A system for automatic structure discovery and reasoning-based navigation of the web

E. Pontelli\(a,*,b\), T.C. Son\(a\), K. Kottapally\(a\), C. Ngo\(a\), R. Reddy\(a\), D. Gillan\(b\)

\(a\)Department of Computer Science, New Mexico State University, Box 30001, Las Cruces, NM 88003, USA
\(b\)Department of Psychology, New Mexico State University, Box 30001, Las Cruces, NM 88003, USA

Available online 6 May 2004

Abstract

In this paper, we highlight the main research directions currently pursued by the investigators for the development of new tools to improve Web accessibility for users with visual disabilities. The overall principle is to create intelligent software agents used to assist visually impaired individuals in accessing complex on-line data organizations (e.g. tables, frame structures) in a meaningful way. Accessibility agents make use of knowledge representation structures (automatically or manually derived) to assist users in developing navigation plans; these are employed to locate given pieces of information or to answer user’s desired goals.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Web accessibility; Agents; Table navigation

1. Introduction

The problem of providing effective multi-modal access to complex on-line bodies of knowledge has quickly become of central importance, providing challenging research problems in the field of Universal Accessibility. The basic premise of universal accessibility of the Internet is the presence of alternative access modalities, to fit the needs of users with different backgrounds, learning styles, abilities and disabilities. Research in this field requires innovative integration of technologies from cognitive sciences, multimedia, and networking. In this project we are particularly concerned with providing the dimension of universal accessibility of the Web to suit the needs of visually

* Corresponding author. Fax: +1-505-646-1002.
E-mail addresses: epontelli@cs.nmsu.edu (E. Pontelli), tson@cs.nmsu.edu (T.C. Son), kkottapa@cs.nmsu.edu (K. Kottapally), cngo@cs.nmsu.edu (C. Ngo), treddy@cs.nmsu.edu (R. Reddy), gillan@crl.nmsu.edu (D. Gillan).
impaired individuals. This group of individuals is becoming significantly large—with increased life expectancy, more senior citizens are becoming part of this segment of the US population (NFB 2000).

The problem is of particular importance, as most of the existing features used to facilitate access to Web documents rely on graphical and multidimensional presentation formats, deepening the divide between visually enabled and visually disabled users. Although our focus is on visual disabilities, the methodologies we propose are fairly general and can be used in other contexts where visual access is infeasible, e.g. when using access devices with limited screen capabilities. In all these situations, textual and graphic presentations are ineffective, while aural (i.e. speech-based) presentations are the only feasible solution. For example, our methodologies can be used to develop question-answering systems with limited natural language processing capability that accesses a website and provides users with the content of the website. A concrete system that can be developed using our methodologies could be a travelers’ assistant: a system that accesses mapquest.com (or any of the websites that provide driving direction) and assists drivers in finding their way to the destination. In this system, the users only need to provide the current location and the destination which calls for very limited natural language processing capability is needed. The system then accesses the website to get the driving direction and responds to the request with this information.

There is evidence that visually impaired users are unable to complete many practical tasks associated with accessing a majority of existing Web sites, such as navigating tables (Gunderson et al. 1997) and completing forms (Kennel et al., 1996; Earl et al., 1999). Efficiency of access and interpretation of non-linear structures (e.g. tables, frames) is directly dependent on knowledge of the spatial layout of the components of the structure. Spatial layout is inherently visual and its absence (or an arbitrary linearization of the structure) deprives users of contextual references required to make documents interpretation feasible. In a variety of situations, the visual layout is actually used to emphasize the existence of a more complex structuring of the information within a document—e.g., the spatial layout can be used to describe a 3D structure or a tree structure (e.g. combining a table structure with the use of colors, header elements, etc.).

Furthermore, user studies have demonstrated that straightforward approaches aimed at replacing thorough knowledge of the spatial layout with other reference mechanisms (e.g. sound cues) are largely ineffective—and they may even lead to a decrease in access and retrieval performance (Pontelli et al., 2002).

In this project we develop a methodology to provide visually impaired users with explicit representation of layouts and ‘navigational’ semantics, along with techniques to assist users in using such representations. Our technology provides a novel framework for the development of software agents that can assist visually impaired individuals in navigating complex web pages.

1.1. Philosophy of our approach

Investigating human interactions with complex on-line documents might begin with two issues—what are the functions and conditions under which people use complex documents? At the highest level, people are clearly interested in interacting with
the information contained in a document. In one functional mode, which we will call *question-answering*, people want to access a limited amount of the information for an immediate need, e.g. to determine the arrival time for a flight or to compare the performance of companies in the stock market. The questions that users want to answer vary in complexity from a single value to a comparison among a set of objects to a calculation using tabled values. Typically, question-answering uses of documents involve brief access to the information with no intention on the user’s part for any long-term retention of the information. In contrast, in a second functional mode of interaction, *knowledge-acquisition*, users want to commit the information from a document to long-term memory so that they can later access the information in the absence of the physical document.

The perceptual and cognitive processes that a document reader uses to find and process information (including, if necessary, to learn it) depend on the functions and conditions of use, e.g. search of a table will be directed towards a specific goal in the question-answering mode, whereas, it will be more diffuse in the knowledge-based mode. The mechanism of navigation of a table will be primarily by visual search if a sighted user is reading an entire table presented on a large computer display; in contrast, a user reading from a limited display (e.g. cell phone), will use vision to read an item but manual means (e.g. arrow keys) to navigate the information space and move the desired information into the display space. A visually impaired user would likely use a combination of auditory and manual (or voice) control to navigate the information. In addition, the configural cues provided by the entire document would be lost in a limited display. These changes increase the user’s workload, cognitive effort, time to complete tasks, and load on working memory, reducing usability of the document and information accessibility.

The work described represents the natural continuation of the efforts described in (Pontelli et al., 2002), where a similar conceptual representation of the navigational structure of a document is employed to support the knowledge-acquisition mode of interaction. Our previous proposal relied on an explicit representation of the navigational structure of the main components of a document, along with a collection of mechanisms to allow the user to interactively navigate such structure and produce aural output.

In this work we propose to tackle the more complex question-answering mode of operation. In this case, the ability to acquire knowledge through user-directed navigation is insufficient—since the objective of the user is to *reason* about the content of the document. We accomplish this by providing the user with a reasoning component (in the form of a software agent) and a mean for the user to express queries on the document’s content. The software agent interacts with the user and the navigational semantics of the document to automatically detect the best answers to the user’s goals.

