A Domain Specific Language Framework for Non-Visual Browsing of Complex HTML Structures

E. Pontelli, W. Xiong
Dept. Computer Science
New Mexico State University
Box 30001/CS,
Las Cruces, NM 88003
{epontell,wxiong}@cs.nmsu.edu

G. Gupta
Dept. Computer Science
University of Texas at Dallas
Richardson, TX 75083
gupta@cs.nmsu.edu

A.I. Karshmer
Dept. Computer Science
University of South Florida
4202 E. Fowler Ave, USF 30838
Tampa, FL 33620-3083
arthur@csee.usf.edu

ABSTRACT

We present a general framework for navigating complex structures—specifically, tables, frames, and forms—found in web-pages. Our framework is based on an (automatically or manually created) program written in a *domain specific language* that captures the semantic structure of the table/frame/form as well as specifies the strategy to be used for navigating it. We describe our general framework and the domain specific language we have designed.

Keywords
HTML, Web Browsers, Domain Specific Languages

INTRODUCTION

The Internet is a very visual medium, and thus largely inaccessible to visually handicapped members of our society. In this paper we are particularly concerned with the use of the Web as a media for courseware engineering and educational support. Indeed, a large amount of educational material is being placed on the Internet, and, in fact, most University instructors distribute course assignments, course syllabi, and other course-related information through the Web. However, while placing the course material on the Web makes it highly accessible to sighted students, it is an impediment to visually impaired students, unless the material is *universally accessible*. The basic premise of universal accessibility for learning is that a curriculum should include alternatives to make it accessible and applicable to students and teachers with different backgrounds, learning styles, abilities, and disabilities in widely varied learning contexts.

In spite of recent initiatives such as the Web Accessibility Initiative (WAI) by the W3C consortium, and recent availability of software tools that attempt to make the Internet universally accessible (e.g., through the use of screen-readers such as Window-eyes and Jaws), many problems remain with respect to universal accessibility. In particular, the majority of the existing tools cannot handle many of the complex features of the Web, such as *tables* and *forms* found in Web pages, as well as *frame-based* Web pages.

The various initiatives that have been put forward in the last two years to improve accessibility of the Web mostly rely on a number of strict policies the Web designer is required to follow. Although remarkable, the scope of these initiatives is necessarily limited: the astronomical number of Web pages developed daily, the relatively limited number of developers trained in accessibility technology, and the lack of support for the WAI requirements in the majority of existing Web development tools.

Comparatively less effort has been placed in the development of tools which improve the accessibility of existing Web documents, not necessarily designed according to the WAI requirements. As discussed later, a number of *aural browsers*—i.e., browsers capable of producing speech output in addition to the traditional visual output—have been recently proposed (see Section on related work). Nevertheless, even these approaches are limited in that they rely exclusively on the *syntactic* information present in the original HTML code. The limitations of these approaches is particularly evident when dealing with non-linear document components, such as tables and frames. In this paper we propose a general framework to improve the capabilities of aural browsers by taking advantage of the *semantic* structure of the document. This framework allows a visually-impaired user to navigate the non-linear components of the document in a more meaningful way.

Our techniques are based on parsing and analyzing the HTML structures that constitute a page (containing tables, forms, frames, etc.) to produce *structural representations*. The structural representation adopted is synthesized according to the syntactic structure of the HTML component (table, frame) and according to the semantic structuring of the document components. The semantic structure is inferred via analysis of the table format (e.g., structuring tags provided in HTML 4.0)—which allows to obtain accurate information for documents structured according to the WAI requirements—as well as via an independent document annotation process, performed either by the document creator or by a third party (e.g., a course instructor who is using web-based course material developed by somebody else). The individual annotating the document (referred to as annotator in the rest of the document) will specify how the table is to be semantically represented and navigated by specifying a series of commands in a specially designed command-language (such specially designed languages are called *Domain Specific Languages (DSLs)* [14]). Our tool will then navigate this structural representation in accordance to the annotator's commands to produce appropriate output, such as audio and speech.

In this paper we focus only on the non-visual browsing of tables, since tabular information is the hardest to convey to a blind person. This is primarily because of the structural information that is implicit in a table, which is lost if the table is linearly read out, as most current screen readers do. In our approach, the structural information is conveyed by the annotator through the navigation commands. Additionally, HTML supports only 2 dimensional tables. Thus, all tables, whether 3 dimensional or higher, have somehow to be mapped to a 2-d structure. Some semantic content is lost when the document's author maps a complex higher dimensional table to a 2-d table. The annotator can recapture this lost semantic information in the navigation commands. Essentially, the navigation of the table is done in such a way that the semantic structure of the table is conveyed. This aspect of our work can also be very useful for sighted users, especially, when visual browsing of large and complex tables is involved. For such large and complex tables, the annotations provided by the annotator or the author can help focus user's attention to the more important parts of the table.

