From Tables to Frames

Aleksander Pivk[1,2], Philipp Cimiano[2], and York Sure[2]
[1]J. Stefan Institute, Dep. of Intelligent Systems, Ljubljana, Slovenia
http://ai.ijs.si/
aleksander.pivk@ijs.si
[2]Institute AIFB, University of Karlsruhe, Karlsruhe, Germany
http://www.aifb.uni-karlsruhe.de/WBS/
{pivk,cimiano,sure}@aifb.uni-karlsruhe.de

July 19, 2004

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 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.

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 [HS03] will not scale in general. More scalable (semi-)automatic approaches known from ontology learning (cf. [Mae02]) 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 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. 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.

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 and semantics. There also exist models to describe tables in similar ways (cf. [Hur00, Hur99]) 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** – 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 ([Hur00]) distinguishes between two functional cell types *access* and *data*. Cells of the 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 a 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>
<tr><td colspan="3">Tour Code</td><td colspan="2">DP9LAX01AB</td></tr>
<tr><td colspan="3">Valid</td><td colspan="2">01.05. - 30.09.04</td></tr>
<tr><td colspan="3">Class/Extension</td><td>Economic</td><td>Extended</td></tr>
<tr><td rowspan="3">Adult</td><td rowspan="6">P
R
I
C
E</td><td>Single Room</td><td>35,450</td><td>2,510</td></tr>
<tr><td>Double Room</td><td>32,500</td><td>1,430</td></tr>
<tr><td>Extra Bed</td><td>30,550</td><td>720</td></tr>
<tr><td rowspan="3">Child</td><td>Occupation</td><td>25,800</td><td>1,430</td></tr>
<tr><td>No occupation</td><td>23,850</td><td>720</td></tr>
<tr><td>Extra Bed</td><td>22,900</td><td>360</td></tr>
</table>

Table 1: Example of a possible table, found in [CTT00]

Regarding the *semantic* description we follow a completely different paradigm as Hurst. Instead of adopting the relational model ([Cod70]), we describe the semantics of a table in terms of F-Logic frames ([KLW95]). F-Logic combines the intuitiveness of modeling with frames with the expressive power of logic. Furthermore, existing F-Logic inference engines such as Ontobroker ([DEFS99]) 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 at 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.

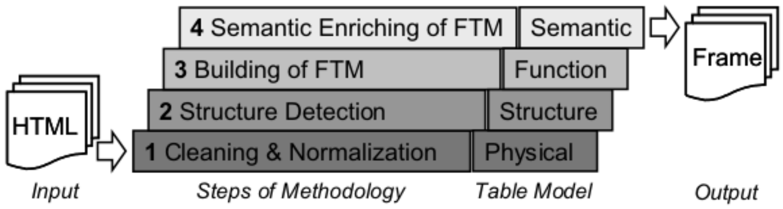

Figure 1: Building blocks of the methodology

2.1 Cleaning and Normalization

First, we assume that documents are represented with the *Document Object Model* (DOM)[1]. 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

[1]http://www.w3.org{/DOM/,/TR/html4/,/People/Raggett/tidy/}

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

Table 2: Table 1 after cleaning and normalization step

contains a string-valued tag (such as `table`, `h1` or `title`) 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 Cleaning and Normalization step we want to construct an initial table model out of a DOM tree. This model cannot be simply generated by applying the algorithm recommended by W3C[1] on a `table` element, but some additional steps of processing and refinement are needed.

HTML documents are often very noisy in a sense that their syntactic structure is incorrect. In order to clean the code and make it syntactically correct, we employ the Tidy[1] utility . The outcome is a cleaned and corrected DOM tree.

The normalization of the rendered table is necessary, when an explicit `rowspan` or `colspan` attribute indicates multiple row or column spanning cells and the actual total number of rows or columns is lower than the attribute value. In this step 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 matrix structure.

2.2 Structure Detection

2.2.1 Assignment of functional types and probabilities to cells.

In the first walk over the `table` element (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* (see Figure 2). At the same time, we assign each cell in the rendered table a *functional type* (A-cell or I-cell) and a probability for this type. By default, a cell is assigned no functional type, which is observed by a probability having value zero, unless the cell includes only/mostly tokens, recognized as dates, currencies, or numerical values. In the latter case the cell is assigned the type `I-cell`, and its probability is calculated based on the proportion of tokens which talk in favour of this type. We also assume that the cell in the lowest right corner is always an I-cell, and the cell in the upper-left corner is an A-cell. Therefore we assign them the type, 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. 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 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 edges between them. This representation is flexible and can be extended to include domain specific information. For example, the numeric type is divided into categories that include range information about the number, i.e. LARGE_NUM ($\geq$ 10.000), MED_NUM ($100 \leq n < 10.000$), SMALL_NUM (< 100), and CURRENCY, which can be treated as a domain specific information.

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 are aligned at the head) and for the right-side tokens (tokens are aligned at the tail). The distance is in the latter case also averaged.

