From Tables to Frames

A. Pivk [a,b,*], P. Cimiano [b], Y. Sure [b],

[a] *Jozef Stefan Institute, Department of Intelligent Systems, Ljubljana, Slovenia*

[b] *Institute AIFB, University of Karlsruhe, Karlsruhe, Germany*

Abstract

Turning the current Web into a Semantic Web requires automatic approaches for annotation of existing data since manual approaches will not scale in general. We here present an approach for automatic generation of F-Logic frames out of tables which subsequently supports the automatic population of ontologies from table-like structures. The approach consists of a methodology, an accompanying implementation and a thorough evaluation. It is based on a grounded cognitive table model which is stepwise instantiated by our methodology.

Key words: Table Structure, Table Modeling, Knowledge Frame, Ontology Generation, Web Mining

1 Introduction

Turning the current Web into a Semantic Web requires automatic approaches for annotation of existing data since manual annotation approaches such as presented in [9] will not scale in general. More scalable (semi-)automatic approaches known from ontology learning (cf. [16]) deal with extraction of ontologies from natural language texts. However, a large amount of data is stored in tables which require additional efforts.

We here present an approach for automatic generation of F-Logic frames [14] out of tables which subsequently supports the automatic population of ontologies from table-like structures. Even successful search engines on the Web

* Corresponding author.
Email addresses: aleksander.pivk@ijs.si (A. Pivk), cimiano@aifb.uni-karlsruhe.de (P. Cimiano), sure@aifb.uni-karlsruhe.de (Y. Sure).
URL: http://dis.ijs.si/sandi (A. Pivk).

currently do not make the content of tables searchable to users. Applying our approach e.g. allows for querying over a heterogeneous set of table-like structures.

Our approach consists of a methodology, an accompanying implementation and a thorough evaluation. It is based on a grounded cognitive table model which is stepwise instantiated by our methodology. In practice it is hard to cover every existing type of a table. We identified a couple of most relevant table types which were used in the experimental setting during the evaluation of our approach.

In this paper we use HTML tables as examples. We would like to point out that the methodology can be applied to any kind of table-like structure. The implementation is almost as generic as the methodology. To apply it for other formats than HTML one would only need to adapt the implementation for the first methodological step (cf. Figure 1).

The paper is structured as follows. In the next Section 2 we first introduce the grounding table model which forms the base for our stepwise approach to generate frames out of tables. Subsequently we explain each step in detail and show relevant substeps. In Section 3 we present a thorough evaluation of the accompanying implementation. Before concluding and giving future directions, we present related work.

2 Methodological Approach

Linguistic models traditionally describe natural language in terms of syntax, semantics and pragmatics. There also exist models to describe tables in similar ways (cf. [10,11]) where tables are analyzed along the following dimensions: (i) **Graphical** – the image level description of the pixels, lines and text or other content areas, (ii) **Physical** – the description of inter-cell relative location, (iii) **Structural** – the organization of cells as an indicator of their navigational relationship, (iv) **Functional** – the purpose of areas of the tables in terms of data access, and (v) **Semantic** – the meaning of text in the table and the relationship between the interpretation of cell content, the meaning of structure in the table and the meaning of its reading.

Our approach builds on the model described above. However, we will not consider the *graphical* dimension as no image processing will be necessary. Regarding the *physical* dimension, we process the tables encoded in HTML format in order to get a physical model of the table. In principle it can be seen as a graph describing the cells being connected together. In order to capture the *structural* dimension of the table, further processing is necessary (i) to

determine the orientation of the table, i.e. top to down or left to right, and, (ii) to discover groups of cells building logical units. When talking about the function of a table, Hurst [11] distinguishes between two functional cell types *access* and *data*. Cells of type *data* are the ones we are interested when reading a table and which contain the actual information, while cells of type *access* determine the path to follow in the table in order to find the *data* cell of interest. Further, he distinguishes *local* (looking for one specific *data* cell) from *global* (comparing the value of different *data* cells) search in a table. In our approach we describe the *functional* dimension of a table in order to support *local* search. Such a functional description requires (i) finding all the *data* cells in a table as well as (ii) all the *access* cells to reach a given *data* cell of interest. In terms of database terminology, we need to find the keys for a certain field in the table. In our approach we distinguish between two *functional types* of cells: A(ttribute)-cells and I(nstance)-cells. A-cells describe the conceptual nature of the instances in a column or row. I-cells represent instances of the concepts represented by a certain A-cell. I-cells can have the two *functional roles* described by Hurst, i.e. they can play the role of *data* or *access* cells.

Table 1
Example of a possible table, found in [1].

Tour Code			DP9LAX01AB	
Valid			01.05. - 30.09.04	
Class/Extension			Economic	Extended
Adult	P R	Single Room	35,450	2,510
		Double Room	32,500	1,430
		Extra Bed	30,550	720
Child	I C E	Occupation	25,800	1,430
		No occupation	23,850	720
		Extra Bed	22,900	360

Regarding the *semantic* description we follow a different paradigm as Hurst. Instead of adopting the relational model [2], we describe the semantics of a table in terms of F-Logic frames [14]. F-Logic combines the intuitiveness of modeling with frames with the expressive power of logic. Furthermore, existing F-Logic inference engines such as Ontobroker [5] allow later on e.g. for processing and query answering. Therefore it was our primary choice as representation language.

We briefly introduce our running example. As input we use Table 1 which is taken from the tourism domain and is (roughly speaking) about room prices. The ideal description in terms of an F-Logic frame of this table, i.e. the output

after applying our approach, could look as follows:

Tour[TourCode => ALPHANUMERIC;
Validity => DATE;
EconomicPrice(PersonType,RoomType) => LARGE_NUMBER;
ExtendedPrice(PersonType,RoomType) => LARGE_NUMBER].

By resorting to F-Logic we are thus able to describe the semantics of tables in a model-theoretic way. Furthermore, as required by Hurst, the frame makes explicit (i) the meaning of cell contents (ii) the functional dimension of the table, and (iii) the meaning of the table abstracting from its particular structure. In this line, different tables with different structures but identical meaning would be described by one and the same frame. In what follows we describe how we process the table in order to yield intermediate descriptions of a table along the dimensions described above as well as how as a last step the table is translated into a F-Logic frame.

As depicted in Figure 1 our methodology consists of four main steps. For each building block of the table model there exists a corresponding methodological step to create this part of the table model. In the following subsections we will describe all steps in detail.