In our approach, the structural information (or hints for comprehending and navigating the table) to be conveyed is formalized as an *annotated graph structure* expressed as a program written in a *Domain Specific Language*. The structure resembles in various aspects a conceptual graph and is designed to embed alternative *navigation viewpoints* [32] and multiple levels of abstractions.

The major contribution of this paper is to present a framework based on *domain specific languages* for representing and navigating tables. The DSL is constructed using state-of-the-art programming language technology and its semantics is based on conceptual graph structures. This approach allows for navigation of the table without unduly restricting the annotator or the end-user. Our approach is useful not only for non-sighted users, but also for sighted users especially when complex tables are involved. In this paper we only consider tables, a similar approach is being developed to handle frames and forms.

TABLE NAVIGATION

Our goal is to develop a framework and techniques that will facilitate the development of tools for navigating non-linear components of HTML documents (such as tables, frames, forms). The techniques we propose are novel as far as handling of these particular components of HTML documents is concerned; the remaining (linear) components of the HTML documents are handled according to the syntactic structure of the documents—i.e., by allowing the user to navigate the natural tree-based syntactic organization of traditional HTML documents.

In particular, in the context of this work we will deal exclusively with the management of *tables*, this being the most difficult aspect of the problem at hand. The ideas presented can be easily extended to deal with frames. The handling of forms will be dealt with in future works.

Our framework relies on two novel components:

1. A semantic representation of the HTML table's structure.

2. A *Domain Specific Language (DSL)* for *navigation* and *manipulation* of HTML tables;

The process for intelligently navigating HTML tables is represented in Figure 1. As illustrated in the figure, the HTML table extracted from the document being currently browsed is the object of a combined analysis phase, where both the syntax and the semantics of the table is studied. Syntactic analysis is used to automatically extract as much structural information as possible from the HTML definition of the table. E.g., the use of HTML 4.0 specific table elements and attributes, such as the `axis` and `scope` attributes. The semantic analysis phase accounts for any additional knowledge about the table which has been provided by either the creator of the document or any third party. Observe that this third party could be a software agent, extracting semantic information from the table according to a number of criteria (e.g., statistics on the previous uses of the table). As illustrated in Figure 1, this knowledge may also be available in form of *scripts* which automatically provide the necessary semantic information for a given family of pages—e.g., all pages automatically generated by a given CGI script.

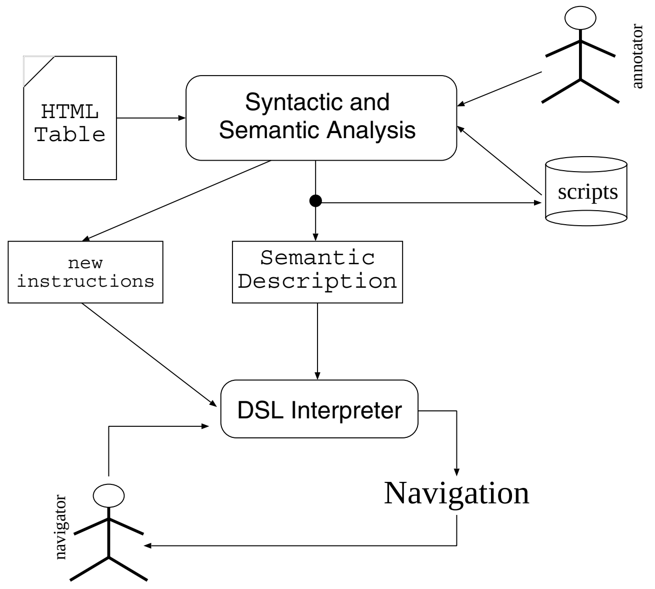

Figure 1: Architecture of the Navigation Process

The main outcome of the analysis phase is a semantic description of the input table. The description is represented as an *annotated graph*, and may encode multiple viewpoints of the table. Each viewpoint is hierarchically structured. The hierarchical view breaks down the table into its components according to a well-defined set of abstraction layers. For example, one standard hierarchical view abstracts all cells in each row, to facilitate row-wise navigation of the table. These hierarchical views are stored as a sequence of commands in the domain specific language. Standard hierarchical abstractions can be automatically generated, while the DSL program for non-standard abstractions has to be either manually written or automatically generated by an annotator (a human using a separate GUI-based tool, or a software agent performing further semantic analysis of the table). The analysis phase may also provide information to modify existing scripts or generate new ones—e.g., generate standard scripts from the first-time execution of a CGI script. The standard hierarchical views automatically constructed during the syntactic and semantic analysis phase may not be enough to meaningfully convey the structure of a table to the user. In such a case, the table author or an annotator (human or software) may have to provide custom hierarchical views expressed as a program in the domain specific language. For example, if the annotator wants the user to focus on cell elements in the diagonals of a table, then he/she will have to construct a separate view abstracting the cells in the diagonals.

