# TEXUS: A Task-based Approach for Table Extraction and Understanding

Roya Rastan[1] Hye-Young Paik[1] John Shepherd[1]

[1] University of New South Wales, Australia
{rrastan,hpaik,jas}@cse.unsw.edu.au

**Technical Report**
**UNSW-CSE-TR-201504**
**May 2015**

THE UNIVERSITY OF
NEW SOUTH WALES


School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

**Abstract**

In this paper, we propose a precise, comprehensive model of table processing which aims to remedy some of the problems in the discussion of table processing in the literature. The model targets application-independent, end-to-end table processing, and thus encompasses a large subset of the work in the area. The model can be used to aid the design of table processing systems (and we provide an example of such a system), can be considered as a reference framework for evaluating the performance of table processing systems, and can assist in clarifying terminological differences in the table processing literature.

# 1 Introduction

Tables are a widely-used structure for data presentation and summarisation in documents from many different domains. Tables use layout to arrange information and convey meaning, and are capable of presenting and communicating complex information to human readers. Human readers, in turn, are capable of using layout features as clues for interpreting the logical meaning of the information in tables. Because tables are a rich and widely-available source of inter-related data, it would be useful if their contents could be automatically extracted and manipulated by computers. However, the diversity of layouts and variety of encodings (e.g. HTML, PDF, plain text) of tabular information makes extraction and understanding a challenging problem.

Various research communities, such as machine learning and information retrieval, have worked on the problem of table processing and many approaches have been proposed. However, existing approaches almost always tackle only a subset of the problem (e.g. tables in a specific domain or with particular layout), or focus on sub-tasks of the complete table processing problem (e.g. locating tables). Importantly, most systems are designed as monolithic black-boxes, which makes it difficult to investigate their structure and performance, or to re-use/replace components to advance the state of the art.

There has been some work towards building a coherent, systematic view of the table processing problem. Hurst [12] provided multiple table models, to define tables from different abstraction levels. Silva [5] defined end-to-end table processing as a set of tasks, thus reducing the monolithic, black-box view of the problem. Long [20] proposed an agent-based architecture in which table processing systems are implemented by composing different agents, showing that reusability is relevant to table processing. Taken together, these works almost provide a complete framework for discussion of table processing. However, they were developed independently and do not dovetail well enough nor provide sufficient detail to support the development and evaluation of full end-to-end table processing systems.

In this paper, we propose a task-based approach to table processing (called TEXUS) and provide detailed data and task models that encompass the essential aspects of the above work, and extends it to the point where it *can* be used as a basis for implementation and evaluation of complete end-to-end table processing systems. We make the following contributions: (1) we define table-processing as a set of well-defined tasks with an aim to build a system that produces application-independent table descriptions. (2) we define precise data models to provide standard task interfaces. (3) we show how the models can be used in implementing an end-to-end table processing system.

# 2 Related Work

In this section, we discuss previous work on table processing techniques and systems, and summarise how we make use of it in developing our models and systems. Before considering the work of others, we need to define precisely what we mean by "end-to-end table processing" (i.e. what are the end points). The starting point for our work is documents in PDF format. We can do this without loss of generality, since PDFs make up a large proportion of the

documents we typically encounter, and all other major document formats (e.g. ASCII text, HTML) can be readily converted to PDF. The ending point is an application-independent representation of the table, which could be used for further tasks such as information extraction. The representation we choose is based on Wang's notation [30].

There have been a number of surveys on table processing, although each survey has dealt with just some aspects of the problem. Zanibbi et al. [33] reported on table models, observations and transformations in the table processing literature. Lopresti et al. [22] focused on the definition of "tabularity" and tabular browsing. Embley et al. [6] explored table transformations in semi-automated table processing systems. We just focus on the work that has attempted to deal with the design and development of an end-to-end table processing solution.

The pioneering work in end-to-end table processing is Hurst's PhD thesis [12] which gives a general table model with different abstraction levels (Physical, Functional, Structural, Semantic) to facilitate the interpretation of tables. Although Hurst provided a comprehensive table model, his focus is on the model rather than on the details of the process of table extraction and understanding. Additionally, his discussion of process deals only with determining the internal structure of an identified table, and not with locating the table in the document.

Silva [5, 2] was the first to identify the tasks involved in end-to-end table processing. She proposed a design for table processing as a sequence of steps, with feedback loops between the steps to reduce the possibility of errors. Silva's work is important in identifying a set of basic tasks for table processing. However, no formal definition of these tasks and their inputs/outputs was given and only some components of the proposed design were implemented.

As noted above, our end goal for table processing is based on the abstract table model defined by Wang in her PhD thesis [30]. While the thesis focuses on table composition problems, it also provides a table abstraction model which neatly separates a table's logical structure from its layout structure and has been widely cited in the literature [12, 14, 29, 26]. Wang formally defined the notion of an abstract table to describe the logical relationships among table items, and proposed a set of logical operations to manipulate tables based on these logical relationships. Because the model describes tables in a layout-independent way, it has been used as an intermediate model for many table processing systems which aim towards information extraction, such as [14]. While Wang's model is popular, it is still a challenge to develop systems which can automatically derive a Wang-representation from a table in a document.

Long [20] was the first to implement a table processing architecture that encourages collaboration and component re-use. Long defined a multi-agent blackboard architecture, along with guidelines for table boundary identification and table interpretation. Individual agents tackle different partial solutions like processing input characters and processing input lines. Although the system design is sound, and suggests that components from other table processing systems might be incorporated, it is not specified how to achieve this, and the issue of conflict resolution over heterogeneous agents is noted as an open problem.

Our goal is to integrate aspects of the above related work to produce a framework for end-to-end table processing which: (a) identifies the sub-tasks involved (a la [2]), (b) gives precisely defined models to describe the input and output of

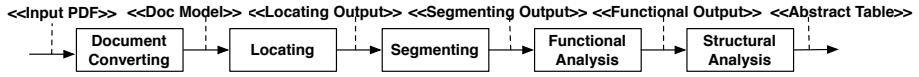


Figure 3.1: Task-based End-to-end Table Processing Pipeline

each sub-task (a la [12]), and (c) packages this as a collection of modules (a la [20]), with well-defined interfaces, to provide a "workbench" for developing tools and techniques to further the state-of-the-art in table processing. The input of a system built using this framework is a PDF document and the output is a set of abstract table models (a la [30]) describing each of the identified tables in the document. Our original contribution is the integration of these ideas, the development of the models for each sub-task, and extensions to some of the methods/models outlined above.

