RI10002, 22 March 2010 Computer Science

# IBM Research Report

## Business Insight from Collection of Unstructured Formatted Documents with IBM Content Harvester

**Biplav Srivastava**
IBM Research Division
IBM Research
New Delhi - 110070. India.

**Yuan-Chi Chang**
IBM T. J. Watson Research Laboratory
Hawthorne, NY 10591. USA

**IBM Research Division**
**Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich**

# Business Insight from Collection of Unstructured Formatted Documents with IBM Content Harvester*

Biplav Srivastava and Yuan-chi Chang

IBM T. J. Watson Research Laboratory
Hawthorne, NY 10591, USA
{sbiplav@in, yuanchi@us}.ibm.com

## Abstract

Today's knowledge workers need to access, apply and reuse content created by office productivity suite such as word processor, spreadsheet and presentation. While the productivity suite revolution in the 90's freed individuals' creativity to generate content, it is increasingly difficult to effectively manage and harvest individuals' creation into knowledge of the whole. One cannot glean into a project document repository, for example, to get a summary of the status of work items. Both keyword search and social tagging fall short of functionality required to harvest and distill content for reuse.

In this paper, we report the development and experiments of IBM Content Harvester (CH), a tool to analyze and recover templates and content from word processor created text documents. CH is part of a bigger effort to collect and reuse material generated in business service engagements. Specifically, it works on unstructured formatted documents and works by extracting content, cleansing off sensitive information, tagging it based on user-defined or domain-defined labels, and making it available for publishing in any open format and flexible querying. As a result, one can search for specific information based on tags, aggregate information regardless of document source or formatting peculiarities and publish the content in any format or template. CH has been applied to a broad variety of document collections containing hundreds of documents, including live engagements, to promising effect.

## 1 Introduction

With the proliferation of office productivity tools, it has been recognized that users need better ways to search, organize and reuse their content when appropriate. The introduction of text indexing and search addressed certain challenges in finding relevant content. However, it is still very time consuming and labor intensive to crawl through the search results in order to identify reusable content piece by piece. In this paper, we describe a tool developed to solve the harvesting and reuse challenge. Specifically, we assume the unstructured content was authored according to a template, which helps the organization of semantically meaningful content. The tool is designed to discover the template(s) from many instance documents and then extract, cleanse and store template-directed content. Such capability is necessary to build knowledge assets and tools that improve worker productivity[4].

We reference two classes of scenarios where collections of documents are generated and the desire to harvest information from them.

### 1.1 Documents in Day to Day Scenarios

We come across many types of documents in daily situations. For example, consider culinary recipes, proposals, resumes, evaluations. Here, there are initial templates which provide a broad framework for recording information. Such documents start from mandated templates but individuals can make changes depending on the information they are recording. Given the documents, one would want to seek information from the document collection and take decisions. For example, a person preparing 4 dishes would want to look at the recipes and make a consolidated list of ingredients to buy from the market. A recruiter would want to look at the resumes and know the qualification of all the applicants he has received. A manager would want to look at the evaluations and know the the performance of a particular person over the years or the performance of all the reportees under him.

The currently available solution in these situations is for a person to open each file one-by-one and then peruse, remember and mentally compare. Or per file, one can peruse, copy and paste content of interest in a separate new document, and then compare manually collated content in the new document. This is tedious, error-prone and non-scalable.

*A shorter version of the paper appears in 15th International Conference on Management of Data (COMAD 2009), Mysore, India.

*15th International Conference on Management of Data*
*COMAD 2009, Mysore, India, December 9–12, 2009*

### 1.2 Documents in IT business

Team-based document creation is wide-spread in Information Technology (IT) software and services business. An example of the former is software design documentation and of the latter is contract scope in services engagements. The documents are created with commercial word processors like Microsoft Word, Lotus Symphony and Open Office. The teams start from mandated templates and team members add content. In the process, they invariably change a document's structure if expedient to capture some specific information.

It is not uncommon for teams to create 100s of documents created on a IT project. Such documents are stored in repositories (including file systems) that traditionally provide only key-word based search. Over time, we have a large collection of documents following different templates: documents from a single client but multiple types of documents (e.g., design, test, specification), documents from different clients and same or multiple types of documents.

A potential consumer of the document will be interested in the content in the document but usually not the template because the latter changes from one client setting to another. For example, many teams will be interested to reuse the design of an online checkout and international shipping feature as implemented on a retail website, but the template in which the information is documented is of little consequence. Consider we have information about this feature from two clients projects in the repository. Since the technology is complex, we would want to compile the full list of features that previous teams have considered. However, the documents available will have an inter-mix of content and the formatting information, and it is very daunting for a third team to sift through the complete feature documents to compile the information they need.

A large services organization does thousands of projects in a year. Given the scale of documents they produce, it is practically impossible today to refer to a subset of design documents, all created in different enagements, and potentially with different templates, and try to obtain content from them in an integrated form. The resulting content can then be published with any template that the new client may want.

### 1.3 Summary and Contributions

We present the Content Harvester framework to solve the challenges. It works on collections of unstructured formatted documents and requires the user to specify the textual segment of information they want to extract, what identifiers they want to replace and what to label the extracted content. The tool first separates the textual and non-textual (including formatting) content[1], and then uses segment specification to extract information of interest. It maintains basic structure of extracted content - paragraph, list and table, and represents it in XML. Next, sensitive information is cleansed off and then extracted content is tagged with labels from the segment specification. The result is available for publishing in any open format like WordML, HTML or PDF by simple XML transformations, and flexible XML queries can be done over it. As a result, one can search for specific information based on tags, aggregate information regardless of document source or formatting peculiarities and publish the content in any format (Word, pdf, html) or template. A scaled-down version of the CH tool is publicly available[2]. In addition, methods are provided to learn templates and segment specifications.

Our contributions in the paper are:

- A format-independent methodology to segment unstructured formatted documents into units of text for cross-document processing.
- An architecture to extract content with user annotated format-independent landmarks for repurposing and reuse.
- A statistical method to recommend segmentation (or landmark) boundaries for user annotation in a cluster of documents.
- A statistical method to analyze and identify clusters of similar documents that are likely to stem from the same templates, using parsed text units.
- Experimental results and pilot experience on the efficacy of the approach.