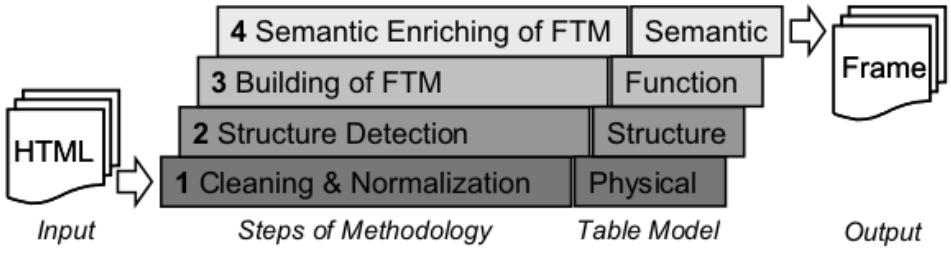


Fig. 1. Building blocks of the methodology.

2.1 Cleaning and Normalization

Web documents are often very noisy in a sense that their syntactic structure is incorrect. Web browsers (i.e. Opera) are capable of dealing with such poorly structured, inconsistent, and incomplete documents. In order to remove this noise we perform the following two steps: a) syntactic correction/cleaning, and b) normalization.

First, we assume that documents are represented with the DOM [1] (Document Object Model). A DOM tree is an ordered tree, where each node is either an element or a text node. An element node includes an ordered list of zero to many child nodes, and contains a string-valued tag (such as table, h1 or title)

[1] `http://www.w3.org/DOM/`

and zero to many string-valued attributes (such as href or src). A text node normally contains a single text string and has no child nodes.

In the two substeps we construct an initial table model out of an updated DOM tree. In order to clean the code and make it syntactically correct, we employ the CyberNeko HTML Parser [2]. The normalization step of the rendered 'table' element node in the DOM tree is only necessary, when an explicit child node's attribute, such as rowspan or colspan, indicates multiple row or column spanning cells, where the actual total number of rows or columns is lower than the attribute value. In both steps our system updates the corresponding DOM subtrees accordingly.

Table 2 shows the final reformulation of the example in Table 1, where cleaning has been performed and copies of cells with rowspan and rowspan attributes have been properly inserted into the matrix structure.

2.2 Structure Detection

2.2.1 Assignment of functional types and probabilities to cells.

In the initial pass over the table element node (of the DOM tree), we convert a sub-tree into a matrix structure, which is populated by cells according to its layout information. During this step the text of each cell is tokenized, and each token is assigned a *token type* according to the tree (leaves) presented in Figure 2. At the same time, we assign each cell in the rendered table a *functional type* and a probability of the type. By default, a cell is assigned no functional type (probability value equals zero), or I-cell type in case the cell includes only/mostly tokens, recognized as dates, currencies, or numerical values. According to this ratio, a probability is calculated. Finally, we assume that the cell in the lower-right corner is always an I-cell, and the cell in the upper-left corner is an A-cell. Therefore we assign those two cells the types, regardless of their content, with probability one.

2.2.2 Detecting table orientation.

One problem related to the interpretation of a table is that its logical orientation is a priori not clear. In fact, when performing a local search on a table, the data of interest can be either ordered in a top-to-down (vertical orientation) or left-to-right manner (horizontal orientation). For example, in figure 1 the relationship "*Tour Code, DP9LAX01AB*" reads left-to-right, but price values of an attribute "*Economic Class*" appear top-to-down. When trying to

[2] `http://www.apache.org/andyc/neko/doc/html`

Table 2
Table 1 after cleaning and normalization step.

Tour Code	Tour Code	Tour Code	DP9LAX01AB	DP9LAX01AB
Valid	Valid	Valid	01.05 - 30.09.04	01.05 - 30.09.04
Class/Ext.	Class/Ext.	Class/Ext.	Economic	Extended
Adult	PRICE	Single Room	35,450	2,510
Adult	PRICE	Double Room	32,500	1,430
Adult	PRICE	Extra Bed	30,550	720
Child	PRICE	Occupation	25,800	1,430
Child	PRICE	No occupation	23,850	720
Child	PRICE	Extra Bed	22,900	360

determine the table orientation we rely on the similarity of cells. The intuition here is that, if rows are similar to each other, then orientation is vertical and on the contrary, if columns are similar, then interpretation is horizontal.

In order to calculate the differences among rows and columns of the table, we need first to define how to calculate the difference between two cells. For this we represent a cell as a vector c of *token types* of all the tokens in the cell. Henceforth, c_i will denote the i-th component of the vector c, corresponding to the token type of the i-th token in the cell. Furthermore, $|c|$ will denote the size of the vector. The token types we consider are given in Figure 2. They are ordered hierarchically thus allowing to measure the distance δ between two different types as the number of edges between them. For example, the numeric type is hierarchically divided into categories that include range information about numbers, i.e. LARGE_NUM (≥ 10.000), MED_NUM ($100 \leq n < 10.000$), SMALL_NUM (< 100). This representation is very flexible and can be extended to include domain specific information. In particular, the 'composed' node includes two nodes containing such information.

In our example, we determine m to be five ($m = 5$). The orientation is calculated by comparing first $m - 1$ columns to the column m and last $m - 1$ rows to the last row. The difference among columns is greater than among the rows, hence the orientation is set to vertical.

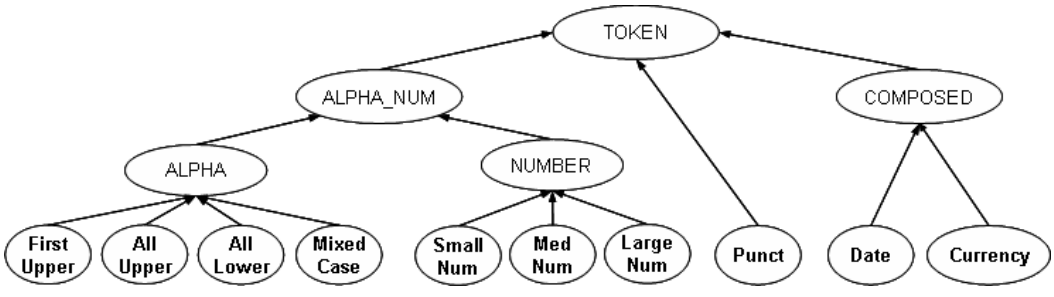


Fig. 2. Hierarchy of token types

Now when comparing the vectors of two cells, we compare the token types with same indices in case the vectors have equal length; otherwise, we calculate the

distance for the left-side tokens (tokens aligned at the head) and for the right-side tokens (tokens aligned at the tail). The distance is in both cases also normalized.