# 3 Task-based Table Processing

As just described, we view end-to-end table processing as a coherent sequence of tasks that takes a document as input and produces an abstract representation of the tables in the document as the final output. We define a series of data models to describe the inputs and outputs of the tasks. The input PDF document is initially partitioned into a sequence of *text chunks*, and these form the atomic elements of further processing. The final output of the table processing is represented by an extension of Wang's *abstract table model* [30]).

Our core table processing tasks are as follows: (1) *Document Converting*: convert the PDF input document to our proposed document model, (2) *Locating*: find the tables in the document (their outer boundaries), (3) *Segmenting*: recognise the inner boundaries of each table (cells, rows and columns), (4) *Functional Analysis:* identify the role of each cell in each table (data or access), and (5) *Structural Analysis*: detect the logical relationships between table cells and provide the result as an abstract table. The last four tasks correspond to the first four tasks in Silva [2]. We omit Silva's fifth task (Interpretation), since it is inherently application-dependent, and our end-point is intended to be application agnostic. The tasks would typically be connected in a simple pipeline, but more complex controls such as looping and nested composition can also be considered (although not in this paper). Figure 3.1 shows a pipeline of these tasks implementing an end-to-end table processing system. In order to discuss table structures, we use Wang's table terminology [30]. According to Wang, tables are divided into four main regions, delineated by means of a *stub separator* and a *boxhead separator* which are frequently, but not always, shown as physical lines. The lower-right region of the table (the *body*) contains the data. The upper-right region (the *boxhead*) contains column headings and sub-headings. The lower-left region (the *stub*) contains labels which provide access to the data rows. The upper-left region (*stub head*) contains the headings for the columns in the stub. Figure 3.2 shows an example table with these regions marked.

Recall that the output of our system is a set of abstract table instances, one for each located table. Wang [30] defines an *abstract table* by an ordered pair $(C, \delta)$ where $C$ is a finite set of labelled domains and $\delta$ is a mapping from $C$ to the universe of possible data values. The categories appear in the table as

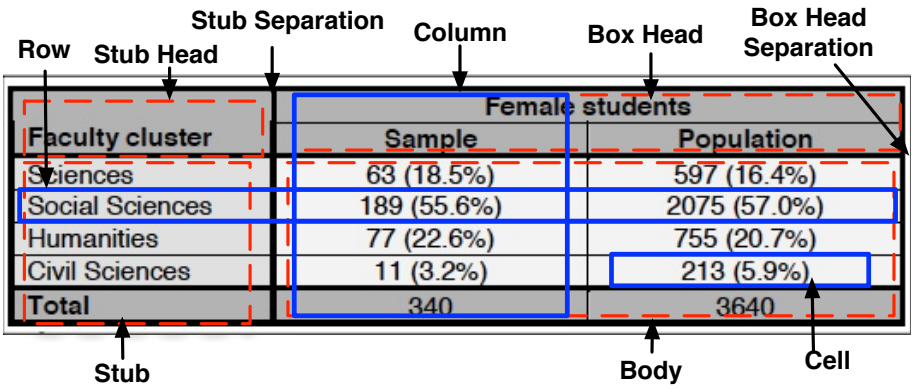


Figure 3.2: Wang Table Terminology

headings. The $\delta$ mapping relates the categories to the data values in the table body.

We illustrate the various concepts in Wang's notation via the example table in Figure 3.2. This table has two dimensions and therefore, two top-level categories. The first category is Faculty Cluster, with five subcategories (Sciences, Social Sciences, ... Total). Female Students is the next category with Sample and Population as its subcategories. The following gives examples of the kind of abstract table output obtained by processing this table:

Category set (hierarchy):
$C = \{ (Faculty\ cluster, \{(Sciences, \Phi), (Social\ Sciences, \Phi), (Humanities, \Phi), (Civil\ Sciences, \Phi), (Total, \Phi)\}), (Female\ students, \{(Sample, \Phi), (Population, \Phi)\}) \}$

Two examples from the $\delta$ mapping:
$\delta(Faculty\ cluster.Sciences, Female\ students.Sample)$ = "63 (18.5)%"
$\delta(Faculty\ cluster.Sciences, Female\ students.Population)$ = "597 (16.4)%"

Note that tables typically appear in documents adorned by a title, caption, notes etc. These are not considered as part of the table, and hence do not appear in the abstract table model, but are included in the final output as *table metadata*.

Our goal is to describe end-to-end table processing via a task-based approach where the inputs and outputs of each task are precisely defined. We believe that this approach provides the following benefits: (1) a well-defined decompostion of the task into generally agreed components (2) a consistent vocabulary for describing system scope and goals, (3) reusablity and repeatability in system design and development (4) the opportunity for interoperability of different implementation techniques and approaches. Our end-point is an application-independent model of the tables in the document (a mapping from headings to data cells, in Zanibbi et al's [33] terms). We believe this provides a suitable starting point for semantic analysis and other kinds of domain-dependent analysis, and so we adopt the well-known Wang abstract table model [30], extended to handle tables with no obvious stub.

# 4 The Building Blocks of TEXUS

In this section, we give more details on the tasks in the end-to-end table processing framework, and the data models that connect them. We first discuss our data model for input documents. We then consider the four table processing tasks, partitioned into two groups (*Table Extraction* and *Table Understanding*), and give details of their data models. Finally, we briefly discuss table metadata.

## 4.1 Document Converting

Since our table processing starts with *documents*, we need a suitable model for documents. Both *layout* and *content* are important in analysing tables [17], so our model needs to consider both aspects. Our input documents are PDFs, and there are a range of tools and techniques for extracting layout and text features [11] from PDF documents. We use the `pdftohtml` tool for our initial processing, followed by pre-processing as described below.

Finding the proper atomic unit of document content for table processing is an important first step. For our purposes, and following from the discussion in [9], individual characters are not a useful atomic unit. Since our primary concern is with the contents of table cells, which typically contain one or more words or numbers, possibly on several lines, we start from the notion of a *text chunk*, which intuitively corresponds to a group of words with a bounding box. In order to identify text chunks, we assume that we can extract properties such as $Left$, $Right$, $Top$, $Bottom$ for each element in the document. Fig. 4.1 shows a document page with some elements of the document model identified (text chunks are indexed as $ch_i$).

While we do not consider individual characters in table processing, they do occur in the document converting step which takes the raw PDF document and produces a tagged document with text chunks as the basic unit. The pre-processor builds text chunks according to the following:

**Character**: Each `Character` $C_i$ is defined as a tuple of attributes $C_i = (Top, Left, Height, Width, Font, CHAR)$ which describe its position in the document page as well as the actual character. Characters include both visible characters as well as whitespace characters such as space, tab, carriage return, and linefeed.