### 1.2. Related work

A number of proposals have recently been made towards the development of tools to improve Web accessibility for the visually impaired users. Some of the noteworthy proposals are the following. The WAB effort at ETH (Kennel et al., 1996) represents one of the first efforts in this area where transcoding proxy servers are employed to extend HTML documents with (static) navigation information (in this case, links to facilitate
the retrieval of titles and hyperlinks). Substantial effort has been invested in IBM’s
HomePage Reader (HPR) (Asakawa and Itoh, 1998) and the related work by Asakawa et al.
(Asakawa and Itoh, 1998; Oogane and Asakawa, 1998). This is indeed the only other
proposal that explicitly deals with the issue of table navigation. Like HPR, the
pwWebSpeak browser (De Witt et al., 1998) also offers speech output through HTML
interpretation. The current reported version of pwWebSpeak does not support frames; it
represents tables by linear sequences of links (each table location is interpreted as a
separate page); tables are sequentially navigated left-right, top-bottom without any special
announcement. The BrookesTalk (Zajicek, 1999) speech-enabled browser provides access
to different parts of a Web page using keys; its novelty is in the use of summarization
techniques to facilitate non-visual navigation.

In this project our aim is to provide semantic specification for the understanding and
navigation of composite HTML structure (forms, frames, tables). Relatively few other
proposals have tackled this problem and most of them rely exclusively on the purely
syntactic HTML component to provide navigation support. The need for adaptation of
browsers and for the direct access to HTML for accessibility support has been raised by
various authors (Brewer et al. 1998; Hendrix et al., 1998). This was further underscored by
recent survey studies (Earl et al., 1999; Gunderson et al. 1997) which explicitly pointed out
the ineffective performance of existing screen readers when coupled with the Web. Gunderson and Mendelson specifically cite the task of locating information in table
structures as one of the most challenging tasks for visually impaired users (Gunderson et al.
1997). Recently, Filepp et al. (2002) provided a static mechanism to explicitly formalize
the navigational structure for HTML tables.

2. Overall system structure

Fig. 1 shows the overall structure of the Accessibility Agent proposed in this project.
The agent is composed of three subsystems: the interface subsystem, the document
analysis subsystem, and the navigation/reasoning subsystem.

The accessibility agent interacts with the environment through two interfaces. The
agent seamlessly interacts with a standard Web browser (Microsoft IE in the current
prototype), filtering each individual HTTP request generated by the browser. The filtering
is realized through a Web Intermediary (Barrett and Maglio, 1999). Each incoming
document is displayed by the browser as well as it is transmitted to the Document Analysis
component of the agent. The agent explicitly interacts with the user, during navigation,
through a natural language interface—that allows the user to express navigation tasks and
goals—and through a speech synthesizer—that allows the agent to provide an aural
representation of the results of the navigation process. Both interfaces (the Web
intermediary and the user interface) are developed using off-the-shelf software
components and will not be discussed any further in this paper.

The Document Analysis component of the agent is aimed at extracting from the
document the knowledge necessary to allow the agent to assist the users in navigating the
document. The document model is explicitly encoded using knowledge representation
structures—specifically conceptual graphs (CG)—and it is employed by the reasoning
component of the agent to develop plans to answer the navigation goals and tasks expressed by the user. CG are derived using syntactic and semantic analyzers (described in Section 3); these can be locally cached in the agent and reused in the future—either directly or as templates for the analysis of related documents.

The Navigation and Reasoning components are in charge of constructing navigation plans to answer the goals and tasks provided by the user. The plans are automatically constructed using reasoning and planning algorithms, applied to the conceptual structures constructed by the document analyzer. In particular, action theories are employed to formalize the conceptual structure and the navigation capabilities of the agent, and logic-based planners are employed to reason on this formal representation and develop the navigation plans. The execution of each navigation plan will guide the user in traversing the document and will produce an aural output in response to each navigation step. The agent is also used to monitor the execution of the plan, thus providing recovery capabilities in case of navigation failure (e.g. if the document is dynamic and changes during the navigation process) and in case of additional interactions with the user (e.g. the user provides additional information during the navigation process). In both cases, the failure will lead to the reconstruction of a new navigation plan.

3. Knowledge representation

The core of the accessibility agent is composed of a reasoning engine, capable of developing navigation plans to address queries submitted by the user. The reasoning process is based on an explicit representation of the navigational semantics of
the incoming documents. The navigational semantics is aimed at explicitly representing the semantic components of the documents along with the relationships between such components. The navigational semantics provides a backbone to guide the user while (interactively) navigating a document-as successfully illustrated in our previous work (Pontelli et al., 2002). Our previous proposal relied on reducing the navigation of a document to an interactive traversal of the conceptual graph that captures the navigational semantics of the document. This approach has been shown (Pontelli et al., 2002) to be highly effective when the visually impaired user has little or no knowledge about the content of the document. On the other hand, forcing the user to interactively navigate the document is impractical in those situations where the user has previous knowledge about the content of the document and he/she is interested in answering specific questions about the document (navigation goals). In this work we demonstrate that the same conceptual structure adopted in our previous effort provides an adequate knowledge representation backbone for a software agent to reason about the content of documents on behalf of the user, and automatically solve navigation goals.

3.1. Conceptual structures

The knowledge representation scheme that we propose to adopt in this project is based on CG (Sowa, 1984). CG and their associated theory, Conceptual Structure Theory, were proposed in the 1970s as a way of drawing logical statements in diagrammatic form rather than in a linear text-based calculus which was and is the norm. A CG can have two kinds of node; a concept node that represents types and objects of that type, and a relation node that represents a relationship between these objects. The theory allows for a basic expressiveness that is equivalent to first-order logic as well as mechanisms for defining concepts and for representing type hierarchies of concepts and relations. Researchers have extended the formalism to allow representation and manipulation of more advanced ontologies, especially those involving actions and events, and higher-level structures such as viewpoints, and nested contexts. Documents are analyzed to automatically create the overall CG for the document; various parts of the CG are detected through syntactic analysis of HTML or through document partitioning techniques.