The final phase in figure 1 is the actual navigation of the table by the user. The user navigates the table by executing Domain Specific Language (DSL) commands. These commands provide instructions for the traversal of the semantic representation of the table, in a fashion consistent with the desires of the annotator. The actual *"meaning"* of each instruction of the DSL is parameterized on the semantic structure of the table under considerations (see later). In fact, these commands are automatically executed, and the user only sees/hears there effect. Th sequence of commands executed is typically derived from the interaction of the non-sighted user with the browser—specialized key sequences are available to the user to control the navigation and each is translated into an appropriate sequence of DSL instructions. Part of the DSL program which performs navigation can be also automatically derived during the semantic analysis of the table or introduced by the annotator—such as semantic constraints which impose a specific order in the navigation of certain parts of the table (e.g., "if you have visited cells A and B, then you need to immediately visit cell C next") [22].

Semantic Structure

As mentioned earlier, one of the key components of our framework is the construction of a semantic representation of the table being examined. The semantic representation is based on an *annotated graph* structure, closely resembling the typical structure of a *conceptual graph* [35]. The nodes of the annotated graph represent either

- an individual location of the table (e.g., a `TD` or `TH` element in the HTML table), or
- a grouping of table cells representing a semantically meaningful component of the table.

As a simple example, Figure 2 represents a fragment of an annotated graph for a 2×3 table, placing emphasis on the decomposition of a table in *rows*. The different levels in the graph in Figure 2 represents the different level of abstractions enforced by the decomposition of the table in rows; the lower level contains the actual table locations (denoted by their indices in the figure). The nodes are connected by two types of edges:

1. the darker edges (*inner links*) are used to navigate within one level of abstraction (e.g., scanning the table row by row);
2. the lighter edges (*outer links*) are used to connect related elements belonging to different levels of abstraction (e.g., one table location with the row containing it).

For the sake of readability, in Figure 2 we have omitted some of the edges (e.g., some of the outer links between the bottom two layers).

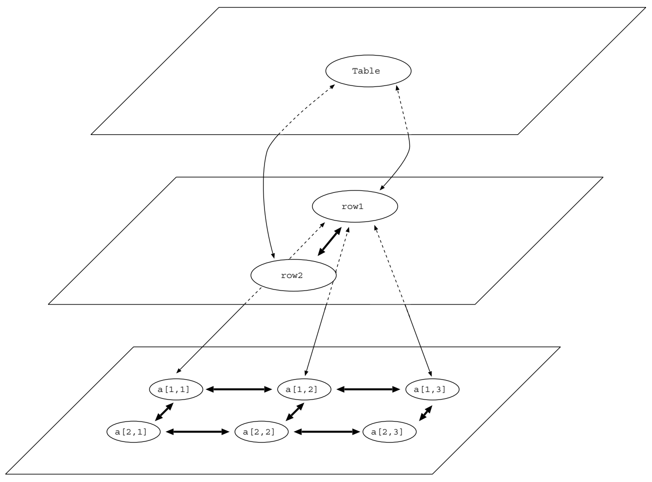

Figure 2: Fragment of a Semantic Representation for a Table

The different abstraction layers provide a vertical (hierarchical) decomposition of the semantic representation of the table. The hierarchical structure allows navigation of the table by successively refining the area of interest, and progressively focusing on a more precise segment of the table. The bi-directionality of the outer links allows the user to return to higher level of abstractions during the navigation.

The previously described vertical structure provides a single decomposition of the table—i.e., it provides a single *"view"*. In order to match the needs of the navigation process, and the fact that each table can be typically decomposed in different ways (e.g., by rows, by columns, by any arbitrary semantic-based data organization), our semantic representation of the table should account for the presence of multiple views (or *"viewpoints"*). This leads to a *horizontal* decomposition of the annotated graph, reflecting the alternative viewpoints provided. Figure 3 shows a simple example of horizontal decomposition of the table, indicating the alternative view of a table as a set of rows and a set of columns. Also in this case, for the sake of readability, we have omitted a number of edges (in particular most of the outer links between the bottom layer and the column layer).

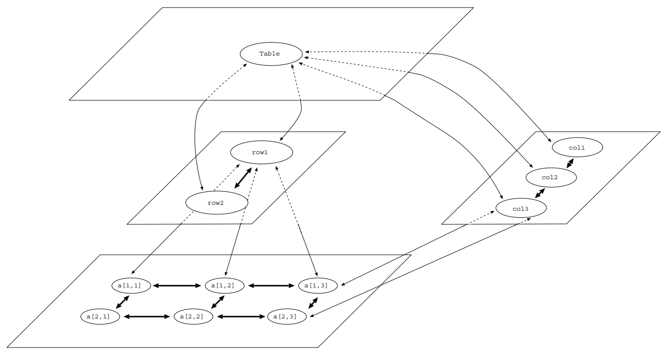

Figure 3: Fragment of a Semantic Representation of a Table

The table representation of Figure 3 provides two viewpoints; these can be selected either from the top-level view of the table (the topmost node in the hierarchical structure) or using outer links at any stage of navigation. For example, from any element of the bottom layer of the hierarchy it is possible to choose alternative viewpoints to move to the the immediately preceding layer.

