listener can activate a rule that renders only the mathematics in a document. Once an equation of interest is encountered, the listener can interrupt the rendering, move the current selection from this point to the enclosing paragraph or sectional unit, and then listen to the relevant portion of the document.

[3] This style presents the listener with the top-level view of an expression before selectively rendering the subexpressions.

4 Rendering the current selection

The current selection is rendered by executing browser command read-current. Renderings commenced by read-current can be interrupted in two ways:

- Quit: Interrupt the rendering and restore the current selection.
- Stop: Interrupt the rendering and leave the current selection at the object last rendered.

Browser command read-rest renders the rest of the document starting from the current selection.

Moving the selection to the next or previous node and then rendering is a common sequence of actions. ASTER provides commands read-previous and read-next (bound to **p** and **n** respectively) that combine these actions. In addition, the browser provides command read-just-this-node (bound to **r**) that renders only the contents of the current node, rather than the entire subtree rooted at this node —this is useful when traversing complex mathematical expressions.

Relative renderings

Suppose the listener stops the rendering, moves through the structure to some point (say, a subscript of a variable), and asks for the current selection to be rendered. Should this rendering be given in context or should it be rendered independently? For example, suppose subscripts are typically rendered in a lower pitch. Should the subscript be rendered in context, in a lower pitch, or independently, as if it were a document all by itself?

Rendering the subscript independently is easy. Rendering it in context may be computationally intensive, since, in a naive implementation, it may require the whole document to be rendered (silently) in order to calculate the context. As an analogy, consider choosing an arbitrary word from the electronic encoding of a book and asking the question:

> On what page of the book does this word appear?

In general, answering such a question would involve completely rendering the book.

In ASTER, much of the computation needed for in-context rendering is avoided by saving, with each document object, the state of the audio system when that object was first rendered. Then, if that object is to be rendered in context again, the state of the audio system can be suitably set with a few instructions.

On the surface, this solution seems storage intensive. However, many objects in a document are rendered using the same state —for example, every word in a paragraph. Since the rendering states are represented by pointers to data, the required space is not infeasible.

Relative renderings are very useful when listening to tables and matrices. These are rendered using stereo effects, with the spatial location of the sound indicating the position of elements in the table. When moving through the elements of a table, hearing each element at the right spatial coordinate is useful in keeping track of the location of the current selection. For example, if the listener moves the current selection to the rightmost column and then moves down this column, each element is spoken on the right speaker.

As another example of using relative renderings, consider traversing the right-hand side of Faa de Bruno's formula shown in equation 2 on page 3. Moving to the subscript of the summation with _ and then executing statement readcurrent-relatively results in the constraint being rendered in the "subscript voice" —this cues the location of the current selection.

5 Cross references

The browser allows the traversal of additional links introduced by cross references in a document. During the recognition step, such cross references are represented as hypertext links in the internal representation. Browser commands allow the listener to traverse such links. Command read-follow-cross-ref (bound to g) can be executed after interrupting the rendering. By default, this renders the next cross reference in the document; a prefix argument[4] results in the previous cross reference being rendered. This is necessary —typically, the rendering will have been stopped close to, rather than on, a cross reference.

AsTeR also allows cross-reference links to be traversed while rendering is in progress. When a cross reference is spoken, AsTeR plays a short sound cue that acts as a prompt. If the user responds with y, the cross reference is rendered before continuing. The presence of this functionality almost obviates the need for the user to call command read-follow-cross-ref interactively.

How a cross-reference tag is rendered depends on the object that is cross-referenced. For instance, rendering a cross-reference tag to a section results in the section number and title being spoken —this is more useful than just speaking the section number. Typically, the renderings of cross-reference tags are designed to give as much information as possible without being unduly verbose. Renderings of cross-reference tags are customizable by providing methods on generic function read-cross-reference; see [9] for examples.

[4] Emacs terminology for C-u as a prefix to the command.

Cross-referenceable objects

Technical documents use cross references to numbered equations and theorems to make the presentation succinct. Cross references in the running text typically use a system-generated number, e.g., equation 3.2.1. Even with printed material, this convention can present problems. Consider, for instance, a proof that reads:

> By equation 3.1, theorem 4.2 and equation 8, we get equation 9 and hence the result.