$$\delta_{cells}(c_P, c_Q) := \begin{cases} \frac{v}{2*u} \left(\sum_{i=1}^{u} \delta(c_{P_i}, c_{Q_i}) + \sum_{i=w}^{v} \delta(c_{P_i}, c_{Q_i}) \right) & \text{if } u \neq v \\ \\ \frac{1}{u} \sum_{i=1}^{u} \delta(c_{P_i}, c_{Q_i}) & \text{if } u = v \end{cases} \quad (1)$$

where $u = min(|c_P|, |c_Q|)$, $v = max(|c_P|, |c_Q|)$ and $w = v - u + 1$. Now given a table with r rows and s columns, the total distance (Δ_{cols}) between columns is calculated by summing up the distance between the last column and each of the preceding $m - 1$ columns, where $m = min(r, s)$, i.e.

$$\Delta_{cols} = \sum_{i=1}^{m-1} \delta_{cols}(col_{s-i}, col_s) \quad (2)$$

$$\delta_{cols}(col_p, col_q) = \sum_{i=m}^{s} \delta_{cells}(c_{i,p}, c_{i,q}) \quad (3)$$

where $c_{x,y}$ is the cell in row x and column y.

The total distance (Δ_{rows}) between rows is by analogy calculated by summing up the distance between the last row and each of the $m - 1$ preceding rows:

$$\Delta_{rows} = \sum_{i=1}^{m-1} \delta_{rows}(row_{r-i}, row_r) \quad (4)$$

$$\delta_{rows}(row_p, row_q) = \sum_{i=m}^{r} \delta_{cells}(c_{p,i}, c_{q,i}) \quad (5)$$

Here we only compare equal number of rows and columns, starting at the lower-right corner, thus optimizing the number of comparisons (not all the rows and columns to the top of the table need to be compared). Finally, to determine the orientation of the table, we compare both results. If the distance between columns is greater than among rows ($\Delta_{cols} > \Delta_{rows}$), orientation is set to *vertical* (top-to-down). On the other hand, if the distance between columns is lower than among rows ($\Delta_{cols} < \Delta_{rows}$), then orientation is set to *horizontal* (left-to-right). In the last case, where the two results are equal, orientation is assigned the default, i.e. *vertical*.

1. Initialize logical units and regions
2. Learn string patterns of regions
for all logical units
  do while (logical unit is not uniform)
    3. Choose the best coherent region
    4. Normalize logical unit

Fig. 3. Algorithm for discovery of regions.

2.2.3 Discovery of regions.

Definition 1 (Logical unit) *A logical unit is a part of a table produced by a horizontal split in case of vertical orientation or by a vertical split in case of horizontal orientation.*

Definition 2 (Region) *A region is a rectangular area of a table consisting of cells with the same functional type. Regions cannot extend over multiple logical units and can therefore only appear within one logical unit.*

Here we will present a step-by-step algorithm for discovery of regions in tables. Pseudocode of the algorithm is given in Figure 2.2.3.

(1) *Initialize logical units and regions.* Note that a table is by definition one logical unit. First, the system tries to split a table into several *logical units*. In particular, when table orientation is column-wise (vertical), the horizontal split is done at every row containing cells spanning multiple columns, or when dealing with row-wise (horizontal) orientation, vertical split is done at every column containing cells spanning multiple rows. Consecutive logical units may then be merged if their layout structure is equal. Over-spanning cells of type I-cell also represent a sign for a split.

For example, Table 1 has three logical units. The first logical unit is extending over the first two rows, the second one over the third row, and the third one over the rest of the table. The first two rows have an over-spanning cell with functional type I-cell and are grouped into one logical unit because their layout structure is equal. A third row has a cell spanning multiple columns, and the rest is grouped into one logical unit.

Once splitting is over, the region initialization step begins. The system starts at a lower-right corner and moves according to its orientation towards upper-left corner over all logical units, thus generating all distinct initial regions. The cell c_N is added to a region r if the following conditions apply (otherwise a new region is created):

(a) the cell c_N is within the same logical unit as other cells,
(b) its size is equal to the size of cells already in the region, and

(c) it keeps the distance among cells in r within a treshold value:

$$\delta_{cells}(c_N, c_{1(r)}) \leq 2 \tag{6}$$

where the value of 2 reflects a significant token type change according to the hierarchy in figure 2.

(2) *Learn string patterns for regions.* For each region r we learn a set P_r of significant patterns, which are sequences of token types and tokens, describing the content of a significant number of cells. The positive property of a pattern is that it generalizes over data strings within regions, which, at the end of table transformation, reflects a generalization of possible concept instance values.

The patterns are of two types: the first represents the content of cells from left-to-right (forward) and the second from right-to-left (backward). The pattern 'FIRST_UPPER Room' for example covers the cells 'Single Room' and 'Double Room'. In our example, the coverage of this pattern is 1/3 for the region extending over the last six rows of the third column. For the purpose of pattern construction we have implemented the DATAPROG algorithm, which is described in [15] together with a detailed pattern learning process. In case there are not enough examples (less than 20) to statistically choose the most significant patterns, only the most specific (having their coverage over the treshold value) are chosen.

Before entering the loop (see algorithm in Figure 2.2.3), the system checks the uniformity of every logical unit. In our case, a logical unit is uniform when it consists of logical sub-units and each sub-unit includes only regions of the same size and orientation. Only the units that are not uniform are further processed within the following steps of the loop.

(3) *Choose the best coherent region.* The best region is used to propagate and normalize neighboring (non-uniform) regions and consequently the logical unit itself. The best region r_{max} is selected according to the formula $\Phi_{r_{max}} = max_{r \,\mathrm{in}\, l}\phi_{r,l}$, which is calculated by the following equation:

$$\phi_{r,l} := \left(\frac{|r|}{|l|} + \frac{1}{|r|} \sum_{c \,\mathrm{in}\, r} Prob(c) + \frac{1}{|r|*|P_r|} \sum_{p \in P_r} covers(p, r) \right) \tag{7}$$

where l denotes a *logical unit*, r denotes a *region* in the unit, c denotes *cells* in the region, and P_r is the set of *significant string (forward and backward) patterns* for the region as described above. The function $covers(p, r)$ returns a number of cells covered by pattern p in region r. According to the above formula, the selected region maximizes the sum of averaged region size (1st operand of the sum), averaged cell probabilities (2nd operand) and averaged pattern coverage over a particular region (3rd operand).

(4) *Normalize neighboring regions of the best region.* The intuition here is to use the best region as a propagator for other regions in their normalization process. First, the system selects (based on the orientation) all neighboring regions, i.e. those that appear in the same rows (left/right) for column-wise orientation, or in same columns (up/down) for row-wise orientation. Now, two possibilities exist: (a) neighboring regions within a common column/row (orientation dependent) do not extend over the boundaries of the best region. In this case, the solution is straightforward, because the 'new' region is extended in a way to cover all common column/row regions. (b) neighboring regions within a common column/row do extend over the boundaries of the best region. In this case, the best region is extended accordingly, and this step repeated.

