Extracting Structured Data from Unstructured Document with Incomplete Resources

Hervé Déjean
Xerox Research Centre Europe
Meylan, France

***Abstract*— We present a method for extracting structured elements of information, called *structured data (sdata)*, from ocr'ed pages. The method first analyzes the layout of the page, building several concurrent layout structures. Then a tagging step is performed in order to tag textual elements based on their content. Combining the layout structures and the tagged elements, layout models for representing the structured data are inferred for the current page. These models are used to correct or tag some elements missed by the tagging step. The final set of structured data is extracted. An evaluation is presented.**

Keywords—Document Layout Analysis, data extraction

I. INTRODUCTION

According to [1], Document Understanding *refers to the field that is concerned with logical and semantic analysis of documents to extract human understandable information and codify it into machine-readable form.*" A common application is data extraction from an unstructured document (often image). We would like to showcase in this paper extraction of **structured data**: data consisting of several **fields** (e.g. an invoice item can be considered as 3 fields: description, price per unit, unit amount). We will refer to such data as **sdata** hereafter (sdatum: a union of several fields).

The main issue in extracting structured data is the lack of correspondence between the data/data fields and the way their layout is realized (except of course when the documents mostly follow a layout template such as forms). In some documents, one single block can contain all the data fields. In another document and for the same data type, each field may be spread over different table cells (see Figs 1-4). The main issue is then to find a **mapping** between layout elements (line, block, and cell) and each data element (fields). Often this mapping varies from document to document for the same collection and the same data type. We present here a method in order to compute this mapping, and hence to extract structured data from ocr'd documents.

Literature is huge regarding Information/Data Extraction. We refer once again to [1] for an overview. Beyond the Document Understanding paradigm, most of the work in data extraction considers text as entry format (involving Natural Language Processing) and do not consider much the layout analysis step (the main exceptions being invoice processing and of course web pages, but techniques used for the latter are very different). We refer here to [2] for a very relevant bibliographical list.

Table understanding [3] is another relevant paradigm since tabular structure is often used for representing structured data. Our layout analysis step includes table analysis, but it goes beyond since other layout structures have also to be covered. Related to Data Extraction and Table Understanding, [4] showcases a fairly comprehensive study (stating problems and some solutions) for financial documents.

This work belongs to the unsupervised Machine Learning paradigm: no annotated data is provided. However we consider that the structure of the data to be extracted is known: the ultimate step being to store them in a database, this knowledge is required, and drives the extraction process (during the inference step).

In the remainder of the article, we first present a specific use-case used to illustrate the method, then the method itself, and finally evaluations, one being conducted by the data owner.

II. USE-CASE

The data we use to illustrate the method correspond to architectural plans. Beside architectural description, they also contain some information about the companies (called *factors*) involved in the construction such as owner, architect, and engineers. One of our customers is interested in automatically extracting such information from scanned plan sheets. Roughly 60,000 plans are processed weekly. Factors mainly occur on the cover sheets of the plans sets (approx. 1-5%). A collection of 1000 construction cover sheets was processed with the described method. One can have an insight of this type of documents by searching on the web images with this query 'architectural plan cover sheets'.

Plan sheets were first OCR'd, then basic blocks were generated (a block corresponds to a set of lines with similar font size and inter-line space). The sdatum we want to extract is composed of 3 main parts: a discipline (mandatory), a name (mandatory) and coordinates: street, phone, … (optional). Here is an example:

```
<FACTOR>
<FIELD name="FOREFRONT STRUCTURAL ENGINEERS, INC."/>
<FIELD discipline="STRUCTURAL ENGINEER"/>
<FIELD street="228 SOUTH WABASH AVENUE"/>
<FIELD zip="60604"/>
<FIELD state="ILLINOIS"/>
<FIELD phone="312.376.1140"/>
</FACTOR>
```

On the textual part, one difficulty is the presence of terms used by factors in many parts of the plan: Disciplines names (architect, mechanical, electrical, plumbing) are used to organize the construction project, and these terms occur in many parts of the plan (indexes). Furthermore, there is some content similarity between factor fields such as discipline and name (for instance *architect* is mentioned in the discipline field and in the name field). Finally some other types of sdata are very close to the ones we want to extract (other data with coordinates).