Document components are classified according to their role in the document (e.g. links, tables, paragraphs)—see Fig. 2. The semantics of each component is encoded as a separate CG. For example, text partitioning techniques are employed to create a conceptual node for each paragraph in the text; ontology keyword occurrences (derived from headers in the text) are used to derive relation nodes connecting paragraphs. The technique is analogous to those used for natural language summarization.

In addition to the description of the navigational semantics of each individual document component, and the classification of the components based on their type, our system (Fig. 2) imposes a global hierarchical structure over these descriptions. The top level of the hierarchy is derived from the frame structure of the document (if any). The description of each frame is partitioned in its components, each classified according to its type and described by a separate conceptual structure.
3.2. Multiple document views

The conceptual structure organization adopted provides alternative views of a document. These are used to provide alternative modalities of navigation (e.g., according to different document components); for example, document headers provide quick summaries of the content of a document; document links provide a summary of the connection structure of the document. The multiple components are connected by relation links, providing the ability to dynamically switch modalities of navigation as well as providing a powerful mechanism for contextual reinforcement, e.g., from each table it is possible to locate the meaningful paragraphs that refer to it; from each paragraph it is possible to locate the corresponding header; from each header it is possible to locate the collection of hyperlinks embedded in the paragraph having such header; etc. A reference locator structure provides a broker system for the management of the relations between document components.

The idea of multiple views extends to the level of the individual document components, allowing us to create alternative viewpoints (Ribiere and Dieng, 1997). For example, the representation of a two-dimensional table typically requires at least two distinct viewpoints—one that represents the table as a collection of rows and another that represents it as a collection of columns. Fig. 3 illustrates this point: the table on the left side
is represented by the graph on the right side. The table contains information about lodging and meals expenses to two cities NY and LA. The data rows represent the information regarding each trip and the columns contain data relating to each category. Cells of the table correspond to nodes in the graph. Two special nodes, Table and gray area are added which represent the table and the special row, respectively. The connections between the nodes represent the relationship between the cells in the table. We note that in the graph representation of the table, the distinction between rows and columns disappears. This provides us the flexibility to present the information in different views. Furthermore, it is useful to allow the user to dynamically add new viewpoints—representing intermediate summaries during the navigation process. The importance and the advantages offered by conceptual structures and viewpoints have been discussed more extensively in Pontelli et al. (2002).

3.3. Tables and frames

Research in the field of non-visual web accessibility has repeatedly highlighted the difficulty in providing effective ways to present document components that have an inherently multi-dimensional structure (tables and frames). In our context, this difficulty translates to the complex process of recognizing the structure of a table or a frame. Ideally, each cell of a table belongs to one or more semantically meaningful aggregations of cells; navigation should be realized by allowing the user to recognize the presence of such aggregations and the relations between cells and aggregations of cells (Fig. 3). In a visual setting, the visual layout of the table is employed to allow the user to recognize the semantic structure of the table. In a non-visual setting such structure has to be made explicit during the navigation process. CGs represent again an ideal notation to express this knowledge.

4. Automatic format discovery

In the previous discussion we have highlighted the importance of developing an appropriate knowledge representation used to guide the navigation process. In our previous efforts (Pontelli et al., 2002) we focused either on manual construction—in Pontelli et al. (2002) we describe a GUI for the semantic annotation of documents—or on the use of purely syntactic analysis of HTML documents. In this paper we report our recent efforts towards the development of methodologies for the automatic discovery of structure in complex HTML components (tables and frames). The ultimate objective is to develop software tools that automatically recognize the headers of an arbitrary HTML table or the index structure of an HTML frame structure.

Our effort makes use of two machine learning (ML) procedures used to detect headers in HTML tables and indices in frame structures. ML is an artificial intelligence methodology which makes use of a collection of examples and some properly formalized background knowledge about a specific domain to derive general rules that can explain the given examples. The derived rules can then be used to analyze new observations and
properly classify them. In our context, we intend to employ ML techniques to derive general rules that allows the automatic identification of the structure of tables and frames.

Knowledge about headers/indices provides a natural skeleton for the CG for a table/frame structure—headers denote semantically relevant grouping of cells and relationships between cells. Progol, an inductive logic programming (ILP) system and Hidden Markov Modeling (HMM), a statistical ML technique, are presented as effective methods to tackle these problems.

4.1. Inductive logic programming techniques

4.1.1. Overview of Progol

Progol (Roberts, 1997) is a ML procedure using ILP. ILP systems develop a declarative knowledge to produce general rules from a training set and background knowledge. Progol requires the background knowledge and examples to be presented in the form of Prolog facts and rules. Prolog is a popular logic programming language, where knowledge is represented as facts (i.e. equivalent to tuples in a relational database) and as rules (equivalent to logical implications). Rules are of the type

\[
\text{Head} : \neg B_1, \ldots, B_n
\]

and they can be read as follows: if \( B_1, \ldots, B_n \) are known to be true, then Head can be considered true as well. Facts are expressed in the format

\[
\text{property(attribute}_1, \ldots, \text{attribute}_n)
\]

stating that the tuple (\( \text{attribute}_1, \ldots, \text{attribute}_n \)) satisfy the property \( \text{property} \).

The output, which is a generalized rule, is also expressed in Prolog syntax. The user must specify the intended structure of the general rule, i.e., the structure of its head and body. This information is given to the system in the form of mode declarations. Mode declarations restrict the predicates which can occur in the head and body of the generalized rules. Progol determines generalized rule as follows. It first constructs a specific rule that explains the first available example in the training set. It then combines the predicates in the most specific clause to make new rules. Each of these new rules is tested against the given examples to determine how many they explain. The rule which explains most of the examples is declared as the generalized rule (Roberts, 1997).

4.1.2. Detecting table and frame structure

The objective is to obtain the most general rule to determine a header in a table and the location of an index frame in a frame structure. There are two steps before we make Progol learn about headers. The first step is to transform a HTML table (or frame structure) into a collection of logical facts. The second step involves providing some basic rules to determine the properties of different types of table cells.

4.1.3. Preparing background knowledge

Step 1: Transforming a table into facts: A table in HTML is transformed into a set of facts in Prolog where each fact represents a cell in the table. The properties and information about each cell are given as arguments to the facts. The prototype of
the fact is:

\[
\text{cell(tablename, cellnumber, rownumber, colnumber, empty?)}. 
\]

The arguments are self-explanatory. They are used in the training process to distinguish data cells from headers. A cell spanning multiple rows or columns is divided into multiple cells with the same cell number.