Content Harvester has been applied to a broad variety of document collections containing 100s of documents, including live engagements, to promising effect. In the rest of the paper, we present the problem, describe our approach and an implementation, discuss initial empirical & pilot results, illustrate CH's ability to bring new insights and conclude with a review of related work.

## 2 Problem

The problem setting is that we have a collection of original documents produced with a document processor like Microsoft Word. A document can consist of headings, paragraphs, lists, tables, images, non-textual generic objects, and any of their combinations (e.g., lists inside tables). The document's content is annotated with formatting/visual cues. A person can identify a particular content of interest by textual markers that serve as content boundaries. Since a marker is also part of the document, it can consist of text and formatting styles.

More precisely, we use the following terms:

- Marker: A piece of text that a user can view in a document, or pre-defined special positions in a document – document start and end.

[1] But keeps record of what was found.

[2] At: http://www.alphaworks.ibm.com/tech/contentharvester

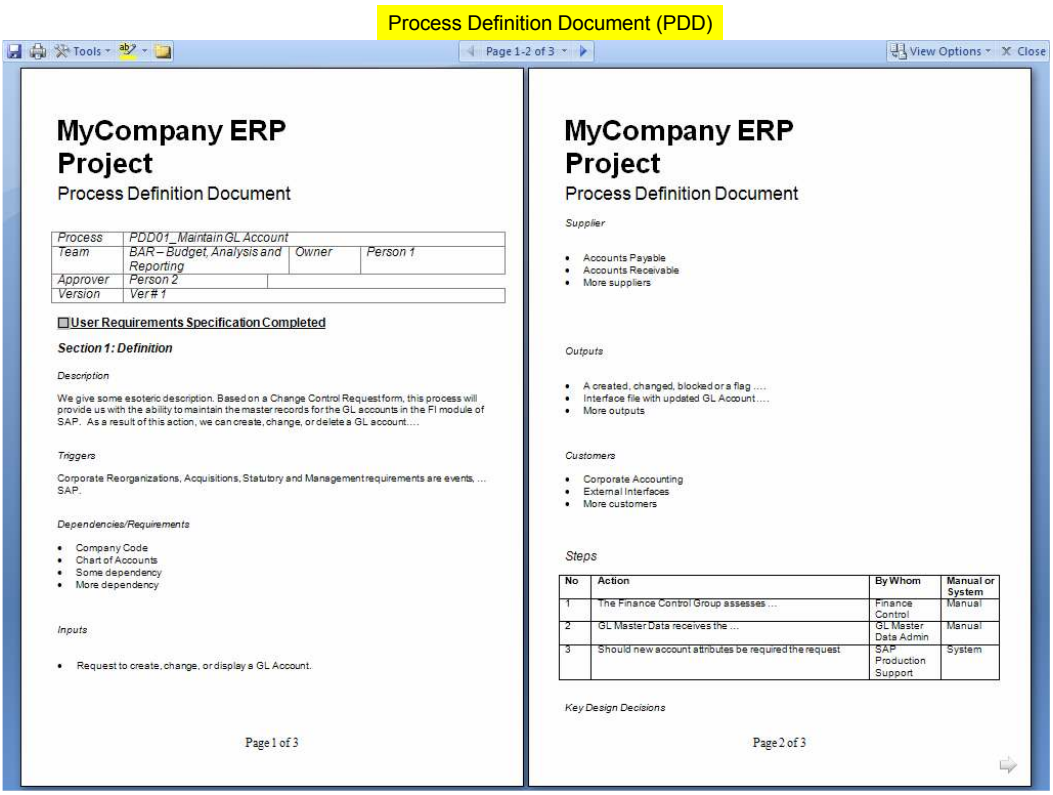


Figure 1: The original Word file of a running example.

- Formatting instructions: bold, italics, underline, font, list type, table, cells, section hierarchy, embedded objects, images, and so on.
- Document fragment: A contiguous fragment of original content of a document including formatting instructions, demarcated by start and end markers.
- Segment or extracted content: A contiguous fragment of content along with its demarcating start and end markers. The formatting instructions are absent in a segment.
- *Original* document: A sequence of one or more document fragments.
- *Harvested* document[3]: A sequence of one or more segments.

Consider the example original document shown in Figure 1. Text *Process* and *Team* are markers. The process name between the two markers is a piece of content the user may be interested in. The two markers and the process name constitute a segment. Note that the three texts are different cells of a table in the example.

[3]The prefix *original* or *harvested* will be dropped when the type of document is clear from context.

The goals of harvesting content from original documents of a collection are to: (1) extract segment(s) with content of interest (2) cleanse extracted content off any sensitive context (3) tag extracted segments with a set of given labels (including the case when the labels come from a known model) (4) enable tag-based and content-type search on extracted content (5) allow output to be published in any encoding format or document type, and (6) identify documents following a common template in a pool from different sources.

## 2.1 Identifying the Challenges

We have some difficult challenges dealing with intermixing of formatting and text and just noisy text.

- How does one robustly demarcate a piece of text? Even a single word, when parsed, could be a set of fragmented nodes containing the different characters of the word and individual formatting attributes.
- How can a user robustly specify a portion of content as being of interest? *What you see is not what you get!*: Microsoft Word has support for *hidden/ vanish* text. So, it is not easy to look at a section header and be sure that it is really what the parser will also find.

- How does the tool extract a piece of content robustly regardless of the formatting seen in a file? The same method has to find content between text boundaries, table cells, formatted headers, etc.

- How to handle noisy documents? Headers and cells of a table can be empty.

There are also practical challenges since we want the tool to be available to non-technical users (i.e., users not aware of information extraction methods): how to package all the capabilities desired in a useful manner, how to represent extracted content, how to do tagging of content, what query mechanisms to support and what guidance to give about when *not* to try content extraction with our approach.

# 3 Solution for Harvesting Documents following a Common Template

The pseudo-code for Content Harvester (CH) is shown in Figure 2 and the architecture of a prototype system is shown in Figure 3. CH works on collections of unstructured formatted documents ($D$) and requires the user to specify the textual segment of information they want to extract and what to label the extracted content with ($L$), and what sensitive identifiers they want to replace ($R^c$). We now explain the main steps in subsequent subsections.