**Word**: A `Word` $W_j$ is a sequence of horizontally consecutive `Visible Characters` $\langle C_1, C_2, ..., C_n \rangle$ not containing any whitespace characters. Whitespace characters are also called `Word delimiters`.

**Text Chunk**: A `Text Chunk` $Ch_j$ is a sequence of horizontally consecutive `Words` $\langle W_1, W_2, ..., W_n \rangle$ with a bounding box, where the distance between consecutive Text Chunks is greater than the size of the smallest `Word delimiter`. The bounding box is determined by the $Left$ of the leftmost character, the minimum $Top$ value of all characters, the maximum $Bottom$ value of all characters, and the $Right$ of the rightmost character. Text chunks do not overlap. **Line**: A `Line` $L_j$ is a sequence of horizontally consecutive `Text Chunk(s)` $L_j = \langle Ch_1, Ch_2, ..., Ch_n \rangle$ with a bounding box based on $Ch_1$ and $Ch_n$ which ends with an End-of-Line delimiter.

**Text Region**: A `Text Region` $TxR_j$ is a sequence of `Lines` $TxR_j = \langle L_1, L_2, ..., L_m \rangle$ with a bounding box based on $L_1$ and $L_m$. A `Text Region` ends when a

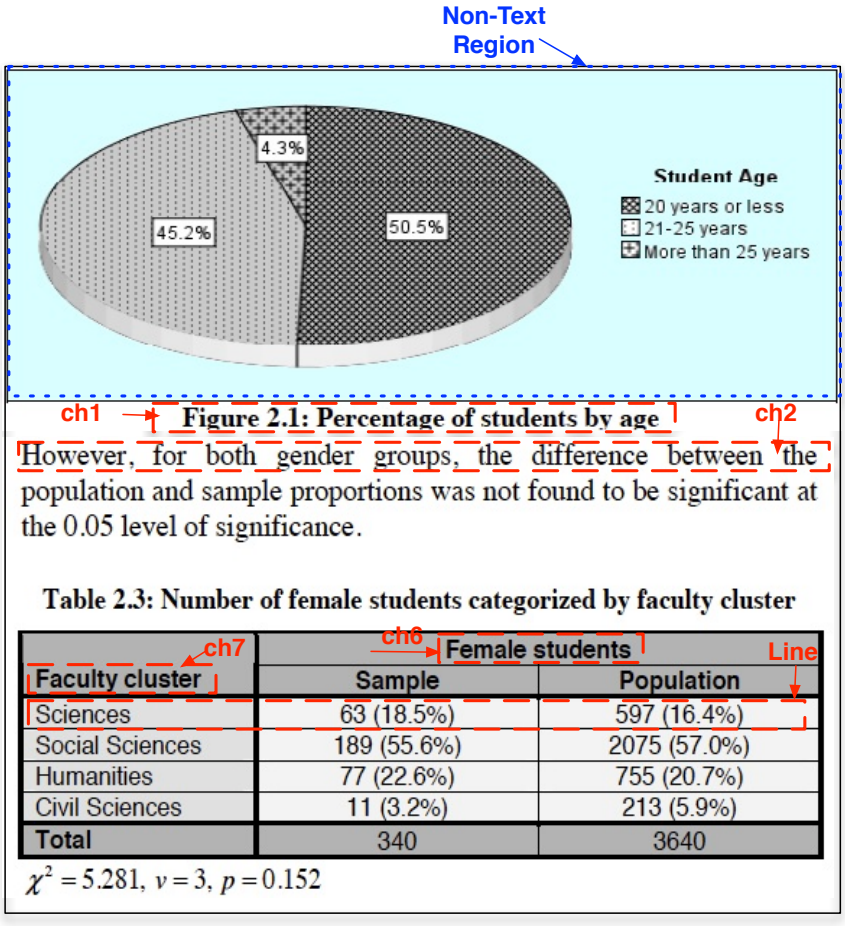


Figure 4.1: Sample of a document page

`Non-Text Region` or End-of-page is encountered. The most common type of `Text Region` is a column of text on a single page.

**Non-text Region**: Any element, such as an image, that is not composed of text is treated as part of a `Non-Text Region`. Such elements can be identified by `pdftohtml`. Consecutive `Non-Text Regions` are merged into a single region.

**Page**: A `Page` is a sequence of `Text Regions` and `Non-Text Regions`. Page boundaries can be detected by noting the resetting of the *Top* values of subsequent document elements.

**Document**: A `Document` *Doc* is a sequence of `Pages`, $Doc = \langle P_1, P_2, ..., P_q \rangle$.

In above definition, we assume that each table is contained within a `Text Region` (and could be the entire `Text Region`). We also assume that tables do not span multiple pages (although work is currently underway to handle this case).

## 4.2 Table Extraction

In this section, we detail the *Table Extraction* part of TEXUS. Table extraction goes beyond simply detecting where tables are in a document; it separates tables from the rest of the document and represents each of them in the form of a *Physical* abstraction model. Table extraction consists of two main processes: *Locating* and *Segmenting* and they are mostly described based on the layout attributes of the document and table.

### Locating

The aim of locating is to find the starting line and ending line of each table in the document. The location of each table is identified by its boundaries. Lines in the input document are either *table lines* or *text lines*.

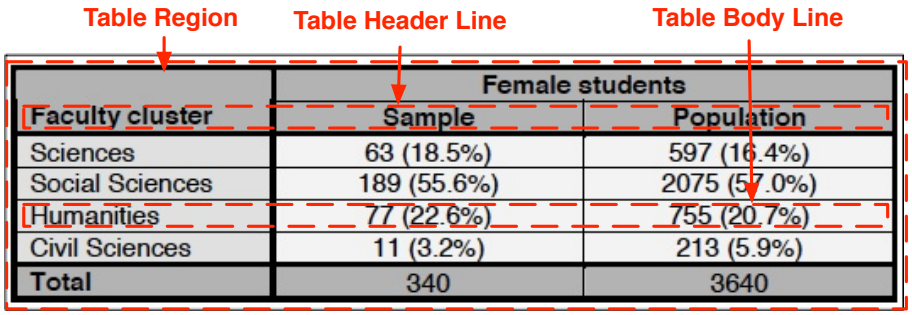


Figure 4.2: Table Location model

Tables contain two distinct types of `Table Line`:

**Table Body Line**: A potential `Table body Line` $TBL_i$ is a `Line` containing two or more `Text Chunks` and located in the Table Body Region.

**Table Header Line**: A `Table Header Line` $THL_i$ is a `Line` containing one or more `Text Chunk(s)` and is placed in Table BoxHead.