\textit{Example:} The following is the table Scitech2; below the table we provide part of the encoding in Progol (Table 1).

\textit{Facts:}

\[
\begin{align*}
\text{cell(scitech2, 1, 0, 0, false).} & \quad \text{cell(scitech2, 1, 0, 1, false).} \\
\text{cell(scitech2, 1, 0, 2, false).} & \quad \text{cell(scitech2, 1, 0, 3, false).} \\
\text{cell(scitech2, 1, 0, 4, false).} & \quad \text{cell(scitech2, 1, 0, 5, false).}
\end{align*}
\]

\textit{Step 2: Providing background rules:} Progol needs to determine the properties of the individual cells in the training process to obtain knowledge about headers. Hence, Progol is provided with some rules to determine the properties and attributes of cells such as spanning, emptiness, position (row, col), etc. Following are a couple of examples,

\[
\begin{align*}
\text{cell(Tablename, Cellno, \_ \_ \_, false) \rightarrow nonempty(Tablename, Cellno)} \\
\text{cell(Tablename, Cellno, 0, \_ \_ \_) \rightarrow firstrow(Tablename, Cellno)}
\end{align*}
\]

The facts produced in the first step, combined with the rules about cell properties in the second step, comprise the background knowledge. With this background knowledge and the positive training examples, Progol can be trained to recognize table headers—similar steps have been adopted to describe index frames in a frame structure (details omitted due to lack of space).

\subsection{4.1.4. Training the system}

The system was trained with a training set composed of 10 tables obtained from different web sites, each having a unique structure in terms of headers. These tables have been transformed into logical facts as discussed earlier. The positive examples in the training set are derived by a description of the location of the headers in each of these tables (provided manually). These headers are given as logical facts. For example, following are the positive examples for the table given in Section 4.1.3.
The system is trained with these facts combined with positive examples and background knowledge.

4.2. Hidden Markov model techniques

A Hidden Markov Model (HMM) is a finite set of states, where each state is associated with a probability distribution. State transitions are governed by transition probabilities and the outcome at a particular state depends on the associated probability distribution. Each state represents a class of strings. The idea is to use the model to properly classify inputs—represented as sequences of symbols—by following the most probable transition from state to state.

In a HMM, the state sequence that produces a sequence is hidden and hence the name. Different state sequences can produce the same sequence of symbols. Even though the state sequence is hidden, it is possible to calculate the probability of states using emission and transition probabilities. In a HMM, a transition probability \( a_{ij} \) is the probability of switching from the state \( s \) to the state \( t \). The transition probability \( e_p(x_i) \) is the probability of producing symbol \( x_i \) while in state \( p \). The joint probability of an observed sequence can be written as follows.

\[
P(\bar{x}, \pi) = a_{0, \pi} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}
\]

Two algorithms—forward and backward—are used to train the HMM (Gusfield, 1997). Since the state sequence is hidden, it is not possible to determine the exact state path for a sequence but we can calculate the most probable path using the Viterbi algorithm (Gusfield, 1997).

4.2.1. Using HMM for table structure detection

HMM is experimented with two types of observation sequences: unlabeled and labeled. HMM trained with Labeled data outperforms its unlabeled counterpart as the former method gives more clues about the structure of training data. Because obtaining labeled training data is not difficult in this case (tables), the method proves to be both efficient and easy to implement. The training data is a file containing a sequence of words where each word reflects a predominant property of the corresponding cell. These properties are chosen such that they help the learning system in differentiating a header from a cell. Eight cell properties are considered and each cell is represented with one of these properties. The properties are listed below:

- \( \text{hspan} \)—the cell spans two or more columns
- \( \text{vspan} \)—the cell spans two or more rows
The implementation of HMM algorithms is obtained from Dr Andrew McCallum (Seymore et al., 1999). For example, for Table 2 we will have the following sequence of unlabeled observations:

Unlabeled observation sequence for Table 2

```
hspan empty urspan urspan urspan col1 ee ee ee ee ee
```

and the corresponding labeled observations are:

Labeled observation sequence for this table

```
hspan header empty cell urspan header urspan header urspan header col1 header ee cell ee cell ee cell col1 header ee cell ee cell ee cell col1 header ee cell ee cell ee cell col1 header ee cell ee cell ee cell col1 header ee cell ee cell ee cell col1 header ee cell ee cell ee cell
```

4.2.2. Creating and training the HMM

Since each state in the HMM represents the class a cell belongs to, the number of states in the trained model depends on the number of classifications we need to make of the cells. Since we are only concerned about separating headers from cells in a given table, only two states are needed: one to represent header and other to represent cell. These two states combined with start and end states form the HMM.

The initial model is created using Bayesian Model Merging (BMM) with a selected set of tables as input (8 tables are used). This model forms the basis for further training. The model is then trained with a variety of tables using forward and backward algorithms. Fig. 4 shows the model after training with 25 tables.

<table>
<thead>
<tr>
<th></th>
<th>Unstandardized coeff</th>
<th>Standard error</th>
<th>Statistical significance (t value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>State population</td>
<td>0.000000046</td>
<td>0.00</td>
<td>3.7*</td>
</tr>
<tr>
<td>% Liberal</td>
<td>-0.614</td>
<td>0.26</td>
<td>-2.4*</td>
</tr>
<tr>
<td>% Sr. citizens</td>
<td>0.59</td>
<td>0.43</td>
<td>1.4</td>
</tr>
<tr>
<td>State spending</td>
<td>0.00014</td>
<td>0.001</td>
<td>1.1</td>
</tr>
<tr>
<td>% College grad</td>
<td>0.10</td>
<td>0.20</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Table 2
Regression model of state ranks

The implementation of HMM algorithms is obtained from Dr Andrew McCallum (Seymore et al., 1999). For example, for Table 2 we will have the following sequence of unlabeled observations:

Unlabeled observation sequence for Table 2

```
hspan empty urspan urspan urspan col1 ee ee ee ee ee
ee e e e e e e e e e e e e e e e e e
ee e e e e e e e e e e e e e e e e e
```

and the corresponding labeled observations are:

Labeled observation sequence for this table

```
hspan header empty cell urspan header urspan header urspan header col1 header ee cell ee cell ee cell col1 header ee cell ee cell ee cell col1 header ee cell ee cell ee cell col1 header ee cell ee cell ee cell col1 header ee cell ee cell ee cell col1 header ee cell ee cell ee cell
```

4.2.2. Creating and training the HMM

Since each state in the HMM represents the class a cell belongs to, the number of states in the trained model depends on the number of classifications we need to make of the cells. Since we are only concerned about separating headers from cells in a given table, only two states are needed: one to represent header and other to represent cell. These two states combined with start and end states form the HMM.