On the layout part, plans are very challenging: large sheets (24'x66') with sometimes very sparse content, sometimes very dense. Being able to conduct a robust layout analysis is very challenging. Furthermore, the possible range of font sizes (from 6 points to 40 points and more) makes OCR engines' life very difficult! Figs. 1-4 show some examples where the structured data fields are organized in various layouts. Lines belonging to homogenous typographical (as computed by ourselves) zones are merged in one block (black rectangle), let alone otherwise. The data may be more or less rich (with full coordinates) or not. The various layout structures go from the simple list of blocks containing the full factor (vertical or horizontal) to a table where each cell contains one single field (factors then correspond to rows or columns).

One challenge is first to identify correctly the layout structures, and then to find the mapping between the layout structure elements and the data elements (fields).

The test-set and a sub-test of development set are available on https://open.xerox.com/Data (50 annotated sheets; more than 100 non-annotated ones; account needed). We describe now the method, and will describe the different conducted evaluations in Section IV.

III. Our Method

The task consists in extracting structured data and in identifying each field of a given sdatum. As for most all data extraction methods, **lexical resources** are required (list of terms, regular expressions), but they do not need to be comprehensive as we will see.

We propose the following method:

1. **Layout analysis** for detecting regular layout structures. **Competing** structures can be generated (one element can belong to several layout structure)

2. **Textual tagging** for detecting some data fields.

3. **Data model generation**. This is the main step of the method. It infers a model (so-called **c-layout** model) which combines content information and layout information. Using this inferred model, it allows for completing some elements missed by the tagging step.

4. **Data extraction**: based on the previously inferred model the final data extraction is performed. Sdata not covered by the c-layout model are also extracted using basic heuristics.

We will now describe each step of the method.

A. Layout Structures Generation

The goal of this step is the generation of layout structures in the page. These structures can correspond to 2-dimensional structures (tabular structure) or 1-dimensional structures (vertical or horizontal elements). First, blocks are generated. Then, elements composing a Manhattan layout are detected similarly to [5]. This allows us to structure elements which have a tabular structure. The output of this step corresponds to a table structured in rows and cells (à la HTML). Finally, sequences of vertical and horizontal elements are also collected (using block alignments). Plan layout can be very challenging, and our solution is simply to generate concurrent analyses: algorithms are tuned so that the emphasis is put on detecting complete structures (even if extra elements are included).

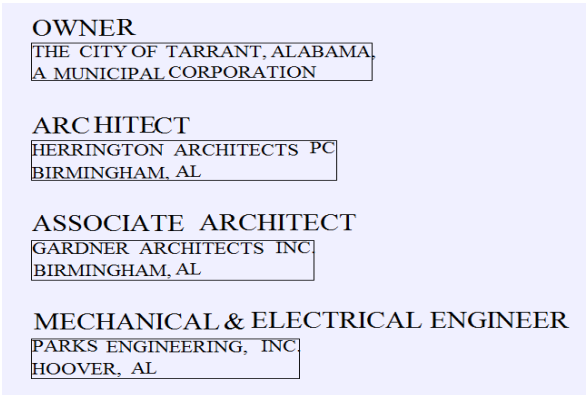

Fig. 1. A vertical layout: disciplines are using bigger font sizes and are in bold. The coordinates correspond to the city name and state abbreviation