This representation of tables presents a number of advantages:

- the representation subsumes any kind of table structuring which can be explicitly created by the Web designer using the accessibility features provided by HTML 4.0 (e.g., through the use of cell attributes such as `scope` and `axis`). Indeed, creating the standard semantic representations presented earlier from an HTML 4.0 table is a relatively straightforward task. Observe that HTML 4.0 allows the description of non-linear structure not in tabular forms (e.g., trees). These can be easily captured in our semantic representation scheme.
- the structure is easily extensible; new viewpoints can be added without disturbing the existing structure. This is particularly important. User studies have shown that, during navigation of tables, users frequently "collect" information along the way, and they use the collected information as reference points during navigation; in our context this behavior can be easily represented by allowing the creation of new viewpoints on the fly.

The examples presented in this sections are clearly very simple and are used only to exemplify the basic idea behind our semantic representation of tables. More general table representations can be easily accomplished in this framework. For example, consider the following table [31]:

	Meals	**Hotels**	**Transport**
San Jose			
25-Aug-97	37.74	112.00	45.00
Seattle			
27-Aug-97	96.25	109.00	36.00
28-Aug-97	35.00	109.00	35.00

Conceptually, this table has a 3-dimensional structure. Its standard encoding in our semantic representation can be automatically generated from a properly formatted HTML 4.0 description of this table [31]. Figure 4 illustrates a fragment of the representation of this table (the complete representation is omitted due to lack of space).

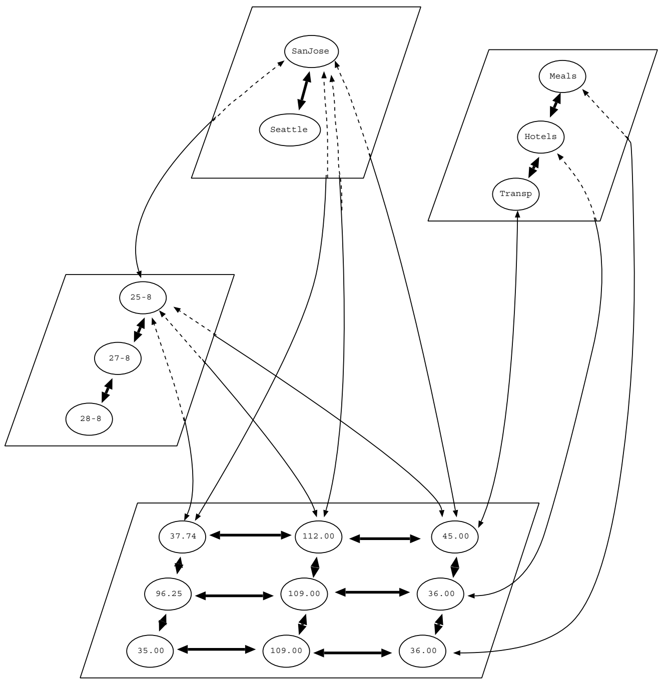

Figure 4: 3-dimensional Table

The Domain Specific Language

The cornerstone of our framework is a *domain specific language (DSL)*, which is used not only for capturing the structure of a table but also for expressing different navigation strategy. Thus, the hierarchical views shown in Figures 2, 3, and 4 will be represented using commands of this DSL. A domain specific language is a small, specialized language for writing programs in a particular application domain. Relying on a domain specific language based approach has a number of advantages: (i) it is platform independent; the DSL-based approach gives a general solution to the problem without committing to an implementation. (ii) the annotator can express the navigation actions in terms of commands of the DSL; we believe this makes the annotator's task easier. (iii) A DSL based approach is relatively easy to implement.

The structure of the DSL is relatively simple. The syntax is easily summarized in Figure 5. The syntax is incomplete since we only want to give the flavor of the DSL and avoid details—we have omitted several operations, e.g., those used to allow the creation of new nodes within an existing abstraction level.

A DSL program executes with respect to a predefined semantic representation of a table. The data domains manipulated by the DSL programs are:

- *nodes* of the table representation—i.e., either locations of the HTML table or elements representing abstraction levels;
- *sets* of items—where the items are themselves nodes of the table representation. Sets are used to represent operations which are nondeterministic in nature—i.e., they do not identify a unique node of the representation. For examples, operations like `abstract` (used to move to a higher level of abstraction)—since each node can be abstracted in different ways whenever there are different viewpoints—and `name`—which effectively creates a new abstraction level for a given set of nodes.

The language comprises different classes of instructions:

1. *Creation Commands:* these instructions associate an identifier or a name with a set of cells in the table. A cell in the ith row and jth column of the HTML table is referred to by the fixed name `Cij` (both rows and columns are numbered 1 onwards). The `group(Id, set)` commands associates the identifier `Id` with the set of cells `set` (in general, `set` could also be a set of previously defined Ids). A set is indicated by enclosing names of cells and Ids of other sets within square brackets. Creation commands are used to build the hierarchical semantic representations discussed in the previous sections.

