AHA: Asset Harvester Assistant

Debdoot Mukherjee, Senthil Mani, Vibha Singhal Sinha, Rema Ananthanarayanan, Biplav Srivastava, Pankaj Dhoolia, ‡ Prahlad Chowdhury

IBM Research – India, ‡ IBM Global Business Services, India
E-mail: {debdomuk, sentmani, vibha.sinha, arema, sbiplav, pdhoolia, prahlad.c}@in.ibm.com

Abstract

Information assets in service enterprises are typically available as unstructured documents. There is an increasing need for unraveling information from these documents into a structured and semantic format. Structured data can be more effectively queried, which increases information reuse from asset repositories. This paper addresses the problem of extracting XML models, which follow a given target schema, from enterprise documents. We discuss why existing approaches for information extraction do not suffice for the enterprise documents created during service delivery. To address this limitation, we present the Asset Harvester Assistant (AHA), a tool that automatically extracts structured models from MS-Word documents, and supports manual refinement of the extracted models within an interactive environment. We present the results of empirical studies conducted using business-process documents from real service-delivery engagements. Our results indicate that the AHA approach can be effective in extracting accurate models from unstructured documents and improving user productivity.

1 Introduction

Large volumes of information are created and captured in documents during service delivery. For example, business-process definitions, usecase/requirement definitions and functional/technical specifications are important deliverables in business-transformation projects. Service enterprises archive such documents into asset repositories to (1) reuse the information in future service delivery, (2) facilitate knowledge transfer, and (3) gain interesting insights from prior projects. Word processors are usually the preferred tools for document creation because of their ease of use, ubiquity, and rich formatting options. However, text documents are unstructured in nature, and simple keyword search is the only practical way to retrieve information from them, which limits the types of queries that can be performed. Therefore, enterprises are increasingly looking toward transforming such unstructured content to structured data so that deeper semantic search may be performed [16, 17].

A possible solution is to let consultants specify tags while producing the content. This can be done by either strictly enforcing document templates across the organization or mandating form-based content creation [12]. However, such specialized authoring environments are rarely used; the norm is still to create documents using tools such as MS-Word. This can occur for various reasons, such as the difficulty of enforcing mandates in a creative setting and the need to avoid placing restrictions.

We propose an alternative solution in which content creation is performed using free-form editors, and an information extractor derives structured models from the unstructured documents in a batch mode. Although numerous information-extraction approaches have presented in the last decade, these approaches are unable to deal with the high degree of heterogeneity (in both structure and semantics) typically present in enterprise documents. Thus, when applied to enterprise documents, the techniques have limited effectiveness in extracting useful models. To date, information extractors have been evaluated on data that have regular structure (e.g., web pages generated via scripts) [19, 21], outputs of scientific simulators [1], or data that have a limited number of semantic entities [5].

In this paper, we introduce the Asset Harvester Assistant (AHA), a semi-automatic tool to help extract content, from unstructured Word documents, in the form of XML-based models that adhere to a user-provided target schema. AHA takes as inputs a set of documents that loosely follow a template and parses out different information segments from the documents. It groups similar segments, potentially referring to the same entity, to derive a template that is implicit in the document set. A user examines the extracted segments and selects tags for them. AHA applies the tags to create models for each document, which may be further refined in an interactive environment.

AHA is tailormade to deal with the structural heterogeneity present in enterprise documents, and is responsive
Figure 1. Illustration of unstructured and structured requirements information; parts (a) and (b) show two variations of the same content in unstructured formats; part (c) shows the formal model for the two variations.

to large, dynamic schemas common in service domains. It is a low-investment system—in terms of the required time, effort, and end-user skill—that jumpstarts information extraction automatically. Subsequently, users can expend additional effort to improve the extraction interactively.

To evaluate AHA, we conducted an empirical study using a large set of real service deliverables. Our results indicate that, for the deliverables considered, AHA can harvest information with significant schema coverage in less time.