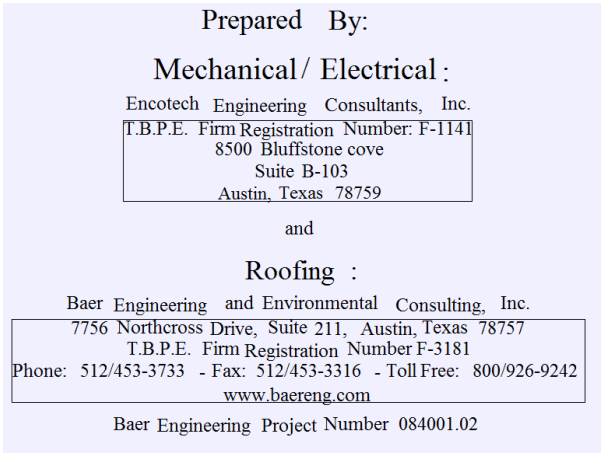

Fig. 2. Centered vertical layout. Different font sizes are used for discipline, names, and coordinates

At this stage an element (line or block) can belong to **several** layout structures. The selection of the 'right' structure is done (indirectly) through the data extraction step: by selecting the structures which provided complete sdata. Remember that the purpose of the method is data extraction, and layout analysis is a means to perform this extraction.

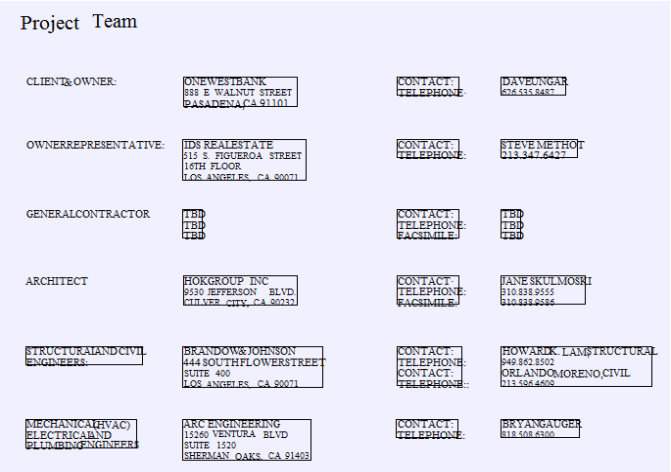

Fig. 3. A tabular layout is here used (a factor corresponds to a row)

B. Content Tagging

We now describe how content is tagged. An sdatum is composed of fields. In order to identify some field candidates, tagging is performed using basic regular expressions or closed lists in order to tag the most frequent elements of a field (most frequent content or pattern). Let us call the tagged elements an entity. Basic entities covered by regular expressions are: phone number, address, URL, date, etc. The entities described by list of terms depend on the application. In the current implementation the considered content element for this tagging corresponds to the line. A part or all the content of a line can be tagged.

Nothing new here, except that the resources used here must be designed so that recall is privileged (precision is less important). We do not want to miss sdata, and the method is able to filter out false positives thanks to the next step: by combining content and layout. This completion step, based on layout structure, is performed in Step 3 (see next section). An example of resources for our use case is given in Section V.

C. Data Model Generation: The C-layout Model

The Data Model Generation Step aims at inferring a twofold-model, combining layout and content structures. Given an sdatum, many layout variations can be used to 'display' it. The goal of this step is then to find out, for a specific page, which layout structures are used, and how the content elements are organized in this layout (by finding out the reading order, especially for tabular structures).

1) Grammar Generation

At this stage, we have a set of layout structures (an element being able to belong to several structures). Layout structures having tagged elements are selected and used in order to generate a data model: the c-layout model (for Content-Layout model). This model associates a layout structure (1D or 2D) and a content structure and describes a specific instance of the sdatum for the current processed document (how the sdatum is laid out in this document). Let us take the elements shown Fig. 5. Layout Structures (rectangle) are shown as well as tagged elements (underlined). The dotted lines represent the tabular structure identified (2 rows, 4 columns).

As input of this inference step, we need to have a sequence of ordered elements. We systematically transform one 2D-structure into **2** 1D-sequences: as a set of rows and as a set of columns (the element in such a sequence is the cell). For 1D structures, the sequence of elements is provided by the 1D-structure itself. In our example, we get two sequences (hereafter seq1 and seq2): seq1 = [1;2;3;4;5;6;7;8] and seq2 = [1;5;2;6;3;7;4;8] (Numbers refer to cell numbers Fig. 5).