2. *Inner Navigation:* these instructions are used to navigate *within* one abstraction layer; these instructions allows the user to follow the inner links in the table representation;

3. *Outer Navigation:* these instructions allows the navigator to move between different abstraction levels. Due to the presence of different viewpoints, these operations may require specification of the specific viewpoint that the user is interested in following.

4. *Extensions:* these instructions are used to extend the table representation on the fly; the user can make use of these instructions to create additional viewpoints or keep track of previously accessed information. A typical use of these instructions allows the user to "keep a finger" on certain previously encountered positions of the table.

5. *Inspectors:* these instructions are used to query the current position in the table representation during navigation; for example, the `path` command allows the user to determine the path from the root of the hierarchy to the current node.

The complete denotational semantics [34] for the basic instructions of the DSL has been developed and can be found elsewhere [25]. The semantics of the DSL is given in terms of sets of cells (these are the cells in the table that the user should be currently focusing on). Thus, each instruction takes a current set of cells and the table as input, and produces a new set of cells that the user should focus on next, and so on.

An Example

Consider the table shown earlier for travel expenses to San Jose and Seattle. The table consists of 6 rows. Using a regular screen readers such as windoweyes or Jaws, we'll just get a linear reading of the table, row by row. This will not give enough information to a blind user and the user may not be able infer what the author is trying to convey through the table. The author of the table intends to convey the following information to the reader, namely, that 1 day was spent in San Jose on 25th August, 1997, and that two days were spent in Seattle on 27th August, 1997, and 28th August, 1997, and then breakdown of expenses is given for all three days. If the table is linearly read out, the user will lose the association between the various amounts and the category on which they were spent. A better way of communicating the information would be to read out the column heading with each amount. A good way to navigate

Program	::=	Commands
Commands	::=	Command ; Commands — Command
Command	::=	CreateCommand — InnerCommand — OuterCommand — ExtendCommand — Inspector
CreateCommand	::=	group(Id, set)
InnerCommand	::=	first — last — next — previous
OuterCommand	::=	concrete (Id) — abstract (Id) — goto (Id)
ExtendCommand	::=	remember — name (Id) — ...
Inspector	::=	path

Figure 5: Syntax of DSL

the table would be to visit the cells of the table in the following sequence:

”San Jose”	⇒	”25-Aug-97”	⇒	”Meals”	⇒
”37.74”	⇒	”Hotels”	⇒	”112.00”	⇒
”Transport”	⇒	”45.00”	⇒	”Seattle”	⇒
”27-Aug-97”	⇒	”Meals”	⇒	”96.25”	⇒
”Hotels”	⇒	”109.00”	⇒	”Transport”	⇒
”36.00”	⇒	”Seattle”	⇒	”28-Aug-97”	⇒
”Meals”	⇒	”35.00”	⇒	”Hotels”	⇒
”109.00”	⇒	”Transport”	⇒	”35.00”	

This sequence of visiting the cells makes a clear association between column headings, the rows and the data in cells. In fact, this table is really a 3 dimensional table: the two halves of the table (the halves for San Jose and Seattle) should be stacked vertically. However, because of the 2-d restriction imposed by HTML, this 3-d structure is lost. This 3-d structure can be conveyed in the semantic (graph-based) representation of the table. The domain specific language program that corresponds to the order of navigation mentioned above is given below. For this navigation strategy the program actually consists of a series of `goto` statements. Note that in the program, the cells in the table are referred to by their coordinates, thus the cell in ith row and jth column is referred to as `Cij`.

```
goto C21  ; goto C31 ; goto C12 ; goto C32 ;
goto C13  ; goto C33 ; goto C14 ; goto C34 ;
goto C41  ; goto C51 ; goto C12 ; goto C52 ;
goto C13  ; goto C53 ; goto C14 ; goto C54 ;
goto C41  ; goto C61 ; goto C12 ; goto C62 ;
goto C13  ; goto C63 ; goto C14 ; goto C64
```

At each step, the contents of the current cell will be then read out and/or highlighted before the next command is executed.

A part of the structure represented in Figure 4 can be constructed using the sequence of DSL commands listed below. This will allow a more flexible navigation on part of the user. The commands below show also how the user can obtain the list of expenses for Seattle on August 28th.

```
// Create New Abstractions
group(expenses, [san-jose, seattle])
group(san-jose, [25august])
group(seattle, [27august,28august])
group(25august, [8-25meal,8-25hotel,8-25transport])
group(27august, [8-27meal,8-27hotel,8-27transport])
group(28august, [8-28meal,8-28hotel,8-28transport])
group(8-25-meal, [37.24])
....

// Sample Navigation
concrete(seattle)
concrete(28august)
concrete(8-28meal)
abstract
concrete(8-28hotel)
abstract
concrete(8-28transport)
```

The outcome of that navigation will be of the type:

expenses	⇒	Seattle	⇒
August 28	⇒	Meals	⇒
$35.00	⇒	Hotels	⇒
$109.00	⇒	Transport	⇒
$35.00			

In this program, the `group` command is used to band the appropriate cells together and give them a name. They are essentially used to build views that the user is interested in. For example, `group(expenses, [san-jose,seattle])` gives the Id `expenses` to the set of cells corresponding to the abstractions `seattle` and `san-jose` (defined in the next lines). The `goto` and `concrete` commands will read out the contents of the associated cells as well as move on those cells/abstractions.