*Algorithm:* **HarvestContentFromSimilarDocs**
*Inputs:* A set of documents $D$ following a common template,
Landmark specification of what to extract, $L$
Rules expressing what phrases to cleanse and the new phrase, $R^c$
Flag for output format
*Output:* A set of cleansed documents, $D^c$
Cleansing report, $R$

*Pre-processing:*
1. For each document $d_i$
2. Convert $D_i$ to XML representation
*Main Steps for each document*
1. For each document $d_i$
2. Parse $d_i$
3. Group characters along word boundaries
4. Group words along paragraph (formatting) boundaries
5. Record paragraphs with basic source formatting information
6. Remove non-textual and non-formatting content
7. Uselandmarks $L$ on textual paragraphs
8. Use markers to identify segments
9. Use formatting information to identify basic content structure
10. Use heuristics to overcome noisy/ missing text
11. Apply $R^c$ on extracted content
12. Use labels from $L$ to tag extracted content
13. Publish $D_i^c$ in XML
14. Record statistics about $d_i$ for reporting
15. Publish final $R$ with overall and per-file statistics

Figure 2: Pseudo-code of Content Harvester.

## 3.1 Parsing Word Documents

In recent years, modern word processing software began to adopt XML as a supported file format, which allows easier access to text content stored in the files. While XML is self-describing, these XML file formats primarily focus on the presentation and rendering styles of the text content, such as character formatting, paragraph spacing, lists, tables and figures, etc. There is a lack of provision for user-annotated constructs to describe the content. Hence a tool like ours is still required to analyze and identify bodies of semantically similar content.

We use two commonly applied standards as examples to describe our approach to segment text content for further analysis, i.e. Office Open XML (OOXML) and OpenDocument Format (ODF). Microsoft Office suite software supports OOXML while Star Office, Google Docs and IBM Lotus Symphony support ODF. In this paper, we will focus on word processing section of the above standards but our approach may be more generally applicable to other sections such as spreadsheet and presentation.

Our approach to extract the raw text content (steps 2-6 in Figure 2) is to identify paragraph boundaries and reassemble texts falling within the same paragraph boundary as a single text block for subsequent analysis. If one views a text file as a sequence of character streams, we find paragraphs to be the basic and natural segmenting markers to group together characters semantically. This view is fairly different from say, indexing a text document for text search, where the natural segmentation will be at the word level.

In the WordProcessing ML section of the OOXML, paragraphs are identified the <w:p> tags, where $w$ is the namespace declaration of WordML. Under each <w:p> tag, there may be styling information about the paragraph such as headings, bulleted lists or numbered lists. The paragraph may also contain one or more character formatting instructions under the <w:r> tags, which define styling information such as bold, italic, underline or color for the associated characters under the <w:t> tags. An example of key tags used by our parser is shown in Figure 4.

In contrast to OOXML, the OpenDocument Format has different tags from headings, such as <text:h>, as well as paragraphs, such as <text:p>. The attribute name *stylename* is used to describe the text styling applied to the content of the XML node. As shown in 5, the example similarly has a subsection heading named *Section 1: Definition*. However, its XML node is at a much higher level and not deeply buried in the nesting. Such document tree layout aids XML parsing and transformation greatly with simpler logic.

In ODF, the full styling information is separated from text content and declared in a *styles.xml* file. An example is shown in Figure 6, where the two styles used in Figure 5 are declared. XML node attributes are applied freely to declare various styling properties. Again we observe the design of the XML tree structure to be aimed at easier parsing and transformation.

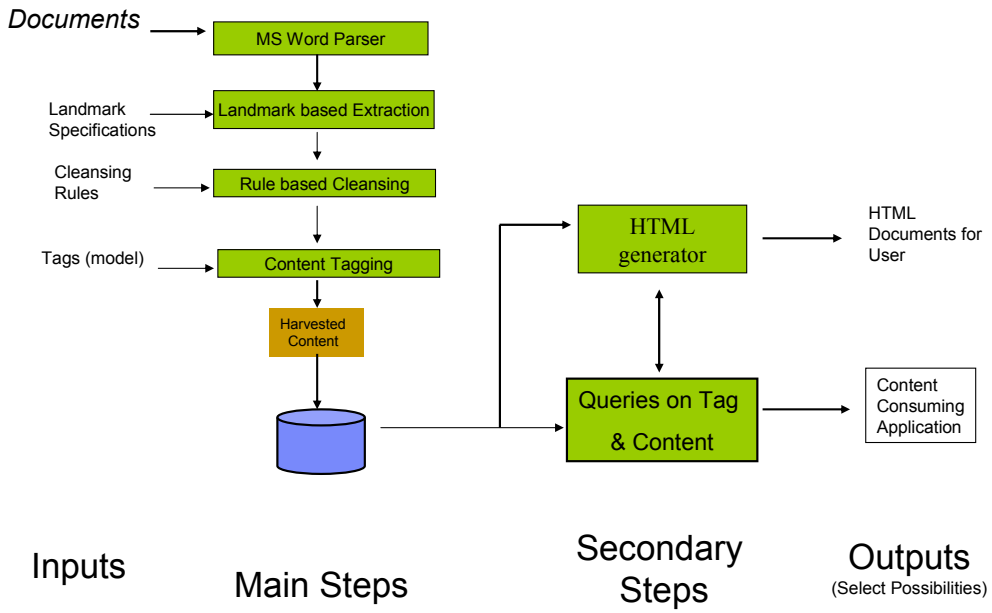


Figure 3: Architecture of Content Harvester.

### 3.2 Landmark Based Extraction

The previous section described how textual and non-textual (including formatting) content is separated. Now, textual content of interest in the document has to be identified. For this, segment specification called *landmark* is used to extract information of interest. We differentiate landmarks from the conventional notion of segments in image and text extraction literature because there markers, or segment boundaries, are seen in the input's bit stream. In our case, the characters in the markers may be fragmented.

We define *landmark* as a specification of a segment whose start pattern is known and its end pattern is optional. If the end pattern is missing, the end of the segment is marked by start pattern of any known landmark specification. It is easy to see that:

- If end is known, the segment becomes neighborhood dependent.
- If end is unknown, the segment is neighborhood independent.

We assume that the user will look at a sample of the documents and create landmark specifications. Consider documents from a Pharmaceutical engagement describing business process descriptions. Figure 1 shows one of the documents (slightly masked). Suppose we want to get the process name and steps from these documents. Process name is contained within a cell of a table and its predecessor cell has text marker, *Process*, and successor cell has text marker, *Team*. Figure 8 shows an example specification of the landmark. Here, *startMarker* is specified and *isStartMarkerSep* notes that the content is separated from the marker by non-textual separator. The end marker is optional. The field *modelReferenceTo* specifies the label to assign the extracted content and *isRepeating* flags absence of tabular content.