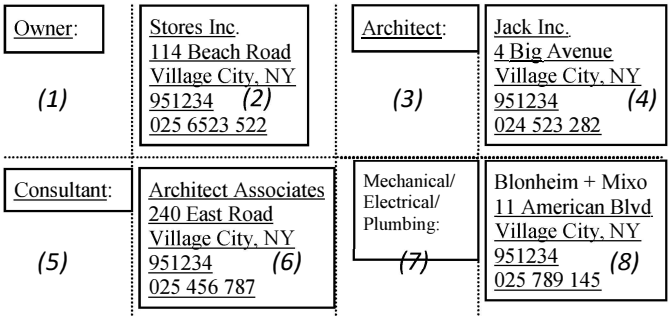

Fig. 4: our running example: a table composed of 8 cells

We will now explain how a grammar which describes the sdata is inferred from a sequence of elements. This is the key step of the method. It is in line with known algorithms in the Grammar Inference domain. One very close work in spirit is [2], which infers structures 'unsupervisedly'. Our main difference with most previous work is the way we characterize elements: using **multiple** features, while in general, an element of the structure has one unique value (surface string or part-of-speech for instance).

The steps of the inference process are as follows (input is an ordered sequence of elements):

1. Elements characterization: Elements are described with a set of multiple **features**. Features are generated from the tagging step: for each tag associated to an element, a feature of Boolean type is generated. Table I shows the list of features associated to each element. Some textual elements are wrongly tagged (cell 6) or not tagged (cell 7).

TABLE I. EACH ELEMENT IS ASSOCIATED A SET OF FEATURES. SEVERAL FEATURES ARE POSSIBLE. SOME ELEMENTS CAN HAVE NO FEATURES.

Element (cell)	Features (Boolean; feature value *True* is skipped)
1	DISCIPLINE
2	NAME,STREET, STATE, ZIPCODE,TEL
3	DISCIPLINE
4	STREET, STATE, ZIPCODE,TEL
5	DISCIPLINE
6	*DISCIPLINE*,NAME,STREET, STATE, ZIPCODE,TEL
7	*[]*
8	STREET, STATE, ZIPCODE,TEL

2. N-grams generation: a set of n-grams (from 1 to N, N being a parameter) is generated for the sequence of elements using elements' features. For instance, seq1 can generate these bigrams (cell number in brackets): (DISCIPLINE[1],

STREET[2]), (DISCIPLINE[1], NAME[2]), … (NAME[2], DISCIPLINE[3])

3. **Repetitive** N-grams are selected and ordered by frequency. For each n-gram a test is performed to assess its **repetitiveness**: considering each sub-sequence covered by a given n-gram, we compute how frequent this sequence is immediately followed by a sub-sequence covered by the **same** n-gram. If this frequency is higher than a given threshold (implemented: 0.5* frequency (n-gram)), then the n-gram is considered as sequential. In our example, the n-gram (DISCIPLINE, STREET) is repetitive (among others).

4. The most frequent n-gram is selected and sub-sequences of elements matching this n-gram are structured (regrouped under a same node). Step 1 is applied iteratively on this sequence, generating a tree structure.

Applying this algorithm as just described generates the following grammars for seq1 and seq2 (+: Kleene plus):

Seq1: (DISCIPLINE, STREET)+ (GR1)

Seq2: (DISCPLINE+, STREET+)+ (GR2)

Both sequences (Seq1 and Seq2) generate two different grammars GR1 and GR2. While both selected the same features (DISCIPLINE, STREET), the grammar structures are different. How to choose the right one for the c-layout model? For this, we use the notion of **sdatum completion**. Back to the example shown Section II, our **factor** structure is composed of mandatory and optional fields. An sdatum is said complete when all mandatory fields are found. In our example, the mandatory fields are discipline and name. From the set of inferred grammars we only keep inferred **grammars in which one level corresponds to a single and complete sdata**. This completion is checked at the data level: elements are parsed using a grammar; and the grammar nodes of the resulting tree structure are checked for completeness: only one tree level can be considered as complete.