Note that the programs are quite simple and intuitive, and therefore easy to write. Moreover, a GUI based tool is being developed that annotators can use to generate these DSL commands automatically.

IMPLEMENTATION TECHNOLOGY

The implementation of our scheme relies on the use of advance programming languages technology. The specification and implementation of the DSL relies on the use of a novel methodology, called *Horn Logic Denotation* [13, 14]. The methodology allows to describe, within a single framework based on standard logic programming [36], all the different components of a language specification—i.e., its syntax, semantic domains, and *valuation functions* (i.e., interpretation of the language's instructions as functions over the semantic domains).

The advantage of Horn Logic Denotation is that it allows to automatically derive, with no extra effort, an *executable interpreter* for the language—the use of logic programming *Definite Clause Grammars* for describing the syntax (as a standard context free grammar) automatically provides an executable parser, while standard logic programming clauses provide a convenient tool for the description of the valuation functions—since logic rules define relations, and valuation functions are nothing else than special kinds of relations. Furthermore, the use of standard partial evaluation techniques [18] allows to automatically combine (fragments of) DSL programs and the DSL interpreter to produce efficient compiled code.

The ability to reason within a single, very declarative framework (such as logic programming) provides other added benefits. The ability to add conditions and constraints to the execution of the interpreter (or to the compiled code) allows to obtain with no extra effort debuggers, profilers, etc., for the DSL. In fact, an interpreter for this DSL has already been developed. More details of the implementation can be found elsewhere [25].

TABLES ANNOTATION

We have left the issue of building the semantic representation of the table open on purpose, this being the current focus of research. The preliminary approach we have taken is articulated in the following steps:

1. the HTML table is extracted from the document and syntactically analyzed using an extended HTML parser. The goal of this parser is to recognize the syntactic structure of the table. Various items are accounted for in this phase, including

   (a) identification of irregular tabular components (e.g., `rowspan` and `colspan` attributes)

   (b) identification of header cells (e.g., `TH`)

   (c) identification of HTML 4.0 specialized attributes for structural information (e.g., `scope`, `axis`)

   This phase provides basic structural knowledge of the table and allows the automatic generation of a number of different viewpoints.

2. the HTML table is subject to a second analysis phase which attempts to extract additional information which can provide structuring knowledge about the table. Various elements of a table are considered as hints for creation of structural components—e.g., use of different background colors for different parts of the table.

3. a human annotator has the ability of affecting the creation of the semantic representation by adding or removing components from the automatically generated semantic structure. This is achieved via a very intuitive graphical tool, which allows the user to review the existing semantic representation, create new abstraction (by clicking on the table cells he is interested in) or modifying existing abstractions (adding or removing cells).

In our future work, we plan to invest considerable effort in improving this phase of the process. Various extensions are currently envisioned:

- if we maintain logs of navigation of tables then these logs can be used to statistically derive models of how the table has been used. These models can be used to derive alternative viewpoints or to classify the existing viewpoints based on how "likely" they are to be used;

- a semantic analysis of the content of the table can be performed to derive measures of how "related" different cells are (using, for example, techniques such as Decomposable models [30] and Latent Semantic Indexing [10]).

RELATED WORK

The issue of promoting Web accessibility for visually-impaired individuals has gained prominence in most of the recent research and development areas associated to the Internet. The W3C Consortium launched in 1997 the *Web Accessibility Initiative (WAI)* [20]. The goal is to promote the development of standards (e.g., HTML 4.0) and guidelines (e.g., HTML authoring guidelines, authoring tool guidelines) to promote Web accessibility—mostly directed towards visually-impaired users. Similar initiatives have been also developed by major corporations, as in the case of the Microsoft accessibility initiative [27] and IBM's Special Needs Systems program [17]. A number of tools have been developed which support the development of accessible documents:

- tools to validate documents w.r.t. the WAI accessibility criteria (Bobby [6] and HoTMetaL [33]);

- toolkits to simplify the development of accessible documents, such as the *Aural Cascading Style Sheets* [24], which allow to associate speech components to the various HTML elements;

- APIs which allows to develop Internet related software with high-level accessibility features, as in the case of *Microsoft Active Accessibility* (technology embedded in most recent MS products which enables software aids to recognize Window components) and in Sun's *Java Accessibility API* [37].

Research has also been done on software engineering methodologies for developing non-visual interfaces [38].

The work we are proposing in this project aims to provide integrated client-side (i.e., at the browser level) and "server"-side (i.e., at the level of the Web page author or provider) 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 was raised in various works [16, 4, 3]. This was also validated by recent survey studies [9, 12] which explicitly pointed out the ineffective performance of existing screen readers when coupled with them Web. Gunderson & Mendelson specifically cite the task of locating information in table structures as one of the most challenging tasks for visually impaired users [12].