The initial model is created using Bayesian Model Merging (BMM) with a selected set of tables as input (8 tables are used). This model forms the basis for further training. The model is then trained with a variety of tables using forward and backward algorithms. Fig. 4 shows the model after training with 25 tables.
4.3. Evaluation

4.3.1. Inductive logic programming methodology

Fig. 5 reports the results obtained from recognizing the headers structure of 50 different tables, randomly collected from a variety of distinct web sites. The original training set is composed of 10 tables (distinct from the 50 reported in the table). Fig. 5 compares the number of headers in the original table (blue) and the number of headers discovered (red).

4.3.2. Hidden Markov model methodology

Using the Viterbi algorithm, the most likely state sequence (sequence of headers and cells) for a given observation sequence (table) is determined. Following is a sample table
tested against the trained model. The input (observation sequence) for Table 3 and the output (most likely state sequence) are also listed.

Observation sequence (input)

```plaintext
hspan urspan urspan urspan urspan empty urspan urspan urspan urspan urspan urspan urspan urspan col1 ee ee ee ee ee ee ee ee ee ee ee ee col1 ee ee ee ee ee ee ee ee ee ee col1 ee ee ee ee
```

Output (all the cells are tagged with their class)

```plaintext
hspan header urspan header urspan header urspan header urspan header urspan header urspan header urspan header col1 header ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell ee cell (null) end
```

The outcome of analyzing the same set of tables as in the case of ILP is reported in Fig. 6.

### 4.3.3. Discussion

Progol successfully detects all headers in most of the tables. However, it fails to detect in some cases. This can be attributed to the following reasons: (i) these headers are not common in most of the tables. Hence the system fails to include these types of headers in the generalized rule; (ii) these headers are difficult to distinguish from other cells using the given cell properties.

Providing more information about a cell such as color, and font size of its text may result in more accurate prediction of headers. Also, varying the background knowledge greatly affects the generalized rule.

The performance tables for Progol trained model and HMM suggest that HMM yields better results than Progol in some cases. For tables with arbitrary headers and cells, combining the results from both the models will help detecting most of the headers. The performance of the HMM can be improved by adding more properties of table cells to the training data. Currently the HMM trained with 25 tables proves to be good enough for

---

Table 3
Steps libraries take to make up for inadequate journals funding

<table>
<thead>
<tr>
<th>Faculty</th>
<th>Health sciences</th>
<th>Humanities</th>
<th>Science-Engng</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel lesser used</td>
<td>52.0</td>
<td>29.3</td>
<td>35.7</td>
</tr>
<tr>
<td>Cancel high cost</td>
<td>34.5</td>
<td>33.3</td>
<td>46.1</td>
</tr>
<tr>
<td>Get articles on demand</td>
<td>43.1</td>
<td>32.5</td>
<td>51.9</td>
</tr>
<tr>
<td>Get electronic only</td>
<td>63.2</td>
<td>24.0</td>
<td>32.1</td>
</tr>
</tbody>
</table>
most of the tables with an expected structure of headers and cells. Fig. 7 compares the accuracy of recognition for the 50 tables using the two methods.

Using these techniques we can effectively process complex tables and frame structures and automatically derive an adequate representation of the semantic structure of such document components. The structure (encoded as a CG) will then be made available to the user to guide the navigation of the document.

5. Navigation as reasoning

The previous sections discussed the techniques that we used in constructing the document structure and creating a conceptual graph for navigation. The conceptual structure is aimed at providing a base guideline that a visually impaired user can follow to acquire knowledge about the content of the document; thus, both functional modes of use of a document (Section 1.1) can be reduced to traversals of appropriate paths within the conceptual structure. In knowledge-acquisition mode, the user is effectively in charge to dynamically (and arbitrarily) select the paths to be explored. On the other hand, in question-answering mode, the user selects the paths of interest according to specific goals he/she intends to accomplish. Given the ability to formalize the user’s goals in terms of properties of nodes and links in the conceptual structure, it becomes possible to view this process as a planning and execution monitoring process. The system can assist the user by (i) constructing the sequence of movements that the user needs to do in order to achieve
this goal; and (ii) executing the sequence of movements (read aloud to the user the content of the nodes along the path). In this section, we will show how these features can be supported by a planning module.

We will employ an action theory framework (Gelfond and Lifschitz, 1998) in creating our planner. The action theory framework is a formal logic-based notation to describe the capabilities of an agent, in terms of properties of the world and actions that the agent can perform to affect the state of the world. The action theory framework has the advantage of allowing developers to describe the agent’s capabilities in a formal and high-level fashion. In turn, algorithms have been developed that, given an action theory description and a goal (which describes what kind of properties the world should have as result of the agent’s activities), produce a plan, i.e. a sequence of actions that, when executed, will transform the world in such a way to satisfy the given goal. Planning algorithms are at the base of the reasoning components of intelligent software agents.

In this framework, the results of the navigation process will be described by a set of properties describing the environment, called fluents. The commands that allow the user to navigate between nodes of the structure are called actions. Actions can be divided into two types, basic actions and document-specific actions. The former is similar to that of moving from node to node—one step at a time—thus providing the basic navigation capabilities to users. Observe that the structure of the basic actions is document independent. The latter is, in many cases, a careful developed procedure that helps users to understand specific parts of a document without having to navigate through the whole document. In the rest of
this section we illustrate what action theories are and how they can be employed to provide the reasoning backbone of the question-answering agent. Observe that the action theories provide only the implementation backbone of the agent—the actual interaction with the user will take place via a high-level natural language query language (currently under development).