In our example, the one covering a complete sdatum is selected (GR1), and GR2 is discarded. This model shows that a 2-cell structure is used to represent a complete sdatum.

2) C-layout Model Generation

We have identified the layout used to represent an sdatum, and find how to read (order) the pieces of information present in the sdatum. But we need a more accurate representation of the sdatum: The purpose of the ***c-layout model*** is to associate a **content tag to a specific layout position using layout sub-elements** (here at the LINE level, the structural cell sub-level). In our example, an sdatum is composed of two contiguous aligned cells (Fig. 4). Then each line in a cell is associated to a content entity. For this, we simply build an histogram of the tags occurring in each line, and keep all tags which are more frequent than a given threshold. Table II shows the resulting structure with the frequency in brackets.

Table II: The final c-layout model (frequency in brackets)

DISCIPLINE [3]	NAME [3] STREET [3] STATE , ZIPCODE [3] TEL[3]

3) Data Correction

Once the c-layout model is inferred, we use it in order to correct some missed elements in the structure (elements not (correctly) tagged). We parse again the elements using the c-layout model. Missing tags are added to lines which were not tagged. In our example, the c-layout model allows for correcting two errors: the first one is in cell 7 (discipline), the second in cell 8 (name).

Table III: Tagging Correction applying the c-layout model.

DISCIPLINE: Mechanical/ Electrical/ Plumbing :	**NAME**: Blonheim + Mixo STREET: 11 American Blvd STATE , ZIPCODE: Town City, NY 95990 TEL: 025 789 145

D. Final Extraction

At this stage, we are able to identify sdata present in a layout structure. Since elements can belong to several layout structures (typically, a 2D structure and a 1D structure), or can appear several times in the document, duplicated sdata are merged (comparison is done using mandatory fields and we keep the one having the most identified fields). The sdata extracted by this method are highly reliable: they rely on regularity at layout level and at content level.

IV. Evaluation

We first report on a quantitative evaluation on 1000 plans, and then on a smaller dataset of 50 plans, whose evaluation has been performed by the data owner himself.

A. Quantitative Evaluation

We compare different ways of extracting data. Using a large number of non-annotated plans (1000), we compute the following features:

- **Baseline**: This baseline shows the number of sdata extracted by simply considering typographical blocks. No further layout analysis is performed
- **Structural**: extraction using the presented method.

We also compare using two options to evaluate the impact of the data correction step:

1. **C-layout method w/o completion**: the presented method is applied with the correction step (Section III C.2., Data Correction).
2. **C-layout method**: the presented method **with** correction step. As expected it obtains the best result, extracting 598 complete sdata (from 521 to 598, a gain of 15%)

Table IV: Results of the data extraction.

Method	Baseline	Structural
Block	254	
c-layout model w/o correction		521
C-layout method		598

The accuracy of complete sdata based on the c-layout model is usually very good: On 20 plans, only 2% of such sdata were automatically checked as wrong (only numerical characters for names, OCR errors not being taken into consideration). Around 30% of the plans have no sdata (or represented only by graphical logos), and 20% only one single sdatum, situation not covered by our method: **to be applicable, our method requires a layout structure composed of at least 2 sdata**.

As expected the correction step increases the recall of the method (up 15%): new mandatory fields are tagged using the c-layout model. Table V shows the impact of this step for each field. The most interesting result is for the *name* field, difficult to tag since company names do not always match a specific pattern (see the pattern used listed Table VII).