The contributions of the paper are:

• The presentation of an approach for extracting structured models from unstructured text documents that meets the challenges posed by enterprise data-sets; the implementation of the approach in a tool called AHA
• The presentation of empirical results that illustrate the effectiveness and efficiency of our approach

The rest of the paper is organized as follows. Section 2 introduces a motivating example. Section 3 presents the challenges in information extraction, and discusses the limitations of existing techniques. Section 4 describes our approach and the AHA tool. Section 5 presents empirical results. Finally, we conclude with pointers to future work.

2 Motivating Scenario

In a service engagement, at the end of the requirements-gathering phase, business analysts produce work products (typically, MS-Word documents) that describe all project requirements. Figure 1(a) shows a sample document containing a business requirement (B1) and two system requirements (S1, S2); the system requirements are further categorized into functional (F1, F2) and non-functional (NF1, NF2) requirements. Once the information is captured in a text format, the semantic categorization of the content as business, system, functional, and non-functional is lost. Figure 1(b) shows a different requirements document that has content with identical semantics, but structured differently and, in some cases, named differently too; for example, “Business Requirements” are called “Client Requirements.” However, the documents contain structural cues that indicate the relationships between different types of requirements. The layout helps identify which business requirement relates to which system requirement, and which functional requirement is part of which system.
tural and layout cues to infer the semantic entities and their relationships, for example, as shown in Figure 1(c). Such a (formal) model supports queries, such as

*Find all non-functional requirements that refer to system performance.*

Moreover, a structured organization of an enterprise repository enables the discovery and establishment of linkages across different types of artifacts. For example, the requirements model (Figure 1(c)) can contain references to entities mined from design or code documents for the same project. This can support sophisticated queries, such as

*Identify which requirements affect which parts of the design / implementation of the system.*

Clearly, such queries provide far greater insight than simple keyword searches. The goal of our approach for information extraction is to construct a structured model, such as the one illustrated in figure 1(c), while reducing the manual effort to the extent possible.

To facilitate the discussion in the rest of the paper, we introduce some terminology.

A **document** is a collection of text, images, and attachments structured with some formatting, which captures information about a set of related semantic entities. A **schema** consists of entities and relationships between entities; each entity has a set of **attributes** associated with it. A **tag** is a name of an entity or an attribute in a schema.

An **extraction set** is a set of documents that are fairly similar in structure and semantics, and that may be acted upon by an information extractor. An **extraction unit** is a contiguous sequence of content from a document that can be assigned a tag. Plain text, list, and rich text are different types of extraction units. A **rich text unit** is a sequence of paragraphs with embedded images, attachments, or other objects. A **list unit** is composed of smaller extraction units where each item maps to a model entity, and the child extraction units map to individual model attributes in each entity. Parts (a) and (b) of Figure 1 highlight some extraction-unit samples.

### 3 Limitations of Existing Techniques

There exists a large body of research in the area of information extraction [10, 14, 18]. **Wrappers** are programs that extract information from web pages, which may contain structured, semi-structured, or plain text. The techniques for wrapper creation broadly fall into three categories. The first category consists of rule-based techniques, in which the extraction patterns are specified using some specialized notation. Wrapper programs are then created to extract information from documents using these patterns. Cut-and-Paste [13], Tsimmis [7], and XWrap [11] are examples of this category of techniques. Lixto [3] provides a visual interface where the tool guides the user to create wrappers by clicking on appropriate structural cues in an example document. The second category consists of techniques based on supervised learning, in which wrappers are inferred based on a set of examples provided by the user. Users manually mark out content of interest on the page for a set of sample documents; the tool then infers the extraction patterns [1, 2, 8, 9]. The third category consists of techniques that automatically try to infer the content extraction patterns based on a collection of documents [15]. Roadrunner [19] generates wrappers automatically, with no knowledge of the schema, by comparing two HTML pages.

Wrapper-generation techniques also differ based on the types of cues used to identify a pattern [14]. A simple approach is to specify the start and end delimiters to indicate a pattern. More sophisticated, structure-based cues are often used to create wrappers based on location and/or type of formatting used in the content [19]. For example, in XHTML, such cues could be XPath expressions for extraction units. In some domains, semantic-based cues are also used; e.g., in [5], an ontology is used to produce the keywords and constants that are used to extract content.