The logical unit is being processed within the loop as long as the system is not able to divide it into logical sub-units, where each sub-unit includes only regions of the same size and orientation (uniformity condition). Note that string patterns, probabilities and functional types of normalized regions are also updated in every iteration. Finally, in this way all logical units are being normalized and prepared for further processing.

2.3 Building of a Functional Table Model

The key step of translating a table into a frame is building a model of the functional dimension of the table. This model is called *Functional Table Model* (FTM) and essentially arranges regions of the table in a tree, whereby the leaves of the tree are all the regions consisting exclusively of I-cells. Most importantly, in the FTM these leaves are assigned their functional role, i.e. *access* or *data*, and semantic labels as described in Section 2.4.1.

The construction of the FTM proceeds bottom up: we start with the lowest logical unit in the table and proceed with further logical units towards the top. For each logical unit in question we first determine its type. There are three possibilities: (a) the logical unit consists only of A-cells, in which case all its regions will be turned into inner nodes of the tree and thus connected to some other nodes in the tree, (b) it consists only of I-cells, in which case they will constitute leaves and will be connected to appropriate inner nodes, and (c) it consists of I-cells and A-cells, in which case we determine the logical separation between them by taking the uniformity condition into account.

In some cases a special *connection node* (see Figure 4) needs to be inserted into the tree. This occurs when we encounter a logical unit that reflects a split in the table, in particular when a previous logical unit contained only A-cells, but the present logical unit again contains I-cells. In such cases, we

check (described later in this paragraph) if reading orientation of the present logical unit is different from the previous one and needs to be changed. If this is true, the logical unit needs to be recalculated, as described in Section 2.2.3. For example, the first logical unit (first two rows) in Table 1 has four regions (each 'logical' cell) and there is no logical unit on top of it. So, if the orientation was vertical (i.e. like in lower logical unit), there would be no inner node (consisting of A-cells) to connect the I-cells to. Thus orientation has to be changed from vertical to horizontal for this logical unit.

As already mentioned above, each region in a leaf position is assigned its corresponding functional role. The role *access* is assigned to all consecutive regions (starting at the left subnodes of a subtree) together forming a unique identifier or key in the database terminology. The rest of the leaf nodes in the subtree get assigned the role *data*.

When all logical units have been processed, we connect the remaining unconnected nodes to a root node. For example, the FTM constructed out of our running example is depicted in Figure 4.

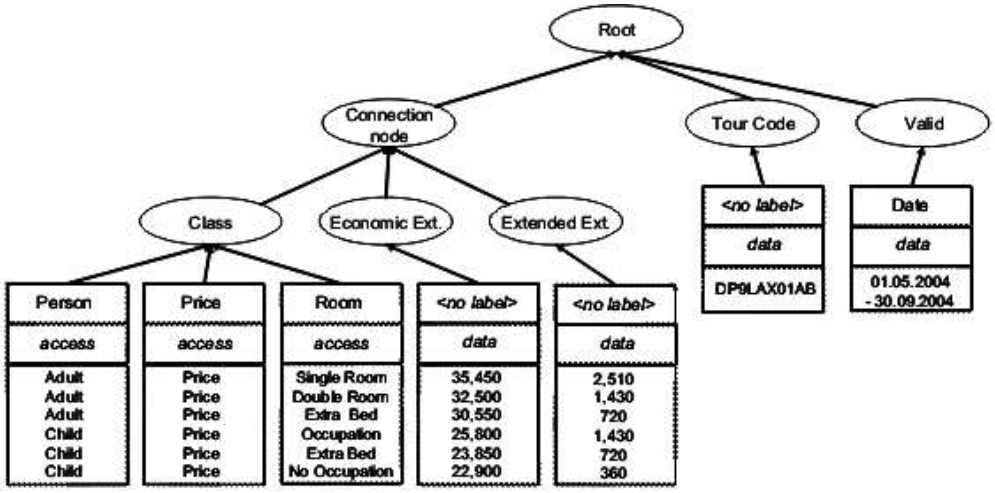


Fig. 4. A functional table model (FTM) of the running example (Table 1) with square components representing I-cells and rounded components representing A-cells.

After the FTM is constructed, we examine if there are any multi-level (at least two levels of inner A-cell nodes) subtrees that might be merged. The candidate subtrees for merging must have the same tree structure (same number of levels and nodes on each level) and at least one level of matching A-cells. If there are any candidates that fulfill the requirements, we perform a process called *recapitulation*, where we merge the nodes at same positions in both subtrees. As we only require one level of matching A-cells, there might be some A-cells that do not match. For every such case, the following steps are taken: (a) find a semantic label of a 'merged' A-cell node (described in Section 2.4.1), (b)

connect the 'merged' A-cell to a new leaf node, which is populated by the A-cell contents of merged nodes, and (c) assign the functional role of the new leaf node to *access*. In this way we check and merge all possible multi-level subtrees of a FTM and finalize the construction process.

2.4 Semantic Enriching of the Functional Table Model

2.4.1 Discovery of semantic labels.

In order to find semantic labels for each table region (node), we resort to the WordNet lexical ontology [8] to find an appropriate hypernym covering all tokens in the cells contained in the region. Furthermore, we also make use of the GoogleSets [3] service to find synonyms for certain tokens. For example, the first region in Table 2 consists of the tokens *adult* and *child*, for which WordNet suggests the hypernym *person*. However, the tokens are not always so 'pure', therefore we stepwise remove words in the cells by the following transformations and consult WordNet after each step to yield a suitable hypernym:

(1) punctuation removal
(2) stopword removal
(3) compute the IDF measure (where the documents are cells in our case) for each word and filter out the ones with value lower than the threshold
(4) select words that appear at the end of the cells as they are more significant [4]
(5) query GoogleSets with the remaining words in order to filter words which are not mutually similar

2.4.2 Map Functional Table Models into Frames.

In order to define how to transform a FTM into a frame, we first give a formal definition of a method and a frame:

Definition 3 (Method) *A method is a tuple* $M := (name_M, range_M, P_M)$*, where (i)* $name_M$ *is the name of the method, (ii)* $range_M$ *is a string describing the range of the method and (iii)* P_M *is a set of strings describing the parameters of the method.*

The method $Price(PersonType, RoomType) \Rightarrow NUMBER$ would for example be formally represented as the tuple (Price,NUMBER,{PersonType,RoomType}). Further, a frame is defined as follows:

[3] `http://labs.google.com/sets`
[4] The intuition here is that for nominal compounds the nominal head is at the end.