Table V: Impact of the correction step based on c-layout model

Sdata fields	# guessed	# accuracy
Discipline	37	73%
Name	76	92%
STREET	34	82%
Zip	32	93%
State	0	-
Phone	42	88%
Email	42	90%
Contact	12	83%
url	6	100%

B. *Qualitative Evaluation*

50 plans were processed and the result was sent to the data owner who performed an evaluation. Since the purpose of this extraction is to feed a database, this one was also used: extracted factors were matched against database entries in order to index plans with them. Matching was performed by comparing factors fields. Fuzzy string matching was used to cope with the numerous OCR errors. The evaluation is conducted at the factor level and was very strict: when one field was not correctly extracted, the factor was considered as wrong. A single OCR error at character level was enough to consider the factor as *missed*. When a database entry was matched, the factor was considered as correct (which improves accuracy). The table below reflects this evaluation.

Table VI: Evaluation on 50 sheets

	Found	Missed	Accuracy	DB match accuracy
Without DB	61	73	45%	N/A
With DB	99	35	74%	91%

Missed factors are mainly due to the following main issues:

- Some layout structures are not properly covered.
- The OCR engine sometimes completely missed the text zone due to: very sparse text, rotated text, colored background, use of fancy fonts badly handled, names represented by logos.

As expected, using the database improves results, and allows for an indexing of the plans. The match between extracted factors and database entries is pretty reliable (91% accuracy).

V. LIST OF RESOURCES

We list here the resources used for the construction plan collection. The list of disciplines was provided by the customer. The street, phone and URL regular expressions were found on internet, as well as the US states list. The name regular expression was designed using the closed suffix list which indicates the legal status (only for USA): Inc, Co, ltd, PC, LLP, PLLC, LP, LC, PS. These are all incomplete (except the states list) or too general (zip code), but there were quickly designed and collected.

Table VII: List of lexical resources

Fields	Resources: regular expression or list
Discipline	48 terms provided by Subject Matter expert
Name	(CO|INC|LTD|PC|LLP|PLLC|LLC|LP|LC)$"
Street	"(?:.. BOX \d+)|([0-9]+ [\.\0-9-a-Z]+ (FLOOR [a-zA-Z]+|APARTEMENT|SUITE [0-9]+|SOUTH|WEST|NORTH|EAST|STREET|ST\.|DRIVE|DR|AVENUE|AVE|ROAD|WAY|RD\.|LOOP|COURT|CIRCLE|LANE|BOULEVARD|BLVD))(?:$|[^a-Z])"
State	List of state name and state abbreviations (52*2)
Phone	"[^0-9]?\(?([2-9]\d{2})\)?(-|.|\s)?([1-9]\d{2})(-|.|\s)?(\d{4})"
Email	"\w([\.a-zA-Z]+@[a-zA-Z0-9\-]+\.[a-zA-Z0-9\-]{,3})\w"
URL	"([a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3})[^a-zA-Z]"

CONCLUSIONS

We have shown, based on a real use-case, how combining layout and content information can be efficient for data extraction. This method has been also successfully applied to other cases, especially on invoices. One key point of the method is to rely on incomplete lexical resources (thus easier to create/manage). In order to circumvent this incompleteness, a model, the *c-layout* model, is generated, combining the layout and textual organization of the sdata. This c-layout model generation is only possible when several sdata are present in a same layout structure.

REFERENCES

[1] A. Dengel and F. Shafait. “Analysis of the logical layout of documents”, in D. Doermann, K. Tombre (eds.), Handbook of Document Image Processing and Recognition, Springer-Verlag, 2014, pp 177-222

[2] E. Cortez, D. Oliveira and A S. da Silva, “Joint unsupervised structure discovery and information extraction”, SIDMOD’11, Athens, 2011

[3] B. Coüasnon, and A. Lemaitre, “Recognition of Tables and Forms”, in D. Doermann, K. Tombre (eds.), Handbook of Document Image Processing and Recognition, Springer-Verlag, 2014, pp 647-677.

[4] A. Costa e Silva, A. M. Jorge and L. Torgo. “Design of an end-to-end method to extract information from tables”, International Journal of Document Analysis and Recognition (IJDAR), 8(2-3), 2006, pp 144-171.

[5] J.-Y. Ramel, M. Crucianu, N. Vincent and C. Faure. “Detection, extraction and representation of tables”, Seventh International Conference on Document Analysis and Recognition, 2003.