Although many information-extraction techniques exist, a straightforward application of these techniques to enterprise documents does not work well because of different characteristics such documents that pose challenges for information extraction.

**Schema complexity** Different types of documents are produced in service engagements, each providing information on different schema entities. Each document type is rich in terms of entities and relationships, making the overall schema (i.e., ontology) in service domains extremely large and complex. Thus, ontology-based information extraction approaches (e.g., [5]), which require an exhaustive listing of extraction patterns for all elements in the ontology, become cumbersome to implement.

**Structural heterogeneity** Many projects create templates for deliverable types to guide content authoring. We consider documents of the same type and from the same project as the extraction sets. However, consultants routinely rename/re-order sections and creatively format content in free-form tools, such as MS-Word, introducing great structural variations even within extraction sets. Currently, data extractors can learn implicit templates only from highly regular input data sets, such as templated web pages [19, 21] and scientific simulator reports [1]. Our experiments with Road Runner [19] did not yield any results. Their approach of formulating fields and tags by studying mismatches across the data set does not work in presence of differences (due to structural noise) that do not necessarily indicate presence of any field/tag.

**Small extraction sets** Size of extraction sets in the services context are mostly limited to a few dozen. Hence, supervised learning based information-extraction approaches
[8, 9] are not attractive because one cannot amortize the initial cost of training over thousands of documents, as is the case in web-data extraction.

**Diverse extraction units** Enterprise information extractors need to handle varying lengths of extraction units and deal with diverse formatting. Figure 1 highlights some extraction unit patterns in enterprise documents. On the one hand, there may be long paragraphs of data with embedded images and tables that can be tagged to a single attribute in the schema. On the other hand, small strings of words inside list items or table cells may also exist as possible extraction units. We find web-data extractors are often customized to handle certain kinds of extraction units better than others, which proves to be a limitation.

**Complex relational patterns** Enterprise documents often contain relational data represented in complex table structures. State-of-the-art techniques [6, 20] can only deal with tables that follow common patterns such as: (1) two-column tables that list a single value per attribute (The name of an attribute is extracted from the first column and its value from the second column); (2) relational tables\(^1\) with headers on the first row only. AHA extends the extraction of single-valued attributes to work for tables with any number of columns. However, recognizing multiple values per attribute in case of relational tables that have merged cells or nested tables remains a challenge.

### 4 The Asset Harvester Assistant

The Asset Harvester Assistant (AHA) provides a semi-automatic method for extracting information from a collection of MS-Word documents. Before describing AHA, we formalize the problem of information extraction.

A *target schema* is a 2-tuple, \(\Sigma = (E, R)\), where \(E\) is the set of entities and \(R\) is the set of relationships between entities. \(E = \{ e : e = (en, A); en\text{ is entity name}, A\text{ is a list of attributes}\}\).

A *model* is hierarchy of objects, each of which are instances of entities in the target schema. We use the term *model objects* synonymously with *record* in database terminology. Formally, a model \(m\) for a document \(d\) is a 2-tuple \((O, s)\); where \(O\) is a set of model objects and \(s \in O\) is the root object. Further, every model object comprises of: (a) entity name, (b) a set of attribute-values, (c) a set of child objects. A model object, \(o = (en, AV, \Omega)\); where \((en, A) \in E\), \(AV = \{ \alpha : \alpha = (a, v); a \in A; v\text{ is a content snippet in } d\}\), \(\Omega = \{ o_child : o_child \in O, (o.en, o_child.en) \in R\}\).

**Problem Formulation** Given an extraction set of documents \(\delta\), and a target schema \(\Sigma\), find models \(m_i\), corresponding to each document \(d_i \in \delta\) that follow \(\Sigma\).

\(^1\)A relational table is one that lists names of semantic entities in a row and values for those entities in subsequent rows, such that all cells in every column share common semantics.

The AHA approach consists of four main steps.

1. **Semantic Structure Extraction.** First, the tool automatically infers a semantic structure that represents a template implicit in the extraction set. A *semantic structure* consists of all potential semantic entities in the document set. Each semantic entity contains links to all extraction units that share the semantics.