The most relevant proposals of browser-level support for accessibility are:

- the work of Asakawa et al. [1, 29], recently integrated in the IBM *Home Page Reader (HPR)* [2]. HPR obtains documents from Netscape—allowing concurrent access by sighted users. Navigation can be performed at the level of links (i.e., move from link to link), words, or page elements. Tables and forms can only be navigated sequentially. Preliminary documentation does not suggest the ability to navigate frames.

- The proposal by Dillin [8] is based on elementary conversion from HTML into a MS-NOTEPAD like style of presentation. Considerable information is lost and special features like tables and frames are ignored.

- the *pwWebSpeak* browser [7]: as for HPR, also pwWebSpeak offers speech output through HTML interpretation. The current reported version of pwWebSpeak does not support frames and tables are represented 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 *BrailleSurf* system allows filtering of HTML into Braille [15]. Tables are prefixed by a table indicator, a caption, and the option to bypass the table; frames are converted in a collection of links plus a prefix.

Other tools have been considered as well (e.g., Sensus Internet Browser, use of Windows-eyes for Internet browsing [11], NLP based summarization system for Web page access [39]). Applications of Web-based course delivery for visually-impaired students have been reported as well by various researchers (e.g., [21, 19, 23]).

Of these efforts, none currently accounts for the possibility of modifying the behavior of the browser according to the layout or content of the individual Web pages. Some of these proposals have the potential to be extended in this direction—e.g., pwWebSpeak makes use of an Audio Stylesheet mechanism to define how the pages should be read, and it is possible to envision the possibility of associating separate stylesheets with different Web pages.

None of the proposals presented in the existing literature has considered viewing the navigation task in terms of execution of programs from a domain specific language.

The semantic representation of tables adopted in this work closely resembles a conceptual graph structure [35], where the nodes represents different abstraction levels in the definition of the table, and the links denote relationships concerning the logical location of the components and the structuring of the abstraction levels. The adoption of alternative abstraction structures for the same table (e.g., rows or columns or diagonals etc.) bears some similarities to the notion of *viewpoints*, originally explored in the context of spatial representations [28], and widely used in the context of knowledge representation [26, 5]. Indeed our annotated graph representation is very close to the extension of conceptual graph with viewpoints introduced by various researchers [32].

DISCUSSION

As discussed in the section on related work, none of the existing proposals in the literature takes the approach of viewing navigation of tables in terms of execution of programs from a domain specific language. This novel view presents a number of advantages:

- the creation of the semantic representation can be itself expressed in terms of execution of a DSL program—since the DSL includes commands for the creation of viewpoints and abstraction layers. This allows to create automatic scripts for the generation of the semantic representation—a feature that can be hardly achieved in other existing proposals, where most of the annotation work (e.g., creation of stylesheets) is mostly a manual activity.
- the ability to adapt the structure and semantics of the DSL according to the specific table considered, allows to make the navigation process flexible, removing restrictions imposed by a fixed user interface (such as those provided by many existing aural browsers).

CONCLUSIONS

In this paper we presented a general framework based on domain specific languages for navigating tables found in web-pages. The semantic description of the table and the strategies for navigating it are both described in this domain specific language by an annotator in order to facilitate the navigation of the table by a visually impaired user. At present the Domain Specific Language has been designed and an interpreter for it already developed. This interpreter is currently being interfaced with a web-browser so that visually impaired individuals can better understand information expressed as a table. A GUI tool for facilitating the task of annotating a table is also being developed.

ACKNOWLEDGMENTS

We wish to thank Doug Gillan for the collaboration in the development of this project. The authors are partially supported by NSF grants CCR-9875279, HRD-9906130, and EIA-9810732.

REFERENCES

[1] C. Asakawa and T. Itoh. User Interface of a Home Page Reader. In *Proceedings of ASSETS*. ACM Press, 1998.

[2] C. Asakawa and C. Laws. Home Page Reader: IBM's Talking Web Browser. Technical report, IBM Ltd., 1998.

[3] D. Bolnick and G. Freed. New Developments in Web-based Accessible Multimedia. In *Proc. of Tech. and Persons with Disabilities Conference*. CSUN, 1998.

[4] J. Brewer, D. Dardailler, and G. Vanderheiden. Toolkit for Promoting Web Accessibility. In *Proc. of Tech. and Persons with Disabilities Conference*. CSUN, 1998.

[5] B. Carré, L. Dekker, and J-M. Geib. Multiple and Evolutive Representation in the ROME Language. *TOOLS*, 1990.

[6] Center for Applied Special Technology. Bobby 3.0. `www.cast.org/bobby`, 1998.

[7] J.C. De Witt and M.T. Hakkinen. Surfing the Web with pwWebSpeak. In *Proc. of Tech. and Persons with Disabilities Conference*. CSUN, 1998.