5.1. Action theories

We adopt the high-level action description language $B$ of (Gelfond and Lifschitz, 1998) to represent action theories. In this language, the environment is described by a set of propositions, called fluents, whose truth values changed as effects of actions. An action theory consists of two finite, disjoint sets of names $A$ and $F$, called actions and fluents, respectively, and a set of propositions of the following form:

\[
\begin{align*}
\text{caused} & : \{p_1, \ldots, p_n\}, f \\
\text{causes} & : (a, f, \{p_1, \ldots, p_n\}) \\
\text{executable} & : (a, \{p_1, \ldots, p_n\}) \\
\text{initially} & : f
\end{align*}
\]

where $f$ and $p_i$’s are fluent literals (a fluent literal is either a fluent $g$ or its negation $\neg g$) and $a$ is an action. A proposition of the form (1), or static causal law (also causal law, for short), states a constraint among the fluents of the domain. It states that whenever the fluent literals $p_1, \ldots, p_n$ hold then $f$ must hold as well. A proposition of the form (2), referred to as a dynamic causal law, represents the (conditional) effect of $a$. It says that $f$ is guaranteed to be true after the execution of $a$ in any state of the world where $p_1, \ldots, p_n$ are true. An executability condition of $a$ (a proposition of the form (3)) says that $a$ is executable in a state in which $p_1, \ldots, p_n$ hold. Propositions of the form (4) are used to describe the initial state. It states that $f$ holds in the initial state. We illustrate the important features of the action theory framework through examples.

Consider a website with different HTML elements such as frames, tables, texts, and pictures. The website will be converted into a conceptual structure, whose nodes and links represent the different elements of the document and the connections between them, respectively. Internally, each node is assigned a unique identifier and a label, that contains the information associated with the node. The logical connections between nodes are encoded as relation nodes. The beginning of a document is represented by a node with the identifier 0 and the label “Root”. This node is connected to the nodes representing the elements that belong to the highest level of the document (e.g. the elements composing the frame panes in a frame-structure document).

Assume that $Id_1$ and $Id_2$ are the two elements of the website which are connected by a relation $R$—e.g. $Id_1$ is cell in a table and $Id_2$ is one of the table headers associated to such cell. The connection between the two nodes $Id_1$ and $Id_2$ is represented by the fluent $\text{connected}(Id_1, R, Id_2)$, e.g. $\text{connected}(TableX, \text{belongs}, \text{HeaderY})$ denotes that the node representing $TableX$ is connected via the relation $\text{belongs}$ to the node for $\text{HeadeY}$
(i.e. the table X is part of the document section with header Y). To record that the user is currently at node Id we write \( at(Id) \).

The basic action that a user can perform in this theory is to move from one node to another node. This is encoded by the action \( move(Id_1, Id_2) \). The effect of moving from \( Id_1 \) to \( Id_2 \) is that the user is at the node \( Id_2 \) and no longer at the node \( Id_1 \). This is expressed by the propositions (for any pair \( Id_1 \neq Id_2 \))

\[
move(Id_1, Id_2) \text{ causes } at(Id_2)
\]

and

\[
\text{if } at(Id_2) \text{ then not at}(Id_1)
\]

To express that the user can move from \( Id_1 \) to \( Id_2 \) only if she/he is at the node \( Id_1 \) and the two nodes are connected, we write

\[
move(Id_1, Id_2) \text{ executable if } connected(Id_1, R, Id_2) \text{ and } at(Id_1)
\]

Finally, to say that the user is at the node designated by 0 initially we write

\[
\text{initially } at(0).
\]

Given an action theory, a transition function \( \Phi \) that maps each pair of an action \( a \) and a state \( s \) into a set of states \( \Phi(a, s) \) can be computed and used for solving the planning problem. Intuitively, \( \Phi(a, s) \) is the set of states that are reachable from \( s \) after executing the action \( a \) in state \( s \), and a state represents the knowledge about the conceptual structure that has been collected up to that point of the navigation. \( \Phi(a, s) \) is an empty set if action \( a \) is not executable in \( s \). The precise definition of \( \Phi(a, s) \) can be found in Gelfond and Lifschitz (1998) and Son et al. (2001).

5.2. Conceptual structures as action theories

Document structures (stored as CGs) discovered in the previous sections can be translated automatically into an action theory as follows. For each CG \( G \), with the set of identifiers \( ID \), let \( G_{\text{connected}} \) be the set of fluents of the form \( connected(Id_1, Relation, Id_2) \) such that \( Id_1 \) and \( Id_2 \) belong to \( ID \) and they are connected through the node Relation in the conceptual graph. Let \( G_{\text{actions}} \) be the set of propositions describing the effects of the actions \( move(Id_1, Id_2) \) for every pair of \( Id_1 \neq Id_2 \). Furthermore, let \( G_{\text{causal}} \) be the set of propositions expressing the fact that the user cannot be at two places at the same time, i.e.

\[
G_{\text{causal}} = \{ causes(at(Id_1), \neg at(Id_2)) | Id_1 \neq Id_2 \}\]

The action theory corresponding to \( G \) is then given by the pair \( (D, I) \) where \( D = G_{\text{actions}} \cup G_{\text{causal}} \) and the set \( I \) consists of the propositions

\[
\text{initially } connected(Id_1, Relation, Id_2),
\]

for each \( connected(Id_1, Relation, Id_2) \) \( \in G_{\text{connected}} \) and a proposition identifying the node where the navigation starts.
5.3. Navigation as planning and plan execution

The action theory and the logic program developed in the previous sections can be used to help users navigating the document in different ways, e.g.,

- Provide suggestions on what can be related to a given node or what needs to be done to reach a particular node.
- Execute a predefined sequence of movements to reach a node.

The first item is related to planning while the second one is related to plan execution. Before we elaborate on each of these options more in details we would like to reiterate that the navigation component of our system consists of two modules: a planner (the Reasoning component in Fig. 1) and a navigator (Fig. 1). The planner is responsible for generating the plan and the navigator will execute it. These two modules work in a loop:

while a goal exists

(Planner) find a plan to achieve the goal

(Navigator) execute the plan

The planner will receive the action theory \((D, I)\) and a goal—which is a request from the reasoning component—as inputs. The goal describes what kind of navigation objectives the user is requesting (e.g., locate information satisfying certain properties within the document). The planner will output a sequence of movements in the abstract version of the document for the navigator to execute (and movements will lead to aural output). The actual implementation of the planner is realized using logic programming technology (answer set planning (Lifschitz, 1999)). We will next elaborate on the planner module.