The Table location model is then defined in terms of `Table Lines`:

**Physical Table Location** ($T_{Loc}^{Phy}$): The model for the $i^{th}$ table $T_{Loc\ i}^{Phy}$ is contained in a `Text Region` and spans a sequence of lines $\langle L_j, L_{j+1}, ..., L_k \rangle$, where zero or more `Table Header Lines` are followed by one or more `Table Body Lines`. $T_{Loc\ i}^{Phy}$ is defined by a `Table Region` $TR_i$ whose bounding box is determined by $(Left(L_j), Top(L_j))$ as `Upper Boundary` $UB$ and $(Bottom(L_k), Right(L_k))$ as `Lower Boundary` $LB$.

### Segmenting

The aim of segmenting is to recognise and detect the inner boundaries of the table (i.e. the rows, columns and individual table cells). Each table cell can be identified by at least a (row,column) position. Table segmentation (e.g. Figure 4.3) is defined via a `Physical Table Segments` model.

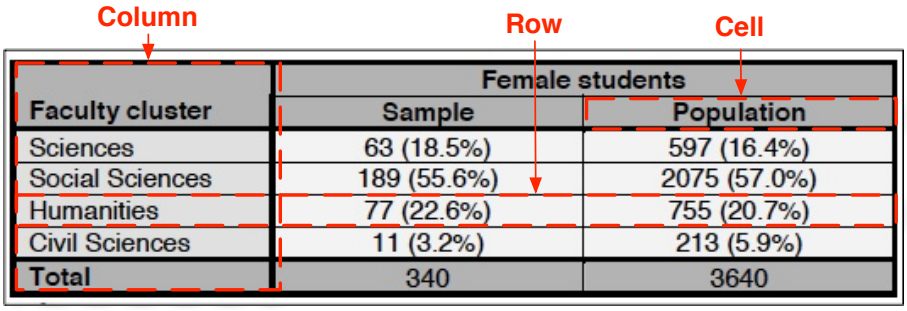


Figure 4.3: Table Segment model

**Physical Table Segments** ($T_{Seg}^{Phy}$): A `Cell` is a `Text Chunk` which is located in a $T_{Loc}^{Phy}$. A `Row` is a sequence of *horizontally aligned* cells, and a `Col` is a sequence of *vertically-aligned* cells. $T_{Seg}^{Phy}$ is a triple $(Cells, Rows, Cols)$ where $Cells$ is set of two or more `Cells`, $Rows$ is a set of one or more `Rows`, $Cols$ is a set of one or more `Cols`. One cell may span multiple cells in a different row/column either horizontally or vertically.

$Cell_i$ and $Cell_j$ are `horizontally aligned` if either $((|Top(Cell_i) - Top(Cell_j)| \sim 0) \wedge (|Bottom(Cell_i) - Bottom(Cell_j)| \sim 0))$ or spanned vertically.

$Cell_i$ and $Cell_j$ are `vertically aligned` if either $((|Left(Cell_i) - Left(Cell_j)| \sim 0) \wedge (|Right(Cell_i) - Right(Cell_j)| \sim 0))$ or spanned horizontally.

$Cell_i$ `spans horizontally` $Cell_j$ if $((Left(Cell_i) - Left(Cell_j) < 0) \wedge (Right(Cell_i) - Right(Cell_j) \geq 0)) \vee ((Left(Cell_i) - Left(Cell_j) \leq 0) \wedge (Right(Cell_i) - Right(Cell_j) > 0))$.

$Cell_i$ `spans Vertically` $Cell_j$ if $((Top(Cell_i) - Top(Cell_j) < 0) \wedge (Bottom(Cell_i) - Bottom(Cell_j) \geq 0)) \vee ((Top(Cell_i) - Top(Cell_j) \leq 0) \wedge (Bottom(Cell_i) - Bottom(Cell_j) > 0))$.

## 4.3 Table Understanding

We now consider the *Table Understanding* part of TEXUS. Understanding a table means going beyond layout features to determine the relationships between the data items. Table understanding consists of two sub-tasks: *Functional Analysis* and *Structural Analysis*.

### Functional Analysis

Functional Analysis identifies the role that each cell plays in the table. Cells either contain data or contain indexes for accessing the data (i.e. table headers). In our model, there are two kinds of headers: *attribute cells* which are normally placed in the table Boxhead, as the top-level headers, and *access cells* placed in the stub in the role of indexes. Data cells are contained in the table body and are the target information in the table. Figure 4.4 gives examples of these.

The functional aspects of cells in a table is described using a `Logical Table Function` model.

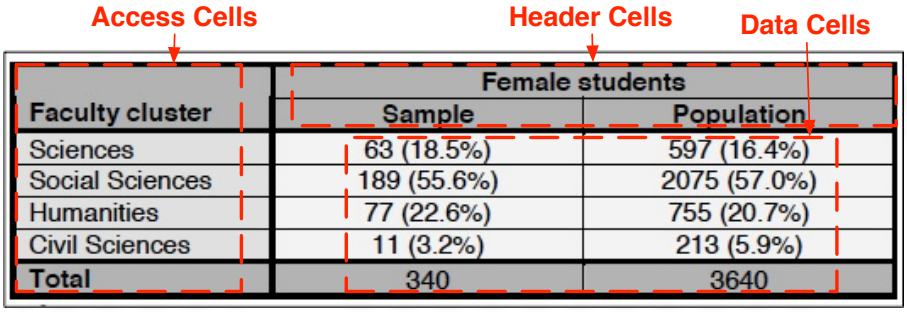


Figure 4.4: Table Function model

**Logical Table Function** ($T^{Log}_{Func}$): $T^{Log}_{Func}$ specifies the role of each cell in a table, and identifies the four functional regions: $BoxHead$, $Stub$, $StubHead$ and $TableBody$. Roles are represented by a set of pairs $(Cell_i, Func_i)$, where $Cell_i$ is an identifier for a cell and $Func_i \in \{Data, Access, Attribute\}$. Each region is defined by the set of cells contained in it.

The following conditions hold on cells in the table:
if $Cell_i \in BoxHead$, $Func(Cell_i) = Attribute$,
if $Cell_i \in \{stub, stubHead\}$, $Func(Cell_i) = Access$
otherwise $Func(Cell_i) = Data$.

A cell can not have more than one Functional role.

### Structural Analysis

Structural Analysis defines the logical relationships between cells, which determines access paths to the data. Tables are essentially multi-dimensional structures which are presented in two dimensions. Reaching a data value in this multi-dimensional space requires following two paths which intersect at the data cell. One path begins from the top-level *Attribute* (top row in $BoxHead$), the other path begins from the $StubHead$ (topmost/leftmost cell of table). This is essentially the Wang abstract table model [30]. In Wang terminology, the top-level header and $StubHead$ are `Categories`. This final output from our end-to-end processing is represented using a `Logical Table Structure` model (or Wang abstract table model). Figure reffig:structural shows examples of the components in such a model.