For tables, *isRepeating* is set to 'yes' and *header*s are specified for each column of the table. An example for steps is in Figure 9. Note that the first column of the table has an empty value in the header. The output of the extracted content in shown in Figure 7.

The extraction of content is described in steps 7-10. CH maintains basic structure of extracted content - consecutive text, list and table. It also uses rules on formatting information from parsing phase (step 10) to handle noise. Some rules are:

- Ignore empty white spaces that are not part of any segments.
- If an empty white space is within a cell, consider it as valid content.

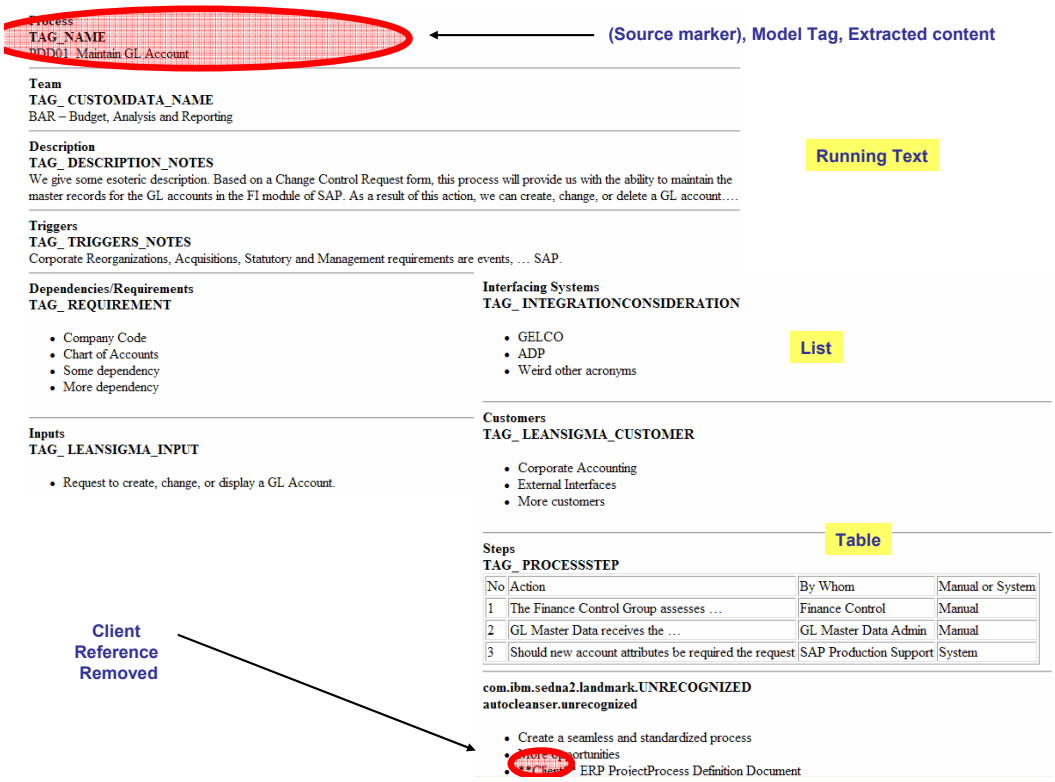


Figure 7: Extracted output in the example.

- If an item is part of a list and is empty, ignore it.

In Figure 7, the middle rule was used on *Steps* table so that the first column header (empty) is recognized. Note that the basic structure of *Process* that it is a piece of continuous text, and of *Steps*, that it has tabular description, is preserved in extracted content.

### 3.3 Post-processing Extracted Content

In Step 11, a regular expression rule processor applies $R^c$ on extracted content to remove references that are either non-relevant in a new context (e.g., version number) or privacy-related (e.g., client name). Then extracted content is tagged with labels from $L$ (step 12) and then published in a XML representation (step 13).

The result is available for publishing in any open format like WordML, HTML or PDF by simple XML transformations, and flexible XML queries can be done over it. As a result, one can search for specific information based on tags, aggregate information regardless of document source or formatting peculiarities and publish the content in any format (Word, PDF, HTML) or template.

## 4 Solution for Harvesting Documents from a Mixed Pool

In the previous section, we assumed the documents to belong to a common template. However, the most common case is that the user has a collection of documents whose provenance and templates they are unsure of. In this case, we propose automatic methods to work with these documents and extend CH's applicability. First we look at how to recommend markers (landmark boundaries) for user annotation in a cluster of documents. Next, we consider how to identify a subset of documents that share a common template.

### 4.1 Proposing Markers in a Mixed Pool

Figure 10 gives the method we use to find the potential markers in a pool of documents following mixed templates. The method first parses the documents to identify demarcating text fragments (pre-processing, lines 1-3) and then looks at their statistical significance to determine if a fragment is a potential marker (lines 1-3).

This pre-processing step has at least two potential usage: (a) it is used by the method in next section to find document clusters, and (b) in the scenario that a user sees a new dataset, the output markers of the method can serve as a starting reference while defining landmark specifications.

```
...
<w:p wsp:rsidR="00000000" wsp:rsidRDefault="0033549C">
    <w:pPr>
        <w:pStyle w:val="Heading2"/>
    </w:pPr>
    <w:r>
        <w:rPr>
            <w:rFonts w:ascii="Arial" w:fareast="MS PGothic"
                w:h-ansi="Arial" w:cs="Arial"/>
            <wx:font wx:val="Arial"/>
            <w:color w:val="000000"/>
            <w:b/>
            <w:sz w:val="20"/>
            <w:sz-cs w:val="14"/>
        </w:rPr>
        <w:t>Section 1: Definition</w:t>
    </w:r>
</w:p>
<w:p>
...
```

Figure 4: An example showing the relationship between <w:p>, <w:r>, <w:t> tags and their associated style tags.

```
...
<office:text>
...
  <text:h text:style-name="Heading_20_2" text:outline-level="2">
      Section 1: Definition
  </text:h>
  <text:p text:style-name="Text_20_Body_20_Single">
      This is the beginning of section 1.
  </text:p>
...
</office:text>
...
```

Figure 5: In ODF, <text:h> and <text:p> are used to tag content of the same style, which is referenced by the style-name attribute.