If the above is difficult to understand when perusing it in print, consider the poor listener, who does not have the luxury of quickly turning pages!

The ability to follow a cross-reference tag during the rendering does mitigate this problem to some extent. However, following such cross references can prove distracting. For example, if the listener were to follow each of the cross-references in the example shown above, he would miss the meaning of the statement that contains the cross-references. AsTeR overcomes this by allowing the listener to assign meaningful labels to cross-referenceable objects when they are rendered. These user-supplied labels are used when rendering cross-reference tags, almost eliminating the need to follow cross references. In our experience, we most often refer back to a cross reference only to identify the entity being referred to, and user-defined labels on such cross-referenceable objects enable precisely this.

To give a concrete example, consider listening to a book on Fermat's last theorem. The first chapter might introduce the subject by stating the theorem. Assume that this theorem is numbered 1.1. The rest of the book might refer to this theorem by number, as in:

> As a corollary of this result, we can prove theorem 1.1.

When AsTeR renders this document, the listener hears the theorem, the system-generated number (theorem 1.1), and a fleeting sound cue. At this point, the user can give the theorem a more meaningful name by pressing y and entering an appropriate label. Assume the listener enters "Fermat's last theorem". Later, when rendering cross references to this theorem, the user-defined label will be used instead of the system-generated theorem number. Thus, the example shown above would be rendered as:

> As a corollary of this result, we can prove Fermat's last theorem.

Bookmarks

The browser provides a simple bookmark facility, which can be used to mark positions of interest to be returned to later. Browser command mark-read-pointer (bound to **m**) prompts the user for a bookmark name and marks the current position. Command follow-bookmark (bound to **f**) takes a bookmark name and returns to the marked position. Bookmarks can be accessed in two ways: **f** renders the marked object without moving the current selection and **C-u f** resets the current selection to the marked position.

References

[1] Arons, B. The design of audio servers and toolkits for supporting speech in the user interface. *Journal of the American Voice I/O Society* (Mar. 1991), 27–41.

[2] Arons, B. Hyperspeech: Navigating in speech-only hypermedia. *In Hypertext '91 ACM* (1991), 133–146.

[3] Arons, B. Techniques, perception, and applications of time-compressed speech. *In Proceedings of 1992 American Voice I/O Society* (Sept. 1992), 169–177.

[4] Knuth, D. E. *The TEXbook.* Addison-Wesley, Reading, Massachusetts, 1984.

[5] Knuth, D. E. *TEX The Program.* Addison-Wesley, Reading, Mass., 1986.

[6] Lamport, L. *LATEX: A Document Preparation System.* Addison-Wesley, Reading, Mass., 1986.

[7] Raman, T., and Gries, D. Audio formatting –making spoken text and mathematics comprehensible. *2nd. International Conference on Auditory Displays* (Nov 1994). Submitted.

[8] Raman, T. V. Documents are not just for printing. *Proceedings of the 1st Workshop on the Principles of Document Processing* (Oct. 1992).

[9] Raman, T. V. *Audio System for Technical Readings.* PhD thesis, Cornell University, May 1994. *URL* ftp://ftp.cs.cornell.edu/pub/raman/aster-thesis.ps.

[10] Raman, T. V., and Gries, D. Documents mean more than just paper! *Proceedings of the 2nd. International Workshop on the Principles of Document Processing* (Apr 1994).

[11] Resnick, P. *HyperVoice: Groupware by Telephone.* PhD thesis, MIT, 1992.

[12] Resnick, P., and Virzi, R. A. Skip and scan: Cleaning up telephone interfaces. *Proceedings of CHI 1992* (1992).

[13] Steele, G. L. *Common Lisp The Language*, second ed. Digital Press, Bedford, Mass, 1990.

[14] Virzi, R. A., Resnick, P., and Ottens, D. Skip and scan telephone menus: User performance as a function of experience. *Human Factors Society, HFS 1992* (1992).

[15] X3J13, A. S. C. *Programming Language — Common Lisp— Draft Proposed.* CBEMA, 1993. *URL* FTP://parcftp.xerox.com/pub/cl/dpANS2.