**Logical Table Structure** ($T^{Log}_{Struct}$): A $T^{Log}_{Struct}$ is a pair $(T^{Struct}_{Access}, F^{Struct}_{Map})$. A $T^{Struct}_{Access}$ is a non-empty set of `Access Path` values, where each `AccessPath` is a hierarchical relationship between an `Attribute` or `Access` cell and their subordinate cells. An `AccessPath` $Access_i$ contains an `Attribute` cell `AttrCell` or `AccCell` cell as `Category` and an associated non-empty list of `SubCategory` values $\langle SC_1 ... SC_n \rangle$. A `SubCategory` $SC_i$ contains an `AttrCell` or `AccCell` cell (and a possibly empty set of `SubCategory` values $\langle SC_{i,1} .. SC_{i,n} \rangle$. A `LeafPathCell` is any `AccCell` or `AttrCell` in a `SubCategory` which has an empty set of `SubCategory` values. $F^{Struct}_{Map}$ is a function that takes a set of access paths and determines the `DataCell` corresponding to those access paths. (Note that examples of access paths were given in Section 3).

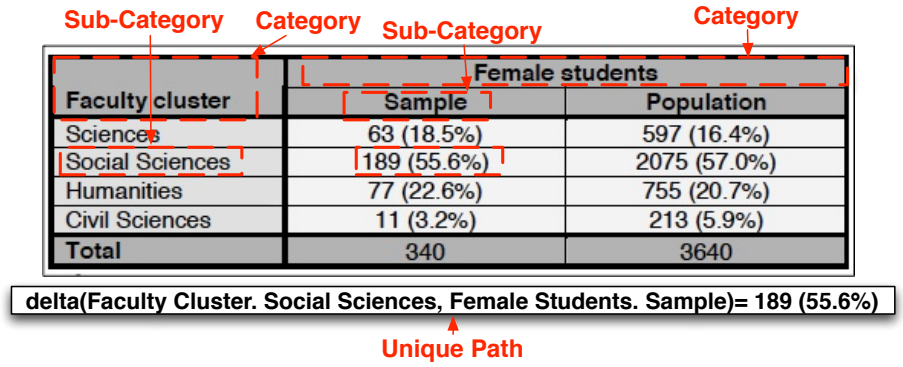


Figure 4.5: Table Structure model

Logical Table Structure models satisfy the following properties: the table must have two or more categories; each category must have a single root header; a category tree cannot contain identical root-to-leaf paths; if a table has $n$ categories, then data cells are accessed via a tuple of $n$ access paths, one through each category tree,

Note that the Wang model does not handle tables where the top row in the *HeadBox* contains more than one cell. We consider a *virtual header* above the top row as a starting point for the access path from the header. Similarly, if a table has an empty *StubHead*, we add a *virtual access* cell to act as the root for the access path from the stub. The names of the virtual cells could either be simple unique identifiers or could be derived based on the names in the header/stub respectively.

Note also that the final data model for our end-to-end processing consists of a set of $T^{Log}_{Struct}$ models, one for each table, along with a set of table metadata entries (discussed in the next section).

In many applications, the logical abstraction obtained from the *Function* and *Structure* views is sufficient to provide effective access to the data. However, other applications may wish to consider domain- and application-specific knowledge. A *Semantic* level allows users to model the meaning of the data contained in the table and to find semantic relationships between table components. This abstraction level is often linked to a domain-specific knowledge base or ontology.

## 4.4 Table Metadata

The above models are built using the information in the tables. One other kind of information, `Table Metadata`, derived from the document context, can also be potentially useful. In particular, the text regions near the table region often contain useful information about the table. There are three major kinds of table metadata:

**Descriptive Metadata** consists of table attributes based on layout, presentation and location, and is intended primarily for indexing the table. Examples of such data include Document Type, Page Number, Caption, etc.

**Structural Metadata** is based primarily on the Function and Structure logical views of the table, and could have potential applications in determining

table similarity. Examples of such data include labels and headers on rows and columns, and the semantic categories behind such labels.

**Semantic Metadata** provides the background semantic structures which are used in interpreting table contents. Examples include knowledge about the domains of data cells and the underlying schema that the table represents.

# 5 Implementation

Through TEXUS, we intend to facilitate a systematic development and reuse of the concepts and their implementation defined within it. The tasks defined in TEXUS can be implemented and utilised as a set of components. Since the models in TEXUS only specify the expected output of each task, the exact details of how the outcome is achieved are left to the developers. They can easily choose compatible table processing standards, leverage existing component implementations and integrate them to build their own solutions, develop new components, or provide alternative implementation of the same components.

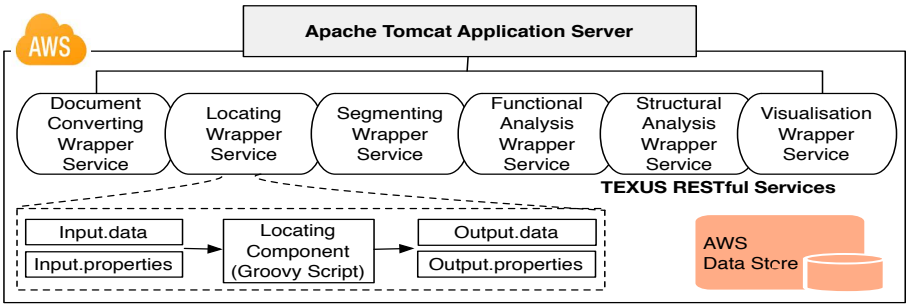


Figure 5.1: Implementing TEXUS Tasks as Web-based Service Component

## 5.1 Overall Design

Figure 5.1 shows our own implementation based on the models in TEXUS. We have implemented each task as a Web-enabled service component. The figure also illustrates that the 'body' part of each component (i.e., implementation of the task) takes two sources of data: an input data model instance and a set of configuration parameters. The outputs are an output data model instance and a set of properties (e.g., header type or error codes if any). To store data model instances, we have chosen to use XML, because (i) it is suitable for describing structured textual information, (ii) it is a platform-independent open standard, and (iii) it is easily transformable into different formats when necessary. For example, we can visualise the output of any components using a simple XSLT[1] script.

Providing the components as services allows the client applications to be built in a flexible manner. For example, a client application can choose to use one component from the available pool, or use a well-known Web service composition technique to wire the multiple components together.