Definition 4 (Frame) *A Frame F is a pair F:=(*$name_F, M_F$*) where* $name_F$ *is the name of the frame and* M_F *is a set of methods as described above.*

Now when generating a frame, we create one method m for every region with functional role *data* with all the regions of type *access* as parameters of this method. This parameters must either be located on the same level within the same subtree or on a parent path to the root node. Here it is crucial to find appropriate names for the method ($name_M$) and parameter identifiers $p \in P_M$. The semantic label for each identifier is a combination of a region label (described in procedure above) and parent A-cell node labels. For better understanding, compare the FTM tree depicted in Figure 4 and the example of the generated frame given below. Further, we also set the range $range_M$ of the method m to the syntactic token type of the region with functional role *data* for which the method was generated. Finally, the frame for the running example, generated by the system, looks as follows:

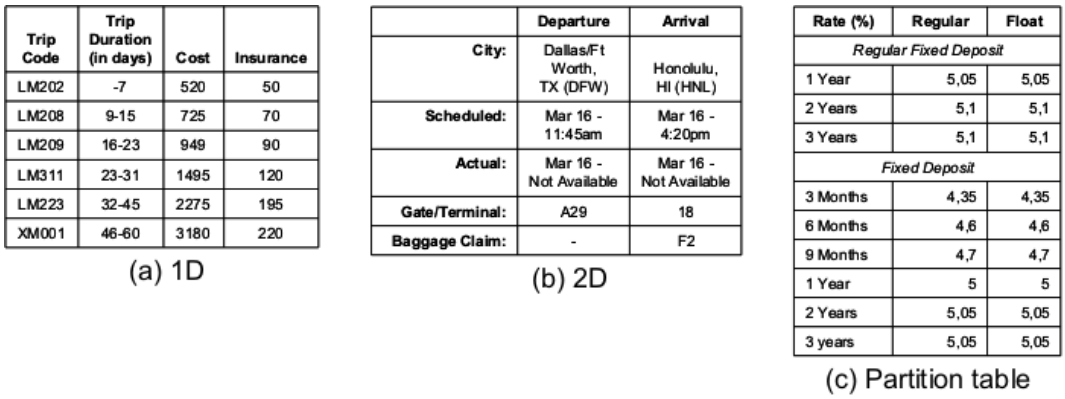
Tour [Code => ALPHANUMERIC;
DateValid => DATE;
EconomicExtension (PersonClass, RoomClass) => LARGE_NUMBER;
ExtendedExtension (PersonClass, RoomClass) => LARGE_NUMBER].

3 Evaluation

In order to evaluate our approach, we compare the automatically generated frames with frames manually created by two different subjects in terms of Precision, Recall and F-Measure. In particular, we considered 21 different tables in our experiment and asked 14 subjects to manually create a frame for three different tables such that each table in our dataset was annotated by two different subjects with the appropriate frame ($14 \times 3 = 21 \times 2 = 42$). In what follows we first describe the dataset used in the experiments. Then we describe the evaluation methodology and present the actual results of the experiment. The definition of the task as well as the instructions for the annotators can be found at `http://www.aifb.uni-karlsruhe.de/WBS/pci/FromTables2Frames.ps`

3.1 Table Classes.

We have identified three major table classes according to their layout that appear frequently on the web: *1-Dimensional* (1D), *2-Dimensional* (2D), and *Complex* tables. The first two classes are more simple and also appear more



Trip Code	Trip Duration (in days)	Cost	Insurance
LM202	-7	520	50
LM208	9-15	725	70
LM209	16-23	949	90
LM311	23-31	1495	120
LM223	32-45	2275	195
XM001	46-60	3180	220

(a) 1D

	Departure	Arrival
City:	Dallas/Ft Worth, TX (DFW)	Honolulu, HI (HNL)
Scheduled:	Mar 16 - 11:45am	Mar 16 - 4:20pm
Actual:	Mar 16 - Not Available	Mar 16 - Not Available
Gate/Terminal:	A29	18
Baggage Claim:	-	F2

(b) 2D

Rate (%)	Regular	Float
Regular Fixed Deposit		
1 Year	5,05	5,05
2 Years	5,1	5,1
3 Years	5,1	5,1
Fixed Deposit		
3 Months	4,35	4,35
6 Months	4,6	4,6
9 Months	4,7	4,7
1 Year	5	5
2 Years	5,05	5,05
3 years	5,05	5,05

(c) Partition table

Fig. 5. Examples of Tables.

often compared to the last class. A similar classification into classes has also been introduced in [24].

1-Dimensional tables: this class of tables has at least one row of A-cells above the rows of I-cells. If there are more than one row of A-cells then we assume that they are hierarchically connected. The content of the I-cells in different columns represent instances of the A-cells above. An example of this type is given in Figure 5 (a).

2-Dimensional tables: this class has a rectangular area of I-cells appearing within columns. This class has at least one row of A-cells above the rectangular area, and at least one column of A-cells on the left side of the rectangular area. Discovering and handling of this class is hard as it is difficult for a system (without any other knowledge) to decide if the first column consists of A-cells or I-cells. Our solution here is to interpret the leading column as A-cells only if its first row cell is a non-spanning cell with an empty label or a label containing a character '/'. An example for this type of table is given in Figure 5 (b).

Complex tables: this class of tables shows a great variety in layout structure. Therefore a table might have the following features:

- **Partition data labels:** Special over-spanning data labels between the data and/or attribute labels can make several partitions of the table. Each partition shares the same attributes, such as in Figure 5 (c). In this case the relation among attribute and data value cannot be obtained directly.
- **Over-expanded labels:** some entries might expand over multiple cells. There are two options: (a) data values span over multiple rows in the same column or (b) an attribute label spans over multiple columns. An example of this class is the part of Table 1 consisting in the lower seven rows.
- **Combination:** large tables might consist of several smaller, simpler ones. For example, Table 1 consists of two structurally 'independent' tables.

In our experiment, we have gathered 21 tables, each belonging to at least one class. Since the first two classes are a bit less interesting, we used only three different tables for each class, but for each complex subclass we used five different tables. All tables were gathered from two distinctive sources: one from tourist domain and another from a source dealing with food research. Domains were quite different, and also tables were selected from different sources in a way that their distribution over classes is uniform.