2. **Bulk Tagging.** Next, the user chooses tags (coming from the target schema) to annotate each extraction unit of interest in the semantic structure.

3. **Model Extraction.** Then, AHA applies the tags to create model objects from the extraction units and establishes relations between model objects to derive a model for every document. This model is serialized as an XML file.

4. **Model Refinement.** Finally, each model may be verified and refined within the AHA Model Refiner. The XML model can then be regenerated.

Next, we describe each of the four steps.

### 4.1 Semantic Structure Extraction

Sections, tables, and lists are commonly used to organize related content in formatted documents. For example, the cells in a column of a relational table\(^1\), or the items in a bulleted or numbered list, share the same semantics. Headings are often used to segment content in a document. A *heading* or *header* is usually a short piece of text that indicates the semantics of the content that follows it. AHA resolves such structural cues and identifies repeating headings across the extraction set \(\delta\) in order to discover its *semantic structure* \(\Psi\) (See Figure 2).

AHA parses XML representations\(^2\) of MS-Word documents for information extraction. First, we identify headers that recur at similar positions in many documents across the extraction set \(\delta\) (lines 7–20 in Figure 2). For this purpose, we filter header candidates \(l_i\) by a threshold on their length (line 10), index them by their position in the XML document \(l_i, \chi\) (line 11), and then apply edit distances on both text and position to find variations of the same header (line 13). We denote the set of frequent headers as \(\kappa\).

We hypothesize that the outline created by headers in \(\kappa\) can closely match a template implicit in \(\delta\) and each frequent header indicates a potential semantic entity. For every document \(d_i\), we create a document tree \(t_i\) that organizes the headers in a hierarchy in which they appear in \(d_i\) (lines 21–38). Every node \(nd\) in \(t_i\) is described by its (1) *header*: associated header; (2) *bookmark*: a bookmark created to cover the content segment that follows the header; (3) *type*: formatting used in the bookmarked content such as “section,” “table,” “col,” etc.; (4) *children*: nodes that have

\(^2\)Word processors export content in XML formats such as ODF, WordML, OpenXML, etc.
begin function extractSemanticStructure(Document set δ)
  01. k ← identifyHeaders(δ); // Derives list of candidate headers
  02. Semantic structure, Ψ ← null
  03. for each document d, ∈ δ do
  04.   l ← getDocumentTree(d, k)
  05.   Merge l, into Ψ to get Ψ = Ψ. TrieEditDistance(Ψ, Ψ') is minimum
  06. return Ψ
end function extractSemanticStructure

begin function identifyHeaders(Document set δ)
  07. headerDictionary ← null
  08. for each document d, ∈ δ do
  09.   for each line l, ∈ Lines(d) do
  10.     if Words(l, ) |c l then
  11.       // = max. number of words allowed in a header
  12.       l, header ← XPath expression from document root in XML(d, )
  13.       l, τ ← Text(l, )
  14.       if l, count ∈ l, count + 1
  15.       else l, count ← 0; Add l, to headerDictionary
  16.     headerList ← null Initialize list of headers
  17.   for each line l, ∈ headerDictionary do
  18.     if l, count > δ |l| then
  19.       if Γ ∈ number in (0,1) used as threshold to choose headers
  20.       return headerList
end function identifyHeaders