[1]XSLT, `http://www.w3.org/TR/xslt`

## 5.2 Implementation of the Components

We have implemented six components: document converting, locating, segmenting, functional analysis, structural analysis and finally a visualisation component that renders the output of any given component.

**Document Converting.** The purpose of this component is to create an instance of our document model. We first convert a PDF document to an XML document using a well-known conversion utility named '`pdftohtml`[2]' . This utility partitions the document into a sequence of `<page>`s, where each page is a sequence of `<textChunk>`s, and each `<textChunk>` is an XML `TEXT` element with the following attributes: `top` (vertical distance from top of page); `left` (horizontal distance from left edge of page); `width` (width of text chunk); `height` (height of text chunk); `font` (size, family, and color of text chunk).

We then perform further optimisation on the XML before producing output. First, we tag possible table cells containing multi-lines. Second, recognising multiple page columns (to be distinguished from table columns later). Third, tag text lines, properly recognising lines in each page column.

**Locating Component.** Taking an instance of TEXUS document model as input, *Loc* first attempts to separate *table lines* from *normal text lines*. It looks for lines with more than one text chunk as a potential table line. *Loc* then uses the transitions from text lines to (potential) table lines, and vice versa, to determine table boundaries. For potential table lines, our implementation also looks for type patterns (numeric, alphabetic, date, etc.). When a sequence of text chunk categories forms a pattern for that line. A sequence of lines that follows the same, or a similar, type pattern, is a strong candidate for containing the data cells of a table. We also look for spatial patterns. The aim is to determine the left and right boundaries of each table column. Since the text chunks in a column are unlikely to have identical `left` and `right` attributes, we form column extents by considering the left boundaries of chunks in column $i$ and the right boundaries of chunks in column $i + 1$. The sequence of chunk extents determined in this way forms a *spatial pattern* for the data in the table. The Locating component outputs XML file which encloses table lines in `<table>` elements, and adds `pattern` attributes to existing `<line>` elements.

**Segmenting Component.** In this component, the main aim is to detect the inner boundaries of the table as cells, rows and columns. First, we look for dominant table line pattern to determine table rows, and then recognise lines that deviate from the pattern. These lines could be considered potential header lines or uncertain table lines (e.g., summary lines like 'Total'). Using the spatial pattern from Locating, starting from the table's dominant spatial pattern, we build a list of column horizontal boundaries, then scans the table and checks cell boundaries against these, allowing it to both detect spanned cells and, by consider each cell's `top` and `bottom`, to determine the vertical extent of the column. Finally, we determine table cells. In a table row, each text chunk has boundaries and content data type. Most table cells are clearly delimited from surrounding cells, but we handle two special cases: (i) *span cells*: if the extent of a single text chunk extends across multiple columns, it is labelled as a span cell, (ii) *blank cells*: once rows and columns are determined, the boundaries of cells are known. we detect whether and expected cell location has no corresponding

[2] pdftohtml, `sourceforge.net/projects/pdftohtml/`

text chunk and labels it as a blank cell. Segmenting produces an XML file which encloses detected cells in `<td>...</td>` and places the text chunks from each table row in `<tr>..</tr>`.

**Functional Analysis Component.** In this component, the goal is to detect each cell function as data, header or access. A bottom-up and a top-down classification algorithm is used at the same time to detect the boundary of table body containing data cells and the table box head enclosing the header cells. We consider the bottom right most cell in the table boundary as a *data cell*. Then neighbour cells are compared considering spatial and type pattern to find cells with the same functionality. At the same time, the top most row is assumed as a *potential* header line and hierarchical structure along with spatial and type pattern help to detect the similarity of neighbour cells. The reaching point of the two algorithms is considered as the boxhead separation. The left most column is always considered as the *access* column. If we encounter a vertical spanned cell in the left most column, we consider the following column for the access functionality class as well. The functional analysis component adds `function` attribute to each `<td>` element of a table row `<tr>`.

**Structural Analysis Component.** Having detected the function of cells in the table, the structural analysis component provides two paths for each data cell encompassing one unique *Header Path* and one unique *Access Path*. The root of header rows in the boxhead and the root of access columns in the stubhead (either they exist or we add virtual ones) are considered as the starting point. The path is then constructed for each data cell based on the following rules: (i) For header path, it starts from the root and follow the header cells in the top-down fashion to reach the last header cell in the same column with the related data cell, (ii) For access path, it starts from the stubhead and follow all access cell from left-to-right to reach the last access cell in the same row with the related data cell, (iii) The content of the spanning cells are copied to their underlying contained cells, (iv) The empty cells in the path are replaced with a unique identifier, and (v) Header or access cells with the same content are distinguished by adding unique indices. The component adds `HeadePath` and `AccessPath` attributes to each data cell in the output. Also the table metadata in three different categories are added to the table presentation.

**Visualisation Component.** Currently, the primary purpose of this component is for debugging. It shows the output data models in a convenient format. It parses the XML output of a component and provides an HTML representation of the content. We used color coding to present different functional regions in the table and also a tree representation of the access and header paths for structural analysis.

## 5.3 Evaluation

To evaluate our implemented system, we used the known available dataset in the community introduced in 'ICDAR 2013 Table Competition[3]'. The dataset consists of 67 documents with 156 tables, and it is ground-truthed for locating and segmenting tasks. The tables are in different styles and from various domains.

The performance comparison of seven academic systems and four commercial

[3]http://www.tamirhassan.com/dataset.html

products (FineReader, Acrobat, OmniPage, Nitro) are presented in the competition. We follow the same evaluation strategy and metrics for reporting our results against the published results of the competition.

The measures *Completeness* and *Purity* are used over the whole dataset. The measures *Recall* and *Precision* are calculated for the unsuccessful cases, counting individual characters for locating, and considering the adjacency relations for segmenting task.

The ICDAR competition reported that in general, the commercial systems performed better than academic ones. Since the details of the algorithms used by commercial systems are not publicly available, it can not be said whether their advantage originates from a better approach to the problem or from having access to a large amount of data which allow them to fine-tune the heuristics in their system overtime.

Overall, TEXUS performed better than most of academic system (except the Nurminen) in all three different evaluations, and even better than Acrobat and Nitro in the commercial system in table extraction. Our performance seems acceptable in locating, however we have difficulties in segmenting especially when the header hierarchy structures are very complex. In the following, we discuss the results of locating and segmentig separately.

Table 5.1 shows the results of locating task. Our system has a problem in locating 'small tables' (tables with less than four rows), and tables with summary lines in the middle of the table. However, it is worth pointing out that we performed well in unruled table. It is mentioned that most of the currently available systems do not perform well with unruled tables. Locating floating tables are another category that TEXUS handles well.