[8] D. Dillin. Concept of a Structural HTML Page Browser to Support Access to WWW for People with Disabilities. Yuri Rubinsky Insight Foundation, 1997.

[9] C. Earl and J. Leventhal. A Survey of Windows Screen Reader Users: Recent Improvements in Accessibility. *Journal of Visual Impairment and Blindness*, 93(3), 1999.

[10] P.W. Foltz. Latent semantic analysis for text-based research. *Behavior Research Methods, Instruments and Computers*, 28(2):197–202, 1996.

[11] D. Geoffray. The Internet Through the Eyes of Windows-eyes. In *Proc. of Tech. and Persons with Disabilities Conference*. CSUN, 1999.

[12] J. Gunderson and R. Mendelson. Usability of World Wide Web Browsers by Persons with Visual Impairments. *Proceedings of the RESNA Annual Conference*, pages 330–332, 1997.

[13] G. Gupta. Horn Logic Denotations and Their Applications. In *The Logic Programming Paradigm: A 25 year perspective* Springer Verlag. pp. 127-160, 1999.

[14] G. Gupta and E. Pontelli. A Horn logic denotational framework for specification, implementation, and verification of domain specific languages. Technical report, NMSU, March 1999.

[15] D. Hadjadj and D. Burger. BrailleSurf: an HTML Browser for Visually Handicapped People. In *Proc. of Tech. and Persons with Disabilities Conference*. CSUN, 1999.

[16] P. Hendrix and M. Birkmire. Adapting Web Browsers for Accessibility. In *Proc. of Tech. and Persons with Disabilities Conference*. CSUN, 1998.

[17] IBM Special Needs Systems. `www.ibm.com/sns`, 1998.

[18] N. Jones. Introduction to Partial Evaluation. In *ACM Computing Surveys*. 28(3):480-503. 1996.

[19] P. Jones. Speech-enabled Web Access: an Instructional Case Study. In *Proc. of Tech. and Persons with Disabilities Conference*. CSUN, 1998.

[20] H. Kochocki, S. Townsend, N. Mitchell, and A. Lloyd. W3C Launches Internation Web Accessibility Initiative. Technical report, W3C Consortium, 1997.

[21] L.E. Kraus. Teaching Mathematics to Students with Physical Disabilities using the World-wide Web. In *Proc. of Tech. and Persons with Disabilities Conference*. CSUN, 1998.

[22] B.C. Ladd, M.V. Capps, P.D. Stotts, and R. Furuta. Multi-Head Multi-Tail Mosaic: Adding Parallel Automata Semantics to the Web. In *Proceedings of the Fourth International World Wide Web Conference*, 1995.

[23] D.J. Leu and D.D. Leu. *Teaching with the Internet: Lessons from the Classroom*. Christopher-Gordon Publishers, 1997.

[24] C. Lilley. Aural Cascading Style Sheets (ACSS). Technical Report NOTE-ACSS-970107, W3C Consortium, 1997.

[25] W. Xiong. Non-Visual Browsing of Complex HTML Structures. Master's thesis. New Mexico State University. Forthcoming.

[26] O. Marino, F. Rechenmann, and P. Uvietta. Multiple Perspectives and Classification Mechanism in Object-Oriented Representation. In *Proceedings of 9th ECAI*, Pitman Publishing, 1990.

[27] Microsoft Accessibility and Disabilities. `www.microsoft.com/enable/default.htm`, 1998.

[28] M. Minsky. A Framework for Representing Knowledge. *The Psychology of Computer Vision*, McGraw-Hill, 1975.

[29] T. Oogane and C. Asakawa. An Interactive Method for Accessing Tables in HTML. In *Proceedings of ASSETS*. ACM Press, 1998.

[30] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Mateo, CA, 1988.

[31] D. Raggett, A. Le Hors, and I. Jacobs. HTML 4.0 Specification. Technical Report REC-html40-19980424, W3C Consortium, 1998.

[32] M. Ribiere and R. Dieng. Introduction of Viewpoints in Conceptual Graph Formalism. *International Conference on Conceptual Structures*, Springer Verlag, 1997.

[33] SoftQuad Inc. *HoTMetaL Pro 5.0 Reviewer's Guide*, 1998.

[34] D. Schmidt. *Denotational Semantics: a Methodology for Language Development*. W.C. Brown Publishers, 1986.

[35] J.F. Sowa. Conceptual Structures. In *Information Processing in Mind and Machine*, Addison-Wesley, 1984.

[36] L. Sterling and E. Shapiro. *The Art of Prolog.* MIT Press, 1996.

[37] Sun Microsystems. *Accessibility Support for the Java Platform*, 1998.

[38] G. Weber. Programming for Usability in Nonvisual User Interfaces. In *Proceedings of ASSETS*, pages 46–48. ACM Press, 1998.

[39] M. Zajicek, C. Powell, and C. Reeves. Web Search and Orientation with Brookestalk. In *Proc. of Tech. and Persons with Disabilities Conference*. CSUN, 1999.