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

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.

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

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

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:

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

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

Here we only compare equal number of rows and columns thus optimizing the comparison. 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*.

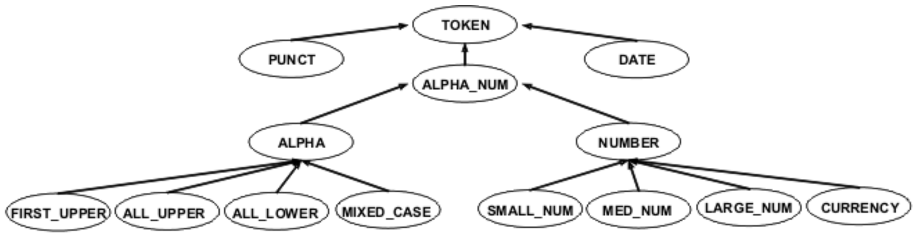

Figure 2: Hierarchy of token types

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 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 Table 3.

1. *Initialize logical units and regions.* First, the system splits a table into *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. Note that a table itself is by definition one logical unit.

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

Table 3: Algorithm for discovery of regions

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, 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) it 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 pre-defined value:

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

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 patterns are of two types: first represents the content of cells from left-to-right (forward) and second from right-to-left (backward). The pattern 'FIRST_UPPER Room' for example covers the cells 'Single Room' and 'Double Room'. For the purpose of pattern construction we have implemented DATAPROG algorithm, which is together with detailed pattern learning process described in [LMK03].

   Before entering a loop, the system checks the uniformity of every logical unit. In our case, the 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.* If a logical unit is not uniform, the system chooses its *best region*, which is used to normalize neighboring regions and consequently the logical unit itself. The best region r_{max} is chosen according to the formula $\Phi_{r_{max}} = max_{r \text{ in } l}\, \phi_{r,l}$, which is calculated by the following equation:

$$\phi_{r,l} := \left(\frac{|r|}{|l|} + \frac{1}{|r|} \sum_{c \text{ in } r} P(c) + \frac{1}{2*|r|*|P_r|} \sum_{p \in P_r} covers(p,r) \right) \quad (7)$$

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 loop. To finalize, in this way all logical units are being normalized and prepared for further processing.

2.3 Building of 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 leafs 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 3) 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 3.

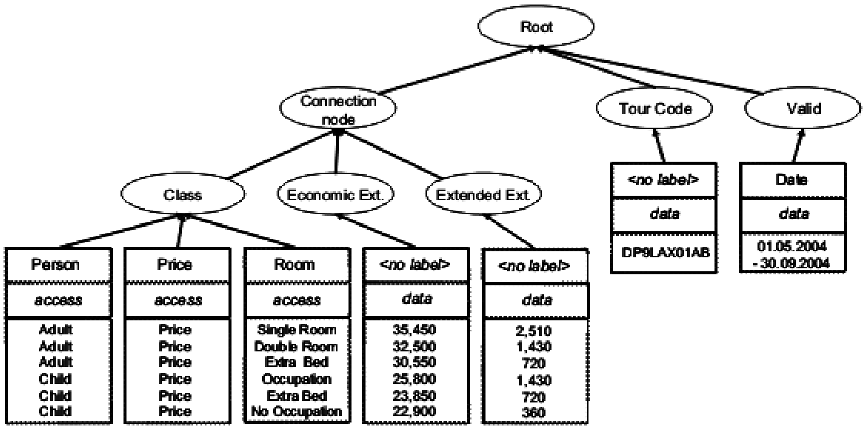

Figure 3: A functional table model (FTM) of the running example (Table 1) with square components representing I-cells and rounded components representing A-cells.

After 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. An example of recapitulation is depicted in Figure 4. 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.

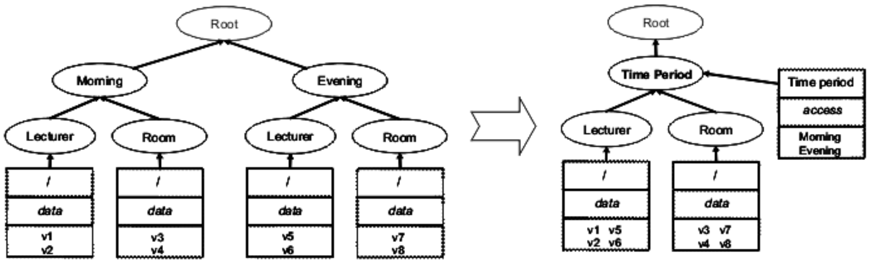

Figure 4: An example of recapitulation process.

2.4 Semantic Enriching of 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 [Fel98] to find an appropriate hypernym covering all tokens in the cells contained in the region. Furthermore, we also make use of the GoogleSets[2] service to find synonyms for certain tokens. For example, a region in Table 2 consists of 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 TFIDF measure (where the documents are cells in our case) for each word and filter those for which the value is lower than 1/3
4. select words that appear at the end of the cells as they are more significant[3]
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:

[2] http://labs.google.com/sets

[3] The intuition here is that for nominal compounds the nominal head is at the end.