Table 5.1: Result of the Locating task

| **System** | **Per-document average** | | | **Table Found Total=156** | |
|---|---|---|---|---|---|
| | Recall | Precision | F-meas. | Complete | Pure |
| FineReader | 0.9971 | 0.9729 | 0.9848 | 142 | 148 |
| OmniPage | 0.9644 | 0.9569 | 0.9606 | 141 | 130 |
| Silva | 0.9831 | 0.9292 | 0.9554 | 149 | 137 |
| Nitro | 0.9323 | 0.9397 | 0.936 | 124 | 144 |
| Nurminen | 0.9077 | 0.921 | 0.9143 | 114 | 151 |
| Acrobat | 0.8738 | 0.9365 | 0.904 | 110 | 141 |
| **TEXUS** | **0.9023** | **0.8832** | **0.8926** | **114** | **138** |
| Yildiz | 0.853 | 0.6399 | 0.7313 | 100 | 94 |
| Stoffel | 0.6991 | 0.7536 | 0.7253 | 79 | 66 |
| Liu et al.2 | 0.3355 | 0.8836 | 0.4864 | 0 | 29 |
| Hsu et al. | 0.4601 | 0.3666 | 0.408 | 39 | 95 |
| Fang et al. | 0.2697 | 0.7496 | 0.3967 | 28 | 41 |
| Liu et al.1 | 0.2207 | 0.8885 | 0.3536 | 0 | 25 |

Table 5.2 shows the results of segmenting task based the correct results of locating as input. We performed well in segmenting the vertical spanned cells and multi-line cell boundary detection. However, in some cases when tables have very irregular alignments in cells content, we ended up adding extra blank column in segmenting.

Table 5.2: Result of the Segmenting task

| | **Per-document average** | | |
|---|---|---|---|
| **System** | Recall | Precision | F-meas. |
| Nurminen | 0.9409 | 0.9515 | 0.9460 |
| **TEXUS** | **0.8423** | **0.8102** | **0.8259** |
| Silva | 0.6401 | 0.6144 | 0.6270 |
| Hsu et al. | 0.4811 | 0.5704 | 0.5220 |

Finally, Table 5.3 shows the results of table extraction as a sequence of table locating and segmenting (i.e., segmenting component directly taking input from locating without corrections). Our performance is acceptable compared with other academic systems.

The task-based design of our system provides the opportunity to evaluate and refine the components separately and then try a new composition to improve the performance. It should be mentioned that participants in the ICDAR competition had the opportunity to test their systems beforehand on a practice dataset for bug fixing or training. We did not have access to that dataset and did not use the test dataset before the performance evaluation.

Table 5.3: Result of Table Extraction

| | **Per-document average** | | |
|---|---|---|---|
| **System** | Recall | Precision | F-meas. |
| FineReader | 0.8835 | 0.871 | 0.8772 |
| OmniPage | 0.838 | 0.846 | 0.842 |
| Nurminen | 0.8078 | 0.8693 | 0.8374 |
| **TEXUS** | **0.7823** | **0.8071** | **0.7945** |
| Acrobat | 0.7262 | 0.8159 | 0.7685 |
| Nitro | 0.6793 | 0.8459 | 0.7535 |
| Silva | 0.7052 | 0.6874 | 0.6962 |
| Yildiz | 0.5951 | 0.5752 | 0.585 |

| TEXUS Name | Synonym Terms |
|---|---|
| Locating | table identification [19], table detection [7], table area selection [4], table boundary detection [18], table spotting [7] |
| Segmenting | table recognition [20], table decomposition[19], table text blocks discovery [13], table internal structure detection[28] |
| Functional Analysis | table header detection [27], table format verification [24], table augmentation[23] |
| Structural Analysis | table factoring [15], table abstraction [30], detecting table read-wise pattern [32]. table access structure detection[16] |

Table 6.1: TEXUS Tasks and the Synonyms

# 6 Further Discussions

Besides providing a basis to design and develop a table processing system, TEXUS may potentially be used as a common ground for evaluating the performance of table processing systems. Although detailing this idea further is our ongoing work, we present a couple of issues in table procesing system evaluation and the relevant concepts in TEXUS for discussion.

## 6.1 Appropriate Points for Evaluation

It is difficult to adequately compare the quality of results of different table processing system even when they seem to work on the same task [2, 8, 10]. We discuss this issue from two perspectives. First, different terms are used to describe systems that perform table extraction tasks belonging in the same scope. For example, there is much work focused on locating tables in a document. In these work, there is a wide variation in terminology to describe their goals, e.g. *table identification*, *table detection*, *table spotting*, etc. In the terms defined in TEXUS, we can describe all of these as *Table Locating*. Since TEXUS defines the atomic tasks involved in a table processing system, each system's goal can either be mapped to one task or a composition of several tasks. Table 6.1 gives examples of terms that appear in the table processing literature and their mapping to TEXUS atomic tasks.

Some researchers suggest that the appropriate point to evaluate a table processing system is after the final result is produced. For example, if the endpoint is to extract particular information from tables, evaluation is based on how closely the extracted information matches what was expected [25]. However, others claim that, since a table processing system involves many steps, the overall performance of the system often relies on how well the intermediate steps interact, and measuring the performance of intermediate steps is important [20]. In fact, as we have shown above, it would be possible to describe the goals of

many table processing systems in terms of logically separated steps. Evaluating table processing at each "step" can also provide a chance to improve each step independently [3], which may lead to increased performance overall. We believe TEXUS, through the well-defined tasks along with concrete data models, can contribute to providing a useful framework to objectively evaluate the systems.

### 6.2 Proper Unit of Measurement

The characteristics of the appropriate metrics which satisfy the needs of table processing tasks is another discussion in the community. One of the considerations is that the granularity level of the elements at input is not the same as that of output in a table processing task. Different suggestions were made regarding this issue. Silva [2] proposed two new metrics: completeness and purity. Long [21] suggested multi-level evaluation methodology.

However, there is not yet any set unit of measurement in task-based evaluation. Take the locating task as an example, Liu et. al [19] evaluated their system in terms of lines, Wang [31] used cells and Chen [1] considered full tables. In the ICDAR'13 table competition [9], they defined the measurement unit for locating at an individual character level. However, the results showed that it was not a proper choice for some cases. This approach gave more weight to the parts of tables that have more characters (e.g., there was a system having a very good precision at the character level, but did not manage to locate even one table completely). Given that the main aim of the locating task, recognising table boundaries should be valued more than say, recognising text areas within a table.