begin function getDocumentTree(Document d, Headers k)
  22. Set root. bookmark covering the entire document, cur ← root
  23. Traverse the XML DOM for d:
  24.     for each section σ traversed st. title(σ) ∈ k do
  25.       Create node sn. sn. type ← “section”; Set sn. bookmark to cover σ
  26.       sn. header ← title(σ); Add sn to cur.children; cur ← nd; for each table τ s.t. τ contains at least one nd ∈ k
  27.       Create a node tbl for τ s.t. tbl. type ← “table”
  28.       Set tbl. bookmark to cover τ
  29.     Apply structural heuristics to identify τ as relational
  30.     or list of attribute-values
  31.       if τ is relational then
  32.         Bookmark each column γ in τ to create a node cγ. cγ. type ← “col”
  33.         cγ. header ← Header(γ); Add cγ to tbl.children
  34.       else Create node αi, for each range starting at ki, & ending before ki+1
  35.       αi. type ← “cont”/“all”; Set αi. bookmark to the range (ki, ki+1)
  36.       αi. header ← ki; Add αi to tbl.children
  37.       Add tbl to cur.children
  38.     for each segment c bounded by ki, ki+1 ∈ k
  39.       Create new node nd
  40.       if c is a bulleted or numbered list then nd. type ← “list”
  41. else nd. type ← “rich text”
  42. //A content can span various formatting elements
  43.   Set nd. bookmark to cover c; nd. header ← ki;
  44.   Add nd to cur.children
  45. return root
end function getDocumentTree

Figure 2. Semantic structure extraction

their bookmarked content segments contained within the range of nd. bookmark. For instance, the nodes created for a table column exist as children of the table’s node. Figure 3(a) shows an example of a semantic node extracted from a document.

The next step is to merge the document trees for all documents in δ into a semantic structure, Ψ, using tree-alignment techniques [21]. Starting from a seed, we keep enriching Ψ as we visit new document trees. For each document tree t, we merge all its nodes into Ψ to obtain Ψ that the tree edit distance [4] between Ψ and Ψ is minimal. After consolidation is complete, each node nd in Ψ holds bookmarks to all content instances in δ that share the semantics indicated by nd. header. Thus, if a user assigns a tag against nd, all the bookmarked content instances get bulk-tagged at once. Note that AHA is able to cope with structural heterogeneity in documents because it leverages similarity in headers to derive indications of semantic similarity instead of solely relying on document structure. Web-data extraction techniques that depend on determination of frequent HTML tag patterns do not work well with dense and irregular formatting. AHA’s document-tree abstraction retains the essential document structure, while eliminating majority of formatting. Thus, fluctuations in formatting styles have minimal impact on AHA’s performance.

4.2 Bulk Tagging

The semantic structure Ψ created for the document set is serialized into an MS-Excel spreadsheet, which acts as AHA’s bulk-tagging interface. The tree hierarchy is represented using Excel’s grouping feature: one can show/collapse the rows to navigate the semantic structure in a top-down manner. By clicking on the bookmarks listed against a node, a user can open up the MS-Word documents and navigate directly to the content segments that are grouped under that node. Such a feature enables a user to comprehend the semantics of extracted information segments and choose appropriate tags. AHA requires a user to specify two levels of tags against nodes in Ψ—an entity tag and an attribute tag. AHA allows the user to tag at multiple levels of granularity. For example, the user can either tag the entire contents of a “section” or tag at the “list” level and direct AHA to create multiple instances of a chosen entity—one for each item in the list. Figure 3(b) illustrates the Excel-based interface for bulk tagging.

4.3 Model Extraction

For each document d, AHA extracts a hierarchical model instance ni = (O, s), which obeys the target schema Σ = (E, R). The root model object s is an instance of a given root entity ρ ∈ E. We traverse the semantic structure Ψ in a depth-first manner to fetch tagged nodes that have bookmarks covering content in di. We create model objects corresponding to the tags specified against such nodes, and add the bookmarked content segments as values of attributes in them. AHA retrieves the content as either plain text or formatted HTML depending upon the type of the attribute as defined in Σ. If there are siblings in Ψ that are assigned the same entity tag, the corresponding model objects are merged. Again, we extract multiple model objects from a single extraction unit if the tagged content is a “list” or a “table” and the schema allows support for such multiplicity. Finally, AHA establishes relationships between a pair of model objects if their corresponding tagged nodes are contained within one another in Ψ and the relationship is valid according to the target schema.
4.4 Model Refinement

The models extracted can be further refined by users using AHA’s Model Refiner (Figure 3(c)). The model can be viewed in a side panel within MS-Word. When a user clicks on a node in the model, the corresponding tagged content segment gets selected within the document. The user can choose to edit a model instance by adding/deleting model objects, object attributes, and linkages between model objects. The user can select any range in the document and tag its contents as the value for an attribute. The editor also lets the user copy a reference to a node and paste it under some other node that allows a link to it.