3.2 Evaluation Methodology.

We evaluated our approach by considering the well-known information retrieval measures Precision, Recall and F-Measure. In particular, for each table we evaluated the frame automatically generated for it by the system with respect to the two frames manually created by two different subjects along the following lines: *Syntactic Correctness, Strict Comparison, Soft Comparison, Conceptual Comparison.*

In order to assess how similar two strings are, we will introduce a string comparison operator $\sigma : String \times String \rightarrow [0..1]$. In particular, in our evaluation we use a string comparison operator based on a combination of a TFIDF weighting scheme with the Jaro-Winkler string-distance scheme. Cohen et al. [4] showed that the operator produces good results in such tasks.

The *Syntactic Correctness* measures how well the frame captures the syntactic structure of the table, i.e. to what extent the number of arguments matches the number of parameters as specified by the human annotator for a given method. In what follows we define three functions *Syntactic* giving the syntactic correctness between two methods as well as a method and a frame, respectively.

$$Syntactic_{M\times M}(m_1, m_2) := \begin{cases} \frac{|P_{m_1}|}{|P_{m_2}|} & \text{if } |P_{m_2}| > 0 \\ 1 & \text{if } |P_{m_1}| = |P_{m_2}| = 0 \\ 0 & otherwise \end{cases} \tag{8}$$

$$Syntactic_{M\times F}(m, f) = Syntactic_{M\times M}(m, m'), \tag{9}$$

where $m' \in f_M$ which maximizes $\sigma'(m, m') * Syntactic_{M\times M}(m, m')$.

Note that the above measures are directed; they will be used in one direction to obtain the Precision and in the other direction to obtain the recall of the system.

Strict Evaluation then checks if the identifier for the method name, the range and the parameters are identical. We also define a corresponding functions *Strict* again defined on two methods and a method and a frame, respectively:

$$Strict_{M\times M}(m_1, m_2) := \begin{cases} 1 \text{ if } name_{m_1} = name_{m_2} \\ 0 \qquad \text{otherwise} \end{cases} \tag{10}$$

$$Strict_{M\times F}(m, f) = max_{m'\in M_f} Strict_{M\times M}(m, m') \tag{11}$$

The Soft Evaluation also measures in how far the identifiers for the method name, the range and the parameters match, but makes use of the string comparison operator defined above:

$$Soft_{M\times M}(m_1, m_2) = \sigma(name_{m_1}, name_{m_2}) \tag{12}$$

$$Soft_{M\times F}(m, f) = max_{m'\in M_f} Soft_{M\times M}(m, m') \tag{13}$$

Further, we have a modified string comparison σ which returns 1 if the string to compare are equivalent from a conceptual point of view and σ otherwise, i.e.

$$\sigma'(s_1, s_2) := \begin{cases} 1 \qquad \text{if } s_1 \text{ a}nd \text{ } s_2 \text{ are conceptually equivalent} \\ \sigma(s_1, s_2) \qquad\qquad \text{otherwise} \end{cases} \tag{14}$$

The *Conceptual* measure was introduced to check in how far the system was able to learn the frame for a table from a conceptual point of view. In order to assess this, two of the authors compared the frames produced by the system and the ones given by the human subjects and determined which identifiers can be regarded as conceptually equivalent. In this line *RegionType*, *Region* and *Location* can be regarded as conceptual equivalent. Here are the formal definitions of the corresponding functions:

$$Conceptual_{M\times M}(m_1, m_2) = \sigma'(name_{m_1}, name_{m_2}) \tag{15}$$

$$Conceptual_{M\times F}(m, f) = max_{m'\in M_f} Conceptual_{M\times M}(m, m') \tag{16}$$

For all the above measures we compare two frames as follows:

$$X_{F\times F}(f, f') = \frac{\sum_{m\in f_M} X_{F\times F}(m, f')}{|f_M|}, \tag{17}$$

where X stands either for *Syntactic, Strict, Soft* or *Conceptual*.

In our evaluation study, we give results for Precision, Recall and F-Measure between the frame F_S produced by the system and the frames $F_1, ...F_n$ (in our case $n = 2$) produced by the human annotators. In particular, we will consider the above evaluation functions *Syntactic, Strict, Soft* and *Conceptual* in order to calculate the Precision, Recall and F-Measure of the system. Thus, in the following formulas, X stands either for *Syntactic, Strict, Soft* or *Conceptual*:

$$Prec_{Avg,X}(F_S, \{F_1, ...F_n\}) = \frac{\sum_{1 \leq i \leq n} X(F_S, F_i)}{n} \tag{18}$$

And Recall is defined inversely, i.e.

$$Rec_{Avg,X}(F_S, \{F_1, ...F_n\}) = \frac{\sum_{1 \leq i \leq n} X(F_i, F_S)}{n} \tag{19}$$

Obviously, according to the definitions of the measures, the following equations hold:

$$\begin{aligned} &Prec_{Strict} \leq Prec_{Soft} \leq Prec_{Conceptual} \quad \text{and} \\ &Rec_{Strict} \leq Rec_{Soft} \leq Rec_{Conceptual} \end{aligned} \tag{20}$$

Furthermore, we also give the value of the precision and recall for the frame which maximizes these measures, i.e.

$$Prec_{max,X}(F_S, \{F_1, ...F_n\}) = max_i \ X(F_S, F_i) \tag{21}$$

And Recall is defined inversely, i.e.

$$Rec_{max,X}(F_S, \{F_1, ...F_n\}) = max_i \ X(F_i, F_S) \tag{22}$$

Obviously, here the following equations hold:

$$Prec_X \leq Prec_{max,X} \quad \text{and} \quad Rec_X \leq Rec_{max,X} \tag{23}$$

The reason for calculating precision and recall against the frame given by both annotators which maximizes the measures is that some frames given by the annotators were not modelled correctly according to the intuitions of the authors. Thus, by this we avoid to penalize the system for an answer which is actually correct. As a byproduct of calculating $Recall_X$ and $Recall_{max,X}$ we can also indirectly judge how good the agreement between human subjects is.

Table 3
Results of the different evaluation measures (in percent).

	Average				Maximum			
	Syntactic	Strict	Soft	Conceptual	Syntactic	Strict	Soft	Conceptual
Precision	48.71	36.78	44.88	56.01	62.85	48.84	58.26	71.02
Recall	50.53	38.81	47.75	58.50	67.54	51.83	61.95	77.65
F-Measure	49.60	37.77	46.27	57.22	65.11	50.29	60.05	74.18

Finally, as is usual we balance Recall and Precision against each other by the F-Measure given by the formula:

$$F_X(P_X, R_X) = \frac{2P_X\ R_X}{P_X + R_X} \tag{24}$$

The system is now evaluated by calculating the above measures for each automatically generated frames and the corresponding frames given by the human annotator.