We believe that TEXUS could provide a basis for task-based evaluation measurement metrics, because the table elements (e.g., table boundaries, rows, columns and cells) which may be relevant for measuring performance are explicitly defined in the tasks and their data models. Following the locating task example, we may consider the "text chunks" as a measuring unit, since cells are the smallest meaningful unit of data in a table structure.

## Bibliography


[1] Hsin-Hsi Chen, Shih-Chung Tsai, and Jin-He Tsai. Mining tables from large scale html texts. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 166–172. Association for Computational Linguistics, 2000.

[2] Ana Costa e Silva. *Parts that add up to a whole: a framework for the analysis of tables*. PhD thesis, The University of Edinburgh, 2010.

[3] Ana Costa e Silva. Metrics for evaluating performance in document analysis: application to tables. *International Journal on Document Analysis and Recognition (IJDAR)*, 14(1):101–109, 2011.

[4] Ana Costa e Silva, Alípio Jorge, and Luís Torgo. Automatic selection of table areas in documents for information extraction. In *Progress in Artificial Intelligence*, pages 460–465. Springer, 2003.

[5] Ana Costa e Silva, Alípio Jorge, and Luís Torgo. Design of an end-to-end method to extract information from tables. *International Journal of Document Analysis and Recognition*, 8(2-3):144–171, 2006.

[6] David W Embley, Daniel Lopresti, and George Nagy. Notes on contemporary table recognition. In *Document Analysis Systems VII*, pages 164–175. Springer, 2006.

[7] Jing Fang, Liangcai Gao, Kun Bai, Ruiheng Qiu, Xin Tao, and Zhi Tang. A table detection method for multipage pdf documents via visual seperators and tabular structures. In *Document Analysis and Recognition (ICDAR)*, pages 779–783. IEEE, 2011.

[8] Jing Fang, Xin Tao, Zhi Tang, Ruiheng Qiu, and Ying Liu. Dataset, ground-truth and performance metrics for table detection evaluation. In *Document Analysis Systems (DAS), 2012 10th IAPR International Workshop on*, pages 445–449. IEEE, 2012.

[9] Max Gobel, Tamir Hassan, Ermelinda Oro, and Giorgio Orsi. ICDAR 2013 table competition. In *12th International Conference on Document Analysis and Recognition (ICDAR'13)*, pages 1449–1453. IEEE, 2013.

[10] Jianying Hu, Ramanujan Kashi, Daniel Lopresti, George Nagy, and Gordon Wilfong. Why table ground-truthing is hard. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 129–133. IEEE, 2001.

[11] Jianying Hu and Ying Liu. Analysis of documents born digital. *Handbook of Document Image Processing and Recognition*, pages 775–804, 2014.

[12] Matthew Hurst. *The interpretation of tables in texts*. PhD thesis, The University of Edinburgh, 2000.

[13] Matthew Hurst. Layout and language: Exploring text block discovery in tables using linguistic resources. In *International Conference on Document Analysis and Recognition*, pages 523–527, 2001.

[14] Piyushee Jha and George Nagy. Wang notation tool: Layout independent representation of tables. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.

[15] Dongpu Jin. An algebraic approach to building category parse trees for web tables. *2012 NCUR*, 2013.

[16] Thomas Kieninger and Andreas Dengel. Applying the t-recs table recognition system to the business letter domain. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 518–522. IEEE, 2001.

[17] Min-Hyung Lee, Yeon-Seok Kim, and Kyong-Ho Lee. Logical structure analysis: From html to xml. *Computer Standards & Interfaces*, 29(1):109–124, 2007.

[18] Ying Liu, Kun Bai, Prasenjit Mitra, and C Lee Giles. Improving the table boundary detection in pdfs by fixing the sequence error of the sparse lines. In *10th International Conference on Document Analysis and Recognition (ICDAR'09)*, pages 1006–1010. IEEE, 2009.

[19] Ying Liu, Prasenjit Mitra, and C Lee Giles. Identifying table boundaries in digital documents via sparse line detection. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 1311–1320. ACM, 2008.

[20] Vanessa Long. *An Agent-Based Approach to Table Recognition and Interpretation*. PhD thesis, Macquarie University Sydney, Australia, 2010.

[21] Vanessa Long, Steve Cassidy, and Robert Dale. A multi-level table evaluation method for plain text documents. In *Extended Abstracts of the 7th International Association for Pattern Recognition Workshop on Document Analysis Systems (DAS 2006)*, pages 21–24, 2006.

[22] Daniel Lopresti and George Nagy. A tabular survey of automated table processing. In *Graphics Recognition Recent Advances*, pages 93–120. Springer, 2000.

[23] George Nagy, Raghav Padmanabhan, RC Jandhyala, W Silversmith, and MS Krishnamoorthy. Table metadata: Headers, augmentations and aggregates. In *Ninth IAPR International Workshop on Document Analysis Systems*, 2010.

[24] George Nagy and Mangesh Tamhankar. Vericlick: an efficient tool for table format verification. In *IS&T/SPIE Electronic Imaging*, pages 1–9. International Society for Optics and Photonics, 2012.

[25] Ermelinda Oro and Massimo Ruffolo. Xonto: An ontology-based system for semantic information extraction from pdf documents. In *20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'08)*, volume 1, pages 118–125. IEEE, 2008.

[26] Raghav Krishna Padmanabhan, Ramana Chakradhar Jandhyala, Mukkai Krishnamoorthy, George Nagy, Sharad Seth, and William Silversmith. Interactive conversion of web tables. In *Graphics Recognition. Achievements, Challenges, and Evolution*, pages 25–36. Springer, 2010.

[27] Sharad Seth, Ramana Jandhyala, Mukkai Krishnamoorthy, and George Nagy. Analysis and taxonomy of column header categories for web tables. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 81–88. ACM, 2010.

[28] Asif Shahab, Faisal Shafait, Thomas Kieninger, and Andreas Dengel. An open approach towards the benchmarking of table structure recognition systems. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 113–120. ACM, 2010.

[29] Cui Tao and David W Embley. Automatic hidden-web table interpretation, conceptualization, and semantic annotation. *Data & Knowledge Engineering*, 68(7):683–703, 2009.

[30] Xinxin Wang. *Tabular abstraction, editing, and formatting.* PhD thesis, University of Waterloo, 1996.

[31] Yalin Wang and Jianying Hu. Detecting tables in html documents. In *Document Analysis Systems V*, pages 249–260. Springer, 2002.

[32] Yingchen Yang. *Web table mining and database discovery.* PhD thesis, Simon Fraser University, 2002.

[33] Richard Zanibbi, Dorothea Blostein, and James R Cordy. A survey of table recognition. *Document Analysis and Recognition*, 7(1):1–16, 2004.