### 4.2 Discovering Documents in a Common Template From a Mixed Pool

Figure 11 gives the pseudo-code of a statistical method to analyze and identify clusters of similar documents that are likely to stem from the same templates using parsed text units. It uses the potential markers output from *FindLikely-Markers* on each document to identify segments in the document (pre-processing, lines 1-3). Then the method builds a profile of each document using the segments present in it (lines 1-6) and calls a clustering method to identify set of documents with similar profiles (lines 7-8).

## 5 Experiment

We now discuss how the presented methods perform in practice. For content extraction (Section 3), we did an evaluation on a diverse dataset as well as verified the method in the field by undertaking a pilot study wherein the tool was released to a team within IBM that cleanses and harvest design documents, and measured their performance. For harvesting documents from mixed pool (Section 4), we did a controlled evaluation using two different datasets.

```
<style:style style:name="Text_20_Body_20_Single"
  style:display-name="Text Body Single" style:family="paragraph"
  style:parent-style-name="Default_20_Text" style:class="text">
      <style:paragraph-properties fo:margin-top="0in"
        fo:margin-bottom="0.0835in"/>
</style:style>
<style:style style:name="Heading_20_2"
  style:display-name="Heading 2" style:family="paragraph"
  style:parent-style-name="Heading"
  style:next-style-name="Text_20_Body_20_Single">
    <style:text-properties fo:font-size="14pt" fo:font-style="italic"
      fo:font-weight="bold" style:font-size-asian="14pt"
      style:font-style-asian="italic" style:font-weight-asian="bold"
      style:font-size-complex="14pt" style:font-style-complex="italic"
      style:font-weight-complex="bold"/>
</style:style>
```

Figure 6: In ODF, text styling is declared in a separate styles.xml file. This examples shows the declaration of *Text_20_Body_20_Single* and *Heading_20_2*.

```
<landmark
        landmarkId="P1"
        isRepeating="no"
        isOptional="no"
        startMarker="Process"
        isStartMarkerSep="yes"
        modelReferenceTo="processTitle"
        isEndMarkerSep="yes">
</landmark>
```

Figure 8: An example of specification of a basic landmark.

### 5.1 Extracting Content from Documents

Here, we investigate how *HarvestContentFromSimilar-Docs*() performs across different data sets with diverse characteristics: documents from day-to-day activities to software/IT business, average page lengths from very small (even 1) to large (≈60-70), different scale in number of documents in a dataset (ranging from a couple to 242), differing fidelity to their common template, and the scale of number of tags of interest (3-24). In the analysis that follows, we consider 5 data-sets corresponding to process design in SAP projects in different industries, 1 from detailed process design in an Oracle project, 1 dataset of personnel evaluations and another on recipes for cooking dishes. Note that the tool has been released publicly and we are aware that it has been downloaded uniquely $\succ 60$ times. Furthermore, we are aware of CH being tried on $\succ 50$ different datasets. The analysis presented is only for a controlled set spanning diverse dataset characteristics.

Table 1 presents a preliminary evaluation. The columns represent average size of documents, # docs for the experiments, the ratio of the number of landmark specs created for each tag, the avg. % of document's content extracted and retained, the average number of tags applied, avg. processing time and finally a review of whether the dataset is amenable to content extraction cost-effectively. All columns are self-explanatory except the fourth one which

*Algorithm:* **FindDocumentClusters**
*Inputs:* A set of documents $D$ following unknown templates
*Output:* A partition of $D$, i.e., set of nonempty subsets of $D$ (clusters representing a template) such that every $d_i$ is in exactly one of these subsets.

*Pre-processing:*
1. $LM$ = Use *FindLikelyMarkers* on $D$ to find markers.
2. For each document $d_i$
3. Parse $D_i$ to create an ordered list of fixed text segments using $LM$

*Main Steps:*
1. Create a co-occurrence matrix to record the $\prec$doc, segment$\succ$ pair
2. Calculate the frequency of each segment across the document repository
3. Apply thresholds to disqualify segments that are rare or too frequent
4. Create a feature vector out of filtered segments
5. Each document corresponds to a row
6. Mark presence of a segment by 1 and absence by 0
7. Perform clustering on the feature vector set to determine the number of clusters
8. Documents following common templates are in same cluster (have similar feature vectors); Prepare output

Figure 11: Pseudo-code of a method to find documents following a common template.

```
<landmark
        landmarkId="RC1"
        isRepeating="yes"
        isOptional="yes"
        startMarker="Steps"
        isStartMarkerSep="yes"
        modelReferenceTo="OtherSection/title"
        isEndMarkerSep="yes">
        <headers>
                <header name="" order="0">
                        </header>
                <header name="Action" order="1">
                        </header>
                <header name="By Whom" order="2">
                        </header>
                <header name="Manual or System"
                        order="3"> </header>
        </headers>
</landmark>
```

Figure 9: An example of specification of a tabular landmark.

we call *variability ratio*. The ratio measures how many landmarks are needed on an average to extract content for each tag in the dataset. Hence, the value conveys how disparate documents in a dataset are. If the ratio is 1, a single landmark is sufficient to get content for a tag from the whole dataset. Hence, the dataset indeed follows the underlying template consistently. The higher the value from 1, the more likely are the documents in the dataset to vary from a common template. We note that 5 of the 8 datasets in the experiment were conforming to a common template with their *variability ratio* (fourth column) in [1, 2] and another at 2.6. In fact, *Pharma-1* has 242 documents and yet has the ratio at 1.04. In contrast, the two data sets of *High Tech* have significant variability in their documents.

The results are very encouraging and show that the users could easily harvest content for a large proportion of tags of interest ($\succ$ 70% and even 100%) across the range of datasets. The content corresponding to these tags could be very specific and low (e.g., 8% of total content in *High Tech*) or as high as 93% in *Pharma-1*. The time to process a document varies with page length but it is about a minute/ document for a typical 10-page document. The extensive runs on large datasets of *Pharma* and *High Tech* indicate that the CH method is robust.