The model editor is useful when certain units in the semantic structure are not extracted at appropriate granular levels, or when there are missing linkages between the extracted model objects. For instance, because the atomic unit of extraction is a line, all semantic entities contained in a fragment of a line need to be tagged manually using the model editor. Again, users can infer linkages between the extracted model objects by understanding the semantic cues embedded in text, which are not interpreted automatically. For example, a requirement description may read as “...the requirement is met by Step 1 in Table 3.” In this case, the user has to manually link the requirement object to the step object extracted from Table 3 using the Model Refiner.

5 Empirical Evaluation

AHA is currently deployed within the IBM service-delivery practice and is being used to harvest information from documents produced during business-transformation projects. We selected a sample of such documents and performed studies to evaluate (1) the effectiveness of AHA’s bulk-tagging feature to extract content, (2) the usefulness of the model-refinement feature to improve quality of the extracted content, and (3) the improvement in productivity that AHA can provide.

5.1 Experimental Setup and Method

We selected a total of 270 Process Definition Documents (PDD) from two service engagements; we refer to these documents as Data-Set1 (DS1) and Data-Set2 (DS2). Table 1 summarizes the data-sets and the distribution of different types of content in those sets. DS1 consists of 234 documents, with a total of 19093 extraction units; of these, 7.56% are rich text, 37.51% are lists, and 54.92% are plain text. Similarly, DS2 consists of 36 documents with 4203 extraction units, of which 18.88% are rich text, 35.21% are lists, and 45.91% are plain text. The data illustrate the diversity in the extraction units. A significant percentage of the extraction units are rich text, which supports our claim about the complexity of enterprise data.

Table 1. The data-sets used in the empirical evaluation.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Documents</th>
<th>Number of Extraction Units</th>
<th>Rich Text (%)</th>
<th>List (%)</th>
<th>Plain Text (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS1</td>
<td>234</td>
<td>19093</td>
<td>7.56</td>
<td>37.51</td>
<td>54.92</td>
</tr>
<tr>
<td>DS2</td>
<td>36</td>
<td>4203</td>
<td>18.88</td>
<td>35.21</td>
<td>45.91</td>
</tr>
</tbody>
</table>

DS1 is one of the largest extraction sets in an asset repository (within IBM), which contains over 14000 documents for a specific business-transformation domain. The domain-level schema has 140 entities, 338 attributes, and 263 relationships. Note that AHA did not adopt an ontology-based approach because such large schemas are common in service domains. However, the subset of the schema applicable to PDD documents, which constitute our experimental data-sets, contained 27 entities and 37 attributes.
To measure the effectiveness of bulk tagging and model refinement, we use two metrics. The document coverage (of a document) is the percentage of words in the document that are tagged by the techniques. The schema coverage (of a target schema) is the percentage of schema elements that are tagged by the techniques.

To collect the data, we used the following method. First, for both the data-sets, we used AHA’s bulk-tagging feature to extract and tag information. We wrote Visual Basic scripts to get word counts from the original and tagged documents to compute document coverage. Similarly, we wrote XSL-based scripts to get information from the generated XML models to compute schema coverage. Next, we used AHA’s model-refinement capability to improve the extracted models such that all content that could potentially be mapped onto the target schema was tagged. Once the manual refinement was completed, we recomputed the document and schema coverage again—the recomputed values indicate the best possible coverage for the data-sets.

For comparison, we collected data on the time taken, and the schema coverage attained, when practitioners used cut-and-paste to copy content from DS1 and DS2 onto a document template (designed according to the target schema).

5.2 Results and Analysis

Table 2 presents the coverage results. For each data set, the table contains two rows: the first row presents the results for bulk tagging, whereas the second row presents the results obtained after the manual-refinement step. Columns 2, 3, and 4 show, respectively, the lowest, highest, and average percentages of document coverage. Columns 5, 6 and 7 show the same data for schema coverage.