Let \((D, I)\) be an action theory and let \(a_0, \ldots, a_k\) be a sequence of actions. Let \(s_0\) be a state that satisfies \(I\)—i.e. it is a description of what the user knows about the document before starting the navigation process. We can check whether \(a_0\) is executable in \(s_0\) and, if so, what is the set of states reachable from \(s_0\) by \(a_0\) by computing the set \(\Phi(a_0, s_0)\). For a set of states \(S\), let \(\Phi(a, S) = \bigcup_{s \in S} \Phi(a, s)\). Let us denote

\[\Phi([a_0, \ldots, a_i], s_0) = \Phi(a_i, \Phi([a_0, \ldots, a_{i-1}], s_0)).\]

Clearly, if \(\Phi([a_0, \ldots, a_i], s_0)\) is not empty, then \(a_i\) is executable in \(\Phi([a_0, \ldots, a_{i-1}], s_0)\) for \(i = 0, \ldots, i\). Answering the question ’what properties hold true after the execution of \(a_0, \ldots, a_k\) in \(s_0\)’ can be answered by computing \(\Phi([a_0, \ldots, a_k], s_0)\). We solve this problem by developing a logic program \(\Pi(D, I)\) whose stable model semantics (Gelfond and Lifschitz, 1990) coincides with the transition function \(\Phi\) of \((D, I)\) (Son et al., 2001). We list below the most important features of \(\Pi(D, I)\):

- \(\Pi(D, I)\) has two sets of rules, one is independent on how \(D\) and \(I\) are defined and the other consists of the rules for encoding \(D\) and \(I\) in logic program terminologies.
- \(\Pi(D, I)\) has one parameter that represents the maximal number of steps our agent is willing to perform in order to achieve some goal. For this reason, we often write \(\Pi_n(D, I)\) instead of \(\Pi(D, I)\) where \(n\) is an integer number representing the maximal
number of steps. This is equivalent to say that we have a limit on the length of the plan needed to achieve some goal or a limit on the time for executing the plan.

- \( \Pi(D, I) \) uses atoms of the form \( \text{occ}(a, i) \) to represent the fact that action \( a \) occurs at the time moment \( i \). We call this an action occurrence. To represent the precondition that action \( a \) can occur only if it is executable, it uses atoms of the form \( \text{possible}(a, i) \).
- \( \Pi(D, I) \) uses atoms of the form \( \text{holds}(f, i) \) (resp. \( \text{holds}(-f, i) \)) to represent the fact that fluent \( f(-f) \) is true at the time moment \( i \).
- Stable models of \( \Pi(D, I) \) can be computed by using one of the efficient systems that can compute stable models of logic programs such as \texttt{dlv} and \texttt{smodels} (Citrigno et al., 1997; Niemela and Simons, 1997).

The full detail about the program \( \Pi(D, I) \) can be found in Son et al. (2001).

To compute \( \Phi(a_0, \ldots, a_i, s_0) \), we need to compute the stable models of the program

\[
\Pi_{i+1}(D, I) \cup \{ \text{occ}(a_i, t) : 0 \leq t \leq i \}.
\]

Whatever is true in the time moment \( i + 1 \) will represent a state reachable from \( s_0 \) by executing the sequence of actions \( [a_0, \ldots, a_i] \). If the program does not have a stable model, the sequence of actions cannot be executed from \( s_0 \).

To find a sequence of actions that changes the world from \( s_0 \) to \( s_n \) where \( s_n \) satisfies the property \( \phi \), we need to compute the stable models of the program

\[
\Pi_n(D, I) \cup \{ \leftarrow \text{not holds}(\phi, n) \}
\]

Each stable model gives us a trajectory \( a_0s_0\ldots a_{n-1}s_n \) such that \( \phi \) is true in \( s_n \). Thus, by using the proposed transformation of the action theory to logic programming it becomes possible to use existing logic programming systems to determine a sequence of actions that guides the user to the desired goal. The determination of the sequence of actions is completely automatic, thus relieving the user from the need of manually traversing the document to retrieve the desired pieces of information.

The code of the planner is available at our website.

The second component of the planner module is the goal representation language. We distinguish different types of goals: simple, complex, and directive. Directive goals are most appropriate in situations when users know nothing about the documents that they are trying to understand. Simple goals allow users to query the document for certain information. Complex goals are often a document-dependent procedures which encode possible ways a document can be presented to users. We will now present the language for goal representation.

**Directive goals.** A directive goal is of the form

\[ ?\text{possible\_move()} \text{ or } ?\text{reachable()} \]

that requests the list of all (neighboring or reachable) nodes of the current node. The system’s response will be the list of all possible single move the user can make or the list of possible nodes reachable from the current node. They represent the elements that are directly or indirectly connected to the current element. For example, a news website like www.cnn.com organizes news by topics and uses the table element in different
ways such as a table with the header ‘Home’ containing all the topics (which is fixed), a table called ‘more top stories’ (which changes whenever a new story arrives), a table named ‘top story’ of the moment with picture and text, a table named ‘business’ with stock information, etc.; if the user starts the navigation of the website and issues a directive goal, then the system will provide the list of the tables of the document (using possible_move) or the list of all the components of the document (using reachable). Selecting one from the list indicates that the user would like to explore that table. On the other hand, if the user is currently reading a stock number in the table ‘business’, the same goal will provide the list of elements that he can move to; this might include the rows/columns/cells that are related to the current cell, the next table, the previous table, the text that related to the row, the paragraph containing the table, or to the beginning of the table. This type of goals is more appropriate for exploratory tasks, where the user is interacting with the agent in refining and solving his/her goal. Observe that the plan to reach a location in the document automatically provides a trace of the connections between the current position and the destination (in terms of document components and relationships between them).

_simple goals._ A simple goal is a request for a path connecting two nodes $Id_1$ and $Id_2$. It is expressed by a query: $\text{?at}(id)$ where $id$ is the identifier of a node that the user wishes to visit. Basically, this query asks the planner for a path that allows the user to move from the current node to the node $id$. For instance, the user is currently looking at a row on ‘more top stories’ and he/she wants to go back to the main table ‘home’, all the user needs to do is to enter the request: $\text{?at}('home')$ where ‘home’ is the identifier of the table with all the topics, that the user already knows. Observe that the user must have some knowledge about the document to issue this request to the planner. The user can, however, get the list of identifiers that he could visit by issuing a directive goal of the form reachable before a simple goal. This is similar to reading the table of content of a book before reading a particular page.