*Algorithm:* **FindLikelyMarkers**
*Inputs:* A set of documents $D$ following unknown templates
*Output:* A set of markers, $M$

*Pre-processing:*
1. For each document $d_i$
2. Convert $D_i$ to XML representation
3. Identify possible markers using Steps 2-6 of Figure 2

*Main Steps on document pool*
1. Build a list of markers in the pool and their frequency
2. Establish thresholds to establish segments that are neither rare (lower limit) nor overly abundant (upper limit)
3. Filter and return markers that are within the thresholds

Figure 10: Pseudo-code of a method to find markers in a mixed pool of documents.

The experience of Oil&Gas-1 dataset is peculiarly interesting. Although the Word documents here were template-wise consistent, they were hard to work with due to *hidden/vanish* feature of Microsoft Word whereby invisible text is included in a document. The user would look at an original document to determine markers and content of interest but the tool may or may not encounter the same marker pattern and content. So, either extraction would fail or different content than what the user expected would come. We had to provide a separate tool to the users to expose hidden text and this solved both the problems.

### 5.2 Pilot Study of Content Harvester on Design Documents

We conducted a pilot study of Content Harvester to understand how feasible the tool's approach is in harvesting and cleansing large document collections in practice. The pilot ran for 5 weeks and involved a user group within IBM that

| Dataset Name | Doc Size (in pages) | # Docs Processed | Ratio (Raw #s): # Landmarks/ # Tags | Parsed & Retained Content(%) | Tags Found (%) | Avg. Proc. time per doc (secs) | Comments |
|---|---|---|---|---|---|---|---|
| Pharma-1 | 4-10 | 242 | 1.04 (25/24) | 93 | 86 | 40 | Good |
| Pharma-2 | 6-10 | 27 | 2.6 (18/7) | 84 | 83 | 81 | Good |
| Oil&Gas-1 | 10-12 | 8 | 1 (10/10) | 29 | ≈80 | 103 | Good (difficult due to hidden text) |
| High Tech-1 | 35-60 | 21 | 4.3 (13/3) | 8 | ≈70 | 520 | Bad (variability) |
| High Tech-2 | 35-60 | 63 | 4.3 (13/3) | 8 | ≈70 | 329 | Bad (variability) |
| Oracle-Des | 15-20 | 2 | 1.7 (5/3) | 16 | 84 | 360 | Good |
| Misc-1 | 4-7 | 3 | 1 (6/6) | 81 | 100 | 44 | Good |
| Misc-2 | 1 | 3 | 1 (5/5) | 75 | 87 | 3 | Good |

Table 1: Evaluation of Content Harvester on Different Datasets.

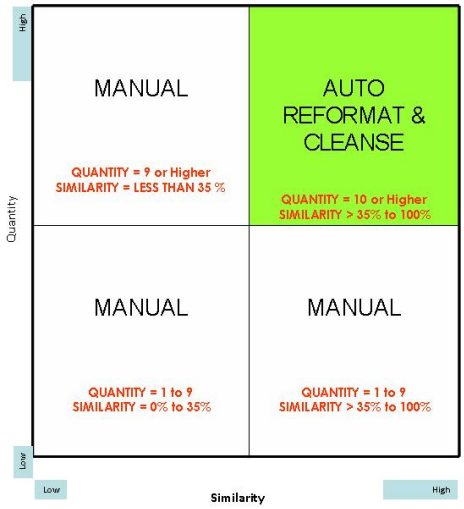


Figure 12: Benefit-Quadrant for Content Harvester in a Pilot Study.

manually cleansed design documents and formatted them to a standard form. The pilot's version of the tool was an enhancement of Content Harvester presented here – the tool was aware of design tags and hence could enforce consistency checks, and could generate output in more formats. The pilot was designed to find where most time would be spent using the Content Harvester tool and for what types of documents this method would be cost-effective. The complexity of a document set was measured in terms of the number of documents and their average number of pages. The similarity of the documents were in terms of the number of common segments/ landmarks (estimated by approximate number of sections, tables, etc). 6 datasets of equal sizes but different characteristics (#pages, formatting and template consistency) were used.

We found that writing landmark specifications was the most time-consuming part of using the tool. However, this was a one-time cost and would be amortized if the number of documents to be processed was not very small and the documents followed a basic level of similarity. Figure 12 shows the result of where the tool would be useful to the target users. For documents of about 10 pages and moderate similarity, which is common for software design, the tool would be more cost-effective than manual cleansing and re-formatting. CH has the added benefit of tagging the harvested content and allowing it to be published in any open format using style-sheets. This makes the output seamlessly usable by other software tools.

### 5.3 Finding Documents in a Common Template From a Mixed Pool

For this experiment, we used a controlled scenario of 2 datasets. We had a repository with 20 documents: 11 from SAP project at Client-1 and 9 from SAP project at Client-2. The templates at the two clients were fairly different even though all were documenting SAP process descriptions (PDDs).

Some statistics of the data-set were:

- Total # of text segments in the documents = 1146
- Min frequency = 1, Max frequency = 20
- (Min,Max) Threshold = $\pm$ 60% of #docs = (8,32)
- After applying thresholds, # text segments left reduces from 1146 to 70

The experiment was run with *FindDocumentClusters* along with Weka3.2's k-Means clustering algorithm[4]. Figure 13 shows the result. In the upper part of the Figure, a fragment of the feature profile of each document is shown consisting of absence or presence of segments in them (indicated by 0 or 1, respectively). In the lower part, the centroids of the clusters found by k-Means is shown. The two datasets consisting of 11 and 9 documents are correctly grouped in the two clusters.

The results indicate that the technique is promising. However, the performance is expected to be sensitive to the minimum number of documents in each data set and the number of datasets in the repository. Future evaluation will investigate this aspect further.

## 6 Business Insight with CH

We now illustrate the kind of business insight possible with CH. Recall the original document shown in Figure 1. It is

[4]See Weka at http://www.cs.waikato.ac.nz/ ml/weka/.