3.2.1 Discussion of Results.

Table 3 gives the results for the Precision, Recall and F-Measure as described above. The first interesting observation is that the values for the *maximum* evaluation are quite higher than the ones of the *average* evaluation, which clearly shows that there was a considerable disagreement between annotators and thus that the task we are considering is far from being trivial.

The results of the *Syntactic* comparison are an F-Measure of $F_{avg,Syntactic} = 49.60\%$ for the *average* evaluation and $F_{max,Syntactic} = 65.11\%$. The values show that the system is interpreting the table to a satisfactory extent from a syntactic point of view, i.e. it is determining the number of parameters correctly in most of the cases. Regarding the naming of the methods, their range and their parameters, the results vary considerable depending on the measure in question. For the *average* evaluation the results are: $F_{avg,Strict} = 37.77\%$, $F_{avg,Soft} = 46.27\%$ and $F_{avg,Conceptual} = 57.22\%$. These results show that the system has indeed problems to find the appropriate name for methods, their ranges and their parameters. However, as the conceptual evaluation shows, most of the names given by the system are from a conceptual point of view equivalent to the ones given by the human annotator. For the *maximum* evaluation we have: $F_{max,Strict} = 50.29\%$, $F_{max,Soft} = 60.05\%$ and $F_{max,Conceptual} = 74.18\%$. Thus, we can conclude that from a conceptual point of view the system is getting an appropriate name in almost 75% of the cases and it is getting the totally identical name in more than 50% of the cases. Actually, our system only failed in processing two of the 21 tables, such that in general we conclude that our results are certainly very promising.

4 Related Work

A very recent systematic overview of related work on table recognition, transformation, and inferences can be found in [32]. Several conclusions can be drawn from this survey. Firstly, only few table models have been described explicitly. Apart from the table model of Hurst which we applied in our approach [10,11] the most prominent other model is Wang's [25]. However, the model of Hurst is better suited for our purpose since it is targeted towards table recognition and transformation whereas Wang is targeted towards table generation. A table model for recognition must support two tasks: the detection of tables, and the decomposition of table regions into logical structure representation. This models tend to be more complex than generative models, since they must define and relate additional structure for recovering the components of generative models [32].

Secondly, research in table recognition, transformation, and inferences so far addressed several types of document encodings. The most work was done on plain text files, images, OCR and HTML documents [32]. Work performed on textual tables, OCR documents and images was mainly oriented towards table recognition [6,13,18,26,29], row labeling [13,17,20], and cell classification [13,17,20], where the work on web tables was extended to indexing relation detection [1,22,31] and cell/row/table merging or splitting [30]. Other approaches aim at the deep understanding of table structure, applying different techniques such as cell cohesion measures [12,27], deriving regular expressions [18], edit distance [18], graphs [13,21] as well shallow parsing of the content. Knowledge engineering techniques employing certain heuristics based on the formatting cues and machine learning techniques like decision trees [18,26], Expectation Maximization [30], Hidden Markov Models [17,26], and conditional random fields [20] have been previously explored. Table extraction methods have also been applied in the context of question answering [19,24], and ontology learning [7,23].

The work done on the task of table detection was performed by [1,13,27,28]. As evident from this work, heuristics and machine learning based approaches have been generally used to perform this task. The documents containing both real tables and tables used for layout formatting serve as an input to a table detection system. The table detection task involves separating tables that contain logical and relational information from those that are used only for layout purposes. As the output the system returns tables classified in two categories: real table and non-real table. Usually this is a pre-step in table extraction process but it could also be combined with the extraction algorithm. In contrast, we assume that tables are already harvested, and we provide a methodology and implementation which completely instantiates a table model and additionally closes the gap to formal semantics provided by ontologies.

Chen et al. [1] present work on table detection and extraction on HTML tables. The table detection algorithm uses string, named entity and number category similarity to decide if it is a real or non-real table. Based on cell similarity the table extraction algorithm identifies whether the table is to be read row wise or column wise. They split the cells which span over multiple cells into individual cells. The table extraction algorithm presented in this work is simple and works only if spanning cells are used for nested labels. The paper did not provide evaluation results for their table extraction algorithm.

The problem of merging different tables which are about the same type of information has been addressed in [30]. The merging task as defined in this paper considers combining different tables into one large table. They define different structures for tables based on the arrangement of the labels and use Expectation Maximization to classify the tables to one of the defined structures. The structure recognition task is similar to the classification task. However structure recognition or merging does not solve the table extraction problem.

Table extraction by wrapper learning has been explored in [3]. Wrappers learn rules based on examples. The rules are composed of tokens made of HTML tags or the content itself. The rules tend to be specific and can be applied only to those documents whose structure is similar to the training documents. The use of tokens to compose rules makes it difficult to generalize across distributed websites. Hence wrappers learned for one website cannot be used on another website. No clear evaluation for table extraction has been described in this work.

Conditional random fields for table extraction from text tables were described in [20]. However the system described does not perform a complete table extraction task; it only classifies the rows of the table into a type such as 'datarow', 'sectionheader' or 'superheader'. They used a set of 12 table types (classes) and achieve a precision of 93.5% for the classification task. Work presented in [13,17] also focused on the task of classifying table rows.

Tijerino et al. [23] presented a vision for a system that would be able to generate ontologies from arbitrary tables or table-equivalents. Their approach consists of a four step methodology which includes table recognition and decomposition, construction of mini ontologies, discovery of inter-ontology mappings, and merging of mini-ontologies. For the purpose of semantics discovery the approach is multifaceted, meaning they use all evidence at their disposal (i.e. Wordnet, data frames, named entities, etc.). Since the paper presents only a vision, no evaluation is provided.

We conclude thus in this section that our approach is indeed novel in the sense that it is the first approach addressing the whole process of transforming a

table into a form reflecting its inherent meaning at a structural, functional and semantic level. Further, as far as we know our method is also original in being the first complete instantiation of a formal table model such as the one described by [10,11].

5 Conclusion

We have presented an approach which stepwise instantiates a formal table model consisting of *Physical*, *Structural*, *Functional* and *Semantic* components. The core steps of the methodology are (i) Cleaning and Normalization, (ii) Structure Detection, (iii) Building of the Functional Table Model (FTM) and (iv) Semantic Enriching of the FTM. We have further demonstrated and evaluated the successful automatic generation of frames from HTML tables. Additionally, our experimental results show that from a conceptual point of view the system is getting appropriate names for frames in almost 75% of the cases and it is getting the totally identical name in more than 50% of the cases. These results are certainly very promising.