_complex goals._ A complex goal can be a formula constructed from directive and simple goals using propositional and temporal logic connectives or a partially complete procedure. The propositional connectives are $\lor$, $\neg$, $\land$, $\rightarrow$, and the temporal connectives are always, next, until, and eventually, or if-then-else and while loop. Complex goals can be

- formulae describing the node that the user wants to visit,
- formulae describing the path that the user wants to take, or
- programs describing the navigation process.

The first type of complex goal is quite useful when the user is looking for a particular type of information. For example, he wants to get to a table in the document, he can write

$$\text{?at}(X) \land \text{type}(X, \text{table})$$

which says that the user wants to get to a node classified of type Table. We note that this type of goals requires the user to have a better understanding about the document she is navigating than the directive or simple goals. If the user is interested in breaking news
about Iraq, then the following goal will be sufficient:

\[ \text{at}(X) \land \text{type}(X, \text{row}) \land \text{connected}(X, \text{row_of}, \text{'more top stories'}) \land \text{content}(X, \text{'Iraq'}) \]

If the document contains a train timetable, then the goal

\[ \text{at}(X) \land \text{type}(X, \text{departure_time}) \land \text{connected}(X, \text{city}, Y) \land \text{content}(Y, \text{ElPaso}) \]

is appropriate to retrieve the departure times to El Paso.

The second type of complex goal uses the temporal connectives to specify the properties of the paths that the user wants to use in navigation. For example, the goal

\[ \text{always} (\text{at}(X) \land \text{type}(X, \text{table}) \rightarrow \text{next}(\text{at}(Y) \land \text{connected}(X, \text{header}, Y))) \]

says that whenever the user is at a node of type table, he/she wishes to be in the header row of the table after the next move. In other words, this goal limits the possible actions that the planner can use whenever the condition \( \text{at}(X) \land \text{type}(X, \text{table}) \) is satisfied. Goals of this type allow user to specify their preferences on paths that they want to follow when moving from one node to another. Not only this goal specifies a property of the path that the user wants to use for navigation, it also helps the planner to eliminate solutions that the user does not want to consider, thus improving the efficiency of the planner.

The third type of complex goal is the most sophisticated type of goals that a user can give to our system. Roughly, this goal is a program which can contain macros, tests, branches, and loops. For example, if the user already knows a procedure, called \( \text{navigate}(X) \), that allows him/her to navigate a frame, then he/she can use it to navigate a 2-frames web site by giving the goal

\[ \text{navigate}(1); \text{navigate}(2). \]

(‘;’ denotes the sequencing operator). For a website with more than 2 frames he/she can use the following goal

\[ \text{while} (\exists X \text{ connected}(X, \text{Root, frame}) \land \text{not visited}(X)) \text{ navigate}(X). \]

Complex goals allow powerful queries to be formed, including queries dealing with different types of aggregations (e.g. sum values present in given locations of the document).

5.4. Natural language interface

The approach to reasoning-based navigation described so far utilizes the expressiveness of first order and temporal logics. The goal language allows users to specify their navigation goals as well as the way to achieve these goals. Relying on logics and knowledge representation techniques makes the development of the navigation agents possible. Since first order logic and temporal logic might not be the language of choice of users trying to navigate web sites, we develop a natural language interface for goal specification. The interface is currently under development and employs a minimal set of vocabularies and an English-like syntax that allow users to describe the goals and the constraints for achieving the goals. The English description supplied by the users through this interface will be translated into an appropriate formula in the goal language and used as the goal of the navigation process. We will now describe the language interface.
For a directive goal, we employ sentences of the form What can I do? and the system will respond with the list of neighboring nodes of the current node;

For a simple goal, we use sentences of the type Visit node (node description) where a (node description) is a sequence of properties of the node described by

- ‘CONTAINING X’ (the node contains the string X);
- ‘NOT CONTAINING X’ (the node does not contain the string X);
- ‘OF THE TYPE X’ (the node of the type X); and
- ‘CONNECTED BY (Relation) TO (node description)’ (the node is connected to another node with the description specified by (node description)).

These properties are connected using ‘AND’ and ‘OR’. The sentence will be processed from left to right. As an example, the sentence “Visit node CONTAINING home AND NOT CONTAINING textbook” describes the goal at(X) where X is a node whose content contains the string home but does not contain the string textbook; The sentence ‘Visit node OF THE TYPE departure_time AND CONNECTED BY city TO a node CONTAINING El Paso’ will ask for a node of the type ‘departure_time’ which is connected to a node that contains ‘El Paso’ through a relation ‘city’;

For a complex goal, the natural language interface allows the use of temporal and control expressions that will be mapped to the corresponding logical constructs. Sample English structures that will be allowed to construct the goals include:

- ALWAYS DO X
- ‘NEXT DO X’
- ‘DO X UNTIL C’
- ‘DO X AND Y’
- ‘DO X OR Y’
- ‘DO X THEN Y’
- ‘ACHIEVE C BEFORE GOAL’
- ‘IF C THEN DO X ELSE DO Y’
- ‘WHILE C DO X’

where

- C is a node description or a node description preceded by one of the keywords ‘THERE IS’, ‘SUCH THAT’, ‘FOR ALL’; and
- X and Y are statements that can either be a sentence of the above form or a sentence describing a simple goal or its negation ‘Do not visit( node description)’.

6. Conclusions

In this paper we presented some key ideas employed in the development of software agents to assist visually impaired users in navigating complex Web pages. The system relies on an explicit encoding of the navigational structure of the
document, and on the use of planning technology to assist users in simple and complex query-answering tasks. Discovery of the structure of the document has been significantly improved through the use of ML techniques. We have successfully completed the integration of a planner that allows users to give directive and simple goals as well as the first type of complex goals. The planner is integrated in an infrastructure (Pontelli et al., 2002) that automatically captures user requests, analyzes incoming documents, and interacts with the user via keyboard input and aural output. We have completed the implementation of the planner that can take complex goals of other types and determine the plans for solving them. We are currently working on the integration of this planner into the system. We are also working on the development of a natural language processing interface to the software agent for visually-impaired individuals.

Acknowledgements

The research has been partially supported by NSF grants EIA-0130887, CCR-9875279, HRD-9906130, EIA-0220590, and EIA-9810732.

References

Filepp R., et al., 2002. Improving the accessibility of aurally rendered HTML tables. ASSETS, ACM.