Document Clustering by Segment Feature Vectors

Segments were filtered down to only keep those with frequency +- 60% of #docs in repository

| Legend | M1 | M2 | M3 | M4 | M5 | M6 | M7 | M8 | M9 | M10 | M11 | M12 | M13 | M14 | M15 | M16 | M17 | M18 | M19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| F2 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| F3 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| F4 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| F5 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| F6 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| F7 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| F8 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| F9 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| F10 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| F11 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| F12 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| F13 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| F14 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| F15 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| F16 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| F17 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| F18 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| F19 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| F20 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

**Experiment using kMeans**

Cluster centroids:

Cluster 0

F10 0.0 0.0 0.0 0.0 1.0 0.8181818181818182 0.0 0.8181818181818182 0.0 0.0 0.0 0.0 0.0 0.0 0.8181818181818182 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.9090909090909091 0.0 0.9090909090909091 0.0 0.0 0.0 0.9090909090909091 0.0 0.0 0.8181818181818182 0.9090909090909091 0.0 0.0 0.0 1.0 0.0 0.0 0.8181818181818182 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.8181818181818182

Cluster 1

F1 0.8888888888888888 1.0 1.0 1.0 0.0 0.0 1.0 0.0 1.0 1.0 1.0 0.8888888888888888 1.0 1.0 0.0 1.0 1.0 1.0 1.0 0.0 1.0 1.0 0.0 1.0 1.0 1.0 0.0 1.0 0.0 0.8888888888888888 1.0 0.8888888888888888 1.0 1.0 0.0 1.0 1.0 1.0 0.0 0.8888888888888888 1.0 0.8888888888888888 0.0 1.0 1.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 1.0 0.0 1.0 1.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.0

Clustered Instances

0 11 ( 55%)

1 9 ( 45%)

Figure 13: Using k-means to identify the correct document clusters.

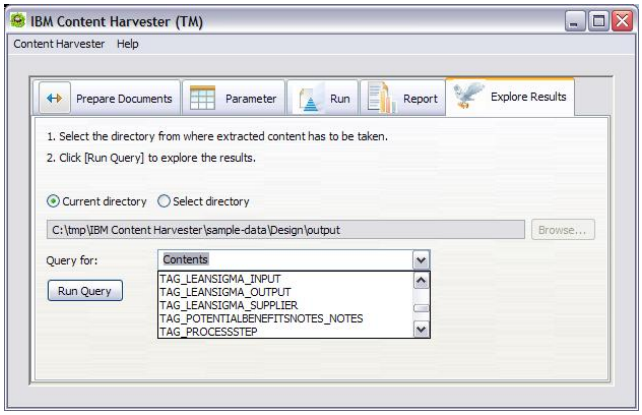


Figure 14: Selecting content based on tags.

available as part of the *Design* dataset available from CH website. In the released tool, the user can use tags to select harvested content of documents in a dataset and post XML queries. In Figure 14, the user is looking for content available based on all the tags declared in the dataset[5]. She selects the tag *TAG_PROCESSSTEP*.

In Figure 15, the content associated with the tag is shown for all the files. The documents in the dataset were differing in common structure as the two tables have different first column; hence multiple landmarks were needed. But once extracted and tagged with CH, the content can be searched across the dataset and new insights be found. Note that besides content, the tabular structure of the process steps is retained during extraction and this can be manipulated by applications.

## 7 Discussion

CH is effective in extracting content from collections of Word files that follow a template and also in identifying documents that may follow a common template. An important side-benefit of the approach is that harvested content is now available in easy-to-consume XML format to support tag-based querying (available in the released tool) or integration with other tools (see [7] for an example).

However the tool can be limiting in some situations.

### 7.1 Limitations

*User has to know about the documents at some level and specify the landmarks in terms of low level information*. She has to select a subset of documents and identify characteristics of the segment whose content they want. The user

[5]The tool manual gives the details of how this can be done.