Using bulk tagging, on average, 92% of the document content for DS1 and 69.06% of the document content for DS2 were tagged. After manual refinement, the document coverage decreased by 1% for DS1 and increased by 1.42% for DS2. Therefore, the data illustrate that the manual refinement may cause the coverage to increase or decrease. We verified that the document content that remained untagged after manual refinement was indeed not important from the tagging perspective. The untagged content included text fragments, such as headings, footers, table of contents, version history, etc. The lowest value of percentage document coverage for DS2 increased significantly by 13%. We observed only two instances in DS2 where the document coverage increased by over 10%.

The average schema coverage, after bulk tagging, for the two data-sets was 41.71% and 44%; after manual refinement, the coverage increased to 44.05% and 47.52%, respectively. Thus, unlike document coverage, schema coverage did not decrease after manual refinement. The average increase was 8% for DS1 and 3.5% for DS2.

Table 3 presents data to illustrate the quality of the harvested content, measured as the percentage of extraction units identified and tagged. Of the 19093 extraction units for DS1, 89.34% were tagged during the bulk-tagging step. For DS2, 87% of the 4203 extraction units were tagged during bulk tagging. This indicates that bulk tagging is quite effective in identifying the extraction units and tagging them appropriately. The manual refinement step helped increase the quality further by identifying finer-grained tags. The average increase in number of extraction units after manual refinement was 11.93% for DS1 and 4.2% for DS2.

Table 4 presents the results for the time taken to harvest content with and without AHA. Column 2 shows the total time taken by the AHA bulk tagging, whereas column 3 shows the average time per document for manual refinement. Columns 4 and 6 list, respectively, the total times for AHA and the alternative manual approach. For DS1, the AHA bulk tagging took 50 minutes and manual refinement took an average of five minutes per document. The total time for harvesting content from 234 documents was 1220 minutes. Without AHA, the practitioners spent on average 40 minutes to cut-and-paste content to a simplistic document template that offered 20% schema coverage. For DS2, the total time required for content harvesting was 216.6 minutes with AHA and 1440 minutes without AHA. Thus, using AHA, efficiency improved by at least 6 times.

5.3 Discussion

Overall, the studies illustrate that the bulk-tagging step of AHA can be very effective: for our data-sets, this step successfully aggregated information and created
near-perfect models with 76.43% document coverage and 42.88% schema coverage; the step also identified 88% of the extraction units. The manual-refinement step of AHA is useful too: it lets the user refine the automatically created models and does not require too much effort (because the automatically created models are very accurate). On average, users spent only about five minutes per document to perform manual refinement. Together, these data demonstrate the effectiveness of our approach.

The study also provided interesting insights into the manual-refinement step. The most commonly used operation in this step was splitting a tagged region, marked as a single instance of an entity, into multiple instances. This is shown by the increase (13% for DS1 and 12% for DS2) in the number of list-entity instances after manual refinement. This occurs because a list of entities can be represented using a variety of formatting patterns; currently, our tool does not identify complex relational patterns, such as nested lists and tables with merged cells. For such (unrecognized) patterns, the tool extracted the contents as a single model object instead of creating multiple model objects. Other manual refinements included the tagging of short text fragments within a line, and the creation of links between model objects based on semantic cues hidden in the text. For many documents, manual refinement decreased the document coverage because white spaces and redundant heading texts were removed during refinement.

6 Conclusion and Future Work

AHA addresses many of the challenges in information extraction from enterprise documents created in service engagements. Existing ontology-based extraction approaches are not suitable because the schemas in the services domain are voluminous and evolving. AHA automatically learns an implicit template in document sets to jumpstart extraction. Our approach can handle diverse extraction units and recognize many relational patterns. AHA also provides an interactive user interface for manually refining the extracted models. AHA has been deployed in the IBM service-delivery practice for over four months now. Based on user feedback, we have enhanced AHA to improve the usability of the model-refinement feature, improve the formatting of the extracted rich-text units, and handle extraction of attachments of various file formats.

Potential directions for future work include improving the semantic structure extraction to deal with more complex relational patterns, adding support in the Model Refiner for detecting repetitive patterns, and extending model refinement to Word processors other than MS-Word.

References