In general, an approach such as presented in this paper is crucial for the Semantic Web to become feasible. Actually it is a generally accepted fact that without the help of tools reliably harvesting information from databases, images, tables or texts the Semantic Web as an extension of the current World Wide Web is doomed to failure. Though our work can still be improved in many ways, we regard it as a first and important step towards extracting knowledge from table structures available in the web which brings us a a little bit closer to the vision of a semantic one.

Acknowledgments This work has been supported by the IST-projects Dot.Kom (Designing adaptive infOrmation exTraction from text for KnOwledge Management, IST-2001-34038) and SEKT (Semantically Enabled Knowledge Technologies, IST-2004-506826), sponsored by the EC as part of the frameworks V and VI, respectively. During his stay at the AIFB, Aleksander Pivk has been supported by a Marie Curie Fellowship of the European Community program 'Host Training Sites' and by the Slovenian Ministry of Education, Science and Sport. Thanks to all our colleagues for participating in the evaluation of the system as well as to the reviewers for useful comments on the paper.

References

[1] H. Chen, S. Tsai, and J. Tsai. Mining tables from large scale HTML texts. In *Proceedings of the 18th International Conference on Computational Linguistics*

(COLING), pages 166–172, 2000.

[2] E.A. Codd. A relational model for large shared databanks. *Communications of the ACM*, 13(6):377–387, 1970.

[3] W.W. Cohen, M. Hurst, and L.S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *Proceedings of the 11th World Wide Web Conference*, pages 232–241, Honolulu, Hawaii, May 2002.

[4] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IIWeb Workshop at the IJCAI 2003 conference*, 2003.

[5] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al., editors, *Database Semantics: Semantic Issues in Multimedia Systems*, pages 351–369. Kluwer, 1999.

[6] S. Douglas and M. Hurst. Layout and language: List and tables in technical documents. In *Proceedings of ACL SIGPARSE Workshop on Punctuation in Computational Linguistics*, pages 19–24, 1996.

[7] D.W. Embley, C. Tao, and S.W. Liddle. Automatically extracting ontologically specified data from html tables with unknown structure. In *Proceedings of the 21th International Conference on Conceptual Modeling*, pages 322–337, Tampere, Finland, October 2002.

[8] C. Fellbaum. *WordNet, an electronic lexical database*. MIT Press, 1998.

[9] S. Handschuh and S. Staab, editors. *Annotation in the Semantic Web*. IOS Press, 2003.

[10] M. Hurst. Layout and language: Beyond simple text for information interaction - modelling the table. In *Proceedings of the 2nd International Conference on Multimodal Interfaces, Hong Kong*, 1999.

[11] M. Hurst. *The Interpretation of Tables in Texts*. PhD thesis, University of Edinburgh, 2000.

[12] M. Hurst. Layout and language: Challenges for table understanding on the web. In *Proceedings of the International Workshop on Web Document Analysis*, pages 27–30, 2001.

[13] J. Hu, R.S. Kashi, D. Lopresti, and G.T. Wilfong. Evaluating the performance of table processing algorithms. *International Journal on Document Analysis and Recognition*, 4(3):140–153, March 2002.

[14] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1995.

[15] K. Lerman, S. Minton, and C. Knoblock. Wrapper maintenance: A machine learning approach. *J. of Artificial Intelligence Research*, 18:149–181, 2003.

[16] A. Maedche. *Ontology Learning for the Semantic Web.* Kluwer Academic Publishers, 2002.

[17] A. McCallum, D. Freitag, and F. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the ICML 2000*, pages 591–598, 2000.

[18] H. T. Ng, C. Y. Kim, and J. L. T. Koo. Learning to recognize tables in free text. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 443–450, Maryland, USA, 1999.

[19] D. Pinto, W. Croft, M. Branstein, R. Coleman, M. King, W. Li, and X. Wei. Quasm: A system for question answering using semi-structured data. In *Proceedings of the Joint Conference on Digital Libraries (JCDL) 2002*, pages 46–55, 2002.

[20] D. Pinto, A. McCallum, X. Wei, and W.B. Croft. Table extraction using conditional random fields. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, pages 235–242. ACM Press, 2003.

[21] P. Pyreddy and W.B. Croft. Tintin: A system for retrieval in text tables. In *Proceedings of the Second ACM International Conference on Digital Libraries*, pages 193–200. ACM Press, 1997.

[22] A. Tengli, Y. Yang, and N. Li Ma. Learning table extraction from examples. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages –, Geneva, Switzerland, May 23–27 2004.

[23] Y.A. Tijerino, D.W. Embley, D.W. Lonsdale, and G. Nagy. Ontology generation from tables. In *Proceedings of 4th International Conference on Web Information Systems Engineering (WISE'03)*, pages 242–249, Rome, Italy, December 2003.

[24] H. L. Wang, S. H. Wu, I. C. Wang, C. L. Sung, W. L. Hsu, and W. K. Shih. Semantic Search on Internet Tabular Information Extraction for Answering Queries. In *Proceedings of the 9th International Conference on Information and Knowledge Management*, pages 243–249, Washington DC, 2000.

[25] X. Wang. *Tabular Abstraction, Editing and Formatting.* PhD thesis, U. of Waterloo, 1996.

[26] Y. Wang, R. Haralick, and I. Phillips. Zone content classification and its performance evaluation. In *Proceedings of the 6th International Conference on Document Analysis and Recognition (ICDAR01)*, pages 540–544, Seattle, Washington, September 2001.

[27] Y. Wang and J. Hu. Detecting tables in HTML documents. In *Document Analysis Systems*, pages 249–260. Springer-Verlag, 2002.

[28] Y. Wang and J. Hu. A machine learning based approach for table detection on the web. In *Proceedings of the 11th International Conference on World Wide Web*, pages 242–250. ACM Press, 2002.

[29] Y. Wang, I.T. Phillips, R.M.Robert, and M. Haralick. Table structure understanding and its performance evaluation. *Pattern Recognition*, 37(7):1479–1497, July 2004.

[30] M. Yoshida, K. Torisawa, and J. Tsujii. A method to integrate tables of the world wide web. In *Proceedings of the International Workshop on Web Document Analysis (WDA 2001)*, pages 31–34, 2001.

[31] M. Yoshida, K. Torisawa, and J. Tsujii. Extracting attributes and their values from web pages. In A. Antonacopoulos and J. Hu, editors, *Web Document Analysis: Challenges and Opportunities*, Series in Machine Perception and Artificial Intelligence, pages 179–200. World Scientific, 2003.

[32] R. Zanibbi, D. Blostein, and J.R. Cordy. A survey of table recognition: Models, observations, transformations, and inferences. *International Journal of Document Analysis and Recognition*, 7(1):1–16, March 2004.