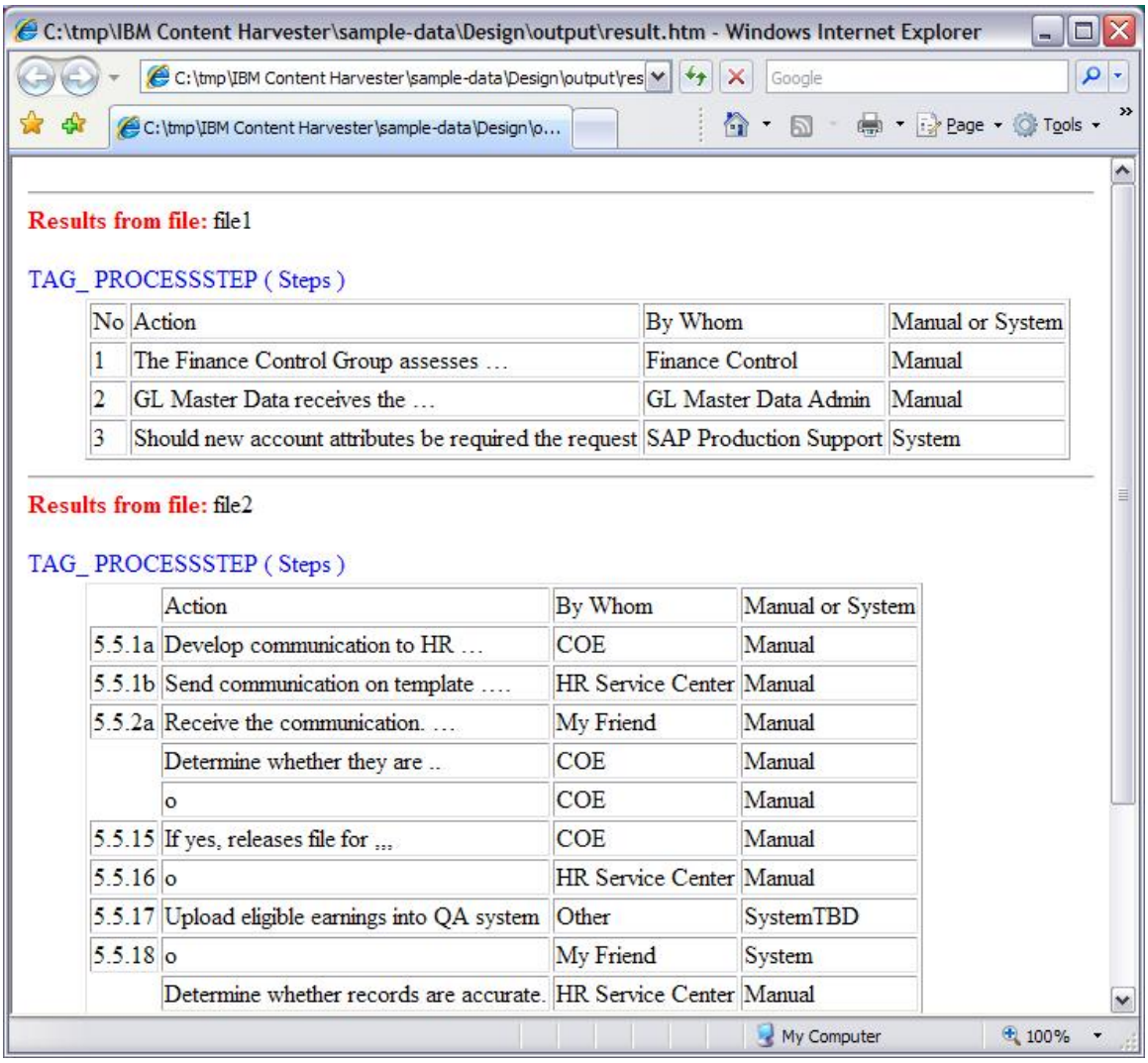


Figure 15: Example of an application enabled by CH. The result for XML query for tag *TAG_PROCESSSTEP* is shown across documents.

has to identify the begin marker, optionally the end marker, whether the content is separated from markers, whether the landmark is mandatory, if the landmark covers a table, and if so, then its headers. The user may be looking at a peculiar document to decide landmarks and this may lead to a number of iterations before satisfactory content is extracted from the collection. The method *FindLikelyMarkers* is helpful but not sufficient.

*The landmark specification is context-independent*. If the document has a similar segment at multiple locations in the file, it is difficult to specify that only a particular instance has to be extracted. Currently, the landmarks are treated independent of context and the tool can extract the first (or alternatively all) matching content. The lack of context specificity may theoretically be an issue but we encountered it very rarely in practice.

We see the limitations as avenues for future work.

### 7.2 Related Work

In the past decade, there was significant amount of research in the domains of information extraction[6], retrieval, data mining and XML that touched upon the challenges of extracting content or analyzing document structure. The seminal paper by Abiteboul defined key aspects of semi-structured data as well as its models and query languages [1]. Specifically, in our problem setting we encountered what Abiteboul defined as the aspects of irregularity, implicity, and blurring boundary between schema and data. The framework of thinking set forth in the paper suggests the missing theory of semi-structured data, which the academia is still seeking since. A related paper by Nestorov presented ways to extract object-graph like schema from semistructured data[5].The emphasis however is to assign approximate data types and relationships in a collection of semistructured data. The schema extracted thus has less use in our problem context.

Wang and Liu first expanded the traditional set-based

data mining problem into mining hierarchical tree structure and subtree mining[8]. In contrast to the IR research which typically focuses on extracting the structure of a single document, their work was formulated on requiring a minimum number of occurrences of subtrees. The formulation allows multiple common subtrees to be identified as long as their occurrences exceed the specified minimum. In[9, 10], a variation of the problem was introduced to discover frequent subtrees within a single large hierarchy and the authors introduced space efficient data encoding for mining performance. Similar work with improved mining algorithms can be found in[2]and [3].

We have found the main challenge in applying the frequent tree mining techniques lies with the document structure recorded in these word processing XML standards. The frequent text markers identified using our approach are not text styling tags used to represent the documents. Therefore while there are lots of frequent subtrees due to text style reuse, these frequent subtrees are independent from the content landmarks we wish to identify. Furthermore, we have seen user-defined templates simply signaling a section heading with an extra blank line immediately after. Human readers can infer the context and meaning of the section heading but it does not show up in special XML tags or as a separate node in the document hierarchy. In these cases, a tree-based analysis algorithm will fail to identify the new section.

## 8 Conclusion

Content Harvester (CH) tackles a pressing hurdle in asset reuse which is how to get information from documents and improve consumption. CH allows harvesting of unstructured, formatted documents by extracting content, cleansing off sensitive information, tagging based on user-defined names and making it available for publishing in any open format and flexible querying. The current version works on MS Word but the approach extends to Open Doc standard also as explained. CH has been applied to 100s of documents from a broad variety of sources, including live engagements, to promising effect. We presented experimental results on document harvesting's effectiveness, pilot experience of the tool's feasibility, and statistical methods to work with unorganized documents to find subsets of similar documents on which the tool can work effectively. The approach is a stepping stone to gain business insights into collections of unstructured documents, and to that end, the released tool supports tag-based querying while integration of the harvested content with other analytical tools is shown in [7].

### 8.1 Acknowledgements

We thank Swaroop Chalasani and Sridhar Maradugu for their contributions to the implementation of the Content Harvester tool, Kathy Byrnes for pilot guidance, and Richard Goodwin, Juhnyoung Lee, Debdoot Mukherjee and Vibha Sinha for useful discussions.

## References

[1] S. Abiteboul. Querying semi-structured data. In *Intl. Conf. Database Theory*, pages 1–18, 1997.

[2] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient substructure discovery from large semi-structured data. In R. L. Grossman, J. Han, V. Kumar, H. Mannila, and R. Motwani, editors, *SDM*. SIAM, 2002.

[3] T. Miyahara, Y. Suzuki, T. Shoudai, T. Uchida, K. Takahashi, and H. Ueda. Discovery of frequent tag tree patterns in semistructured web documents. In *PAKDD '02: Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 341–355, London, UK, 2002. Springer-Verlag.

[4] R. Mohan, B. Srivastava, P. Mazzoleni, and R. Goodwin. Challenges in moving from documents to information web for services. In *In 7th Information Integration on the Web (IIWeb-09) workshop at IJCAI, Pasadena, USA*, 2009.

[5] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. *SIGMOD Rec.*, 27(2):295–306, 1998.

[6] S. Sarawagi. Information extaction. In *Foundations and Trends in Databases, Vol. 1, No. 3, Pg. 261 to 377*, 2007.

[7] B. Srivastava and D. Mukherjee. Organizing documented processes. In *In IEEE Services Computing Conference, Bangalore, India*, 2009.

[8] K. Wang and H. Liu. Discovering structural association of semistructured data. *IEEE Trans. Knowl. Data Eng.*, 12(3):353–371, 2000.

[9] M. J. Zaki. Efficiently mining frequent trees in a forest. *ACM SIGKDD Conf.*, pages 71–80, 2002.

[10] M. J. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. In *IEEE Transaction on Knowledge and Data Engineering*, pages 1021–1035, 2005.