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:

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 3 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`

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

Figure 5: Examples of Tables

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 often compared to the last class. A similar classification into classes has also been introduced in [WWW$^+$00].

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 incorporate a great deal of 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. Example of this class is part of the Table 1, in particular 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 ([CRF03]). 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.

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

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

where m' is that method 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:

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

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

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:

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

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

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.

$$\sigma'(s_1, s_2) := \begin{cases} 1 & \text{if } s_1 \ and \ s_2 \text{ are conceptually equivalent} \\ \sigma(s_1, s_2) & \text{otherwise} \end{cases}$$

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:

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

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

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

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

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*:

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

And Recall is defined inversely, i.e.

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

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

$$Prec_{Strict} \le Prec_{Soft} \le Prec_{Conceptual}, \ Rec_{Strict} \le Rec_{Soft} \le Rec_{Conceptual}$$

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

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

And Recall is defined inversely, i.e.

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

Obviously, here the following equations hold: $Prec_X \leq Prec_{max,X}$, $Rec_X \leq_{max,X}$. 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.
Finally, as is usual we balance Recall and Precision against each other by the F-Measure given by the formula: $F_X(P_X, R_X) = \frac{2P_X\ R_X}{P_X+R_X}$
The system is now evaluated by calculating the above measures for each automatically generated frames and the corresponding frames given by the human annotator.

3.3 Discussion of Results

Table 4 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.

	Average				Maximum			
	Synt.	Strict	Soft	Conc.	Synt.	Strict	Soft	Conc.
Prec.	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-Meas.	49.60%	37.77%	46.27%	57.22%	65.11%	50.29%	60.05%	74.18%

Table 4: Results of the different evaluation measures

4 Related Work

A very recent systematic overview of related work on table recognition can be found in [ZBCar]. 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 have used as a baseline [Hur99, Hur00] the most prominent other model is from Wang [Wan96]. However, the model of Hurst is better suited for our purpose since it is targeted towards table *recognition* whereas Wang is targeted towards table *generation*. Secondly, it becomes clear that research so far in table recognition focused on recovering tables from encoded documents. In contrast, we assume that tables are already harvested. Furthermore, we provide a methodology and implementation which completely instantiates a table model and additionally closes the gap to formal semantics provided by ontologies. Our approach allows subsequently for using the full potential ontologies offer e.g. for query answering or knowledge retrieval over a set of tables.

5 Conclusion

We have shown how our methodology stepwise instantiates the underlying table model which consists 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 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.
Future research is aiming into two main directions. Firstly, generating one frame per table might not always be the preferred solution. E.g. tables might have different structural components, but still represent the same data. Or, they might even represent different data, but one most common frame covering all table data instead of a number of different frames seems desirable to reduce complexity. Thus, our aim is to generalize the approach to be able to generate out of multiple tables one (most general) frame.
Secondly, after having generated frames an intuitive next step is to use these frames to automatically populate ontologies with instances. The core idea is that the approach allows for automatic population of ontologies from arbitrary table-like structures. We expect that our approach potentially will be used to semantically enrich a variety of legacy data and will support the conversion of the existing Web into a Semantic Web.

References

[Cod70] E.A. Codd. A relational model for large shared databanks. *Communications of the ACM*, 13(6):377–387, 1970.

[CRF03] 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.

[CTT00] H. Chen, S. Tsai, and J. Tsai. Mining tables from large scale HTML texts. In *Proc. of the 18th Int. Conf. on Computational Linguistics (COLING)*, pages 166–172, 2000.

[DEFS99] 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.

[Fel98] C. Fellbaum. *WordNet, an electronic lexical database*. MIT Press, 1998.

[HS03] S. Handschuh and S. Staab, editors. *Annotation in the Semantic Web*. IOS Press, 2003.

[Hur99] M. Hurst. Layout and language: Beyond simple text for information interaction - modelling the table. In *Proc. of the 2nd Int. Conf. on Multimodal Interfaces, Hong Kong*, 1999.

[Hur00] M. Hurst. *The Interpretation of Tables in Texts*. PhD thesis, University of Edinburgh, 2000.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1995.

[LMK03] K. Lerman, S. Minton, and C. Knoblock. Wrapper maintenance: A machine learning approach. *J. of Artificial Intelligence Research*, 18:149–181, 2003.

[Mae02] A. Maedche. *Ontology Learning for the Semantic Web*. Kluwer Academic Publishers, 2002.

[Wan96] X. Wang. *Tabular Abstraction, Editing and Formatting*. PhD thesis, U. of Waterloo, 1996.

[WWW$^+$00] 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 *Proc. of the Ninth Int. Conf. on Information and Knowledge Management, Washington DC*, pages 243–249, 2000.

[ZBCar] 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*, to appear.