Extracting Document Structure to Facilitate a Knowledge Base Creation for The UML Superstructure Specification*

Mehrdad Nojoumian
School of Information Technology and Engineering, University of Ottawa
mnojoumi@site.uottawa.ca

Timothy C. Lethbridge
School of Information Technology and Engineering, University of Ottawa
tcl@site.uottawa.ca

Abstract

The research presented in this paper aims at facilitating the creation of knowledge bases (KBs) for software specifications, of which the UML superstructure specification is our initial target. Our motivation is that such specifications are dense, repetitive and difficult to use. They are written primarily in semi-structured text, but the structure must be maintained manually as they are edited, resulting in inconsistency. End users cannot use them efficiently because of the duplications, numerous concepts connected only implicitly, and general complexity of the document. Our immediate objective is to generate a KB for the UML specification by extracting knowledge from as many sources as possible in the document such as document structure, embedded natural language, as well as implicit and explicit cross references. In this paper our focus is the first step: extraction of the document's logical structure. Many key concepts of a document are expressed in this structure, which includes the headings of the chapters, sections, subsections, etc. By extracting such a structure in XML format, we can form a good infrastructure for the subsequent KB creation steps.

Keywords: *Document analysis, Logical structure, Document conversion, Information extraction, Knowledge acquisition*

1. Introduction

Published electronic documents, such as specifications, are often rich in knowledge, but that knowledge is only partially structured. This makes it difficult for human beings to make maximum use of the documents, and it means automatic computer processing of the knowledge is not immediately possible.

Characteristics of the documents that cause challenges include the following:

- The conventions used to structure them, such as section numbering, style sheets, etc, are only partially specified and are not always properly adhered to.
- A lot of the knowledge is embedded in cross references. Some cross-references are explicit (e.g. hyperlinks, or explicit references to other numbered sections), but many are just implicit, such as simple use of a word defined elsewhere. Human beings quickly become confused by the network of references, and there are often undetected inconsistencies.
- Natural language is still prevalent in the details expressed in the document. The names of entities also contain embedded natural language.
- The documents themselves tend to be only publicly available online in either html or pdf formats. A lot of the markup in these formats is 'noise', including such things that are imposed by program which generated the html or pdf from some word processing file.

The overall objective of our work is to extract as much knowledge as possible from a published specification, to represent this knowledge formally in a knowledge base (KB), and then to create tools that can allow for easy exploration and editing of that knowledge. For example, our vision is that the person in charge of the specification could generate views of relevant relationships in order to ensure they are correct, and make changes where necessary. It would be possible to generate a specification in pdf format from the knowledge base, but an end user would be much better off exploring the knowledge base using the more flexible KB exploration tool.

This paper describes the very first step towards our vision: extracting the most useful knowledge from an electronic document.

In general, document processing can be divided into two phases: document analysis and document understanding. A document has several layers of structure. Extraction of the *geometric structure* (including

* This research is supported by the IBM Ottawa Software Lab

entitles such as pages, blocks, lines, and words) is referred to *document analysis.* Mapping this structure into a *logical structure* (including titles, headings, abstract, sections, subsections, footnotes, tables, lists, explicit cross-references, etc.) is referred to *document understanding* [2]. Extracting concepts embedded in the document structure, such as realizing that the names of some sections represent concept names, and the cross-references represent relationships among the concepts, is a form of *knowledge acquisition*.

From the structural point of view, a document can be *unstructured*, *semi-structured* or *structured*. A plain text document with nothing marked other than the normal conventions of natural language (e.g. a period at the end of a sentence) would be considered unstructured. A document with tags dividing it into paragraphs, headings, and sections would be considered semi-structured; most web pages are of this type. A document in which all the elements are marked with meta-tags, typically using XML, would be considered structured. A structured document can be represented as a tree, with leaf nodes representing very small snippets of textual content, such as the names of entities.

In practice, software specification documents fall somewhere on the continuum between semi-structured and structured. However, the markup is usually noisy.

When more structure is imposed on a document, the resulting richer representation allows computers to make use of the knowledge directly. Unstructured documents or sections have to rely on natural language understanding technology before the knowledge can be used.

To conclude, one of the major advantages of electronic documents is that we can partition them into a hierarchy of physical components, such as pages, columns, paragraphs, lines, words, tables, figures, etc or a hierarchy of logical components, such as titles, authors, affiliations, sections, subsection, etc. This structural information can be very useful in information extraction and knowledge acquisition, which are essential steps for KB creation.

This paper is organized as follows. Section 2 reviews the existing literature on the document analysis and knowledge extraction. Section 3 presents the properties of our targeted document. Section 4 focuses on divers document transformations. Section 5 illustrates three experimental results for the document's logical structure extraction. Finally in Section 6 we present future work and concluding remarks.

2. Literature Review

In this section, we review the document analysis and knowledge extraction literature in order to form a clear vision of these areas; we refer to many interesting ongoing research projects.

In [1], Mao et al propose numerous algorithms to analyze the physical layout and logical structure of document images in many different domains. The authors provide a detailed survey of diverse algorithms in the following three aspects: physical layout representation, logical structure representation, and performance evaluation.

S. Klink et al. [2] present a hybrid and comprehensive approach to document structure analysis. Their approach is hybrid in the sense that it makes use of layout (geometrical) as well as textual features (logical) of a given document.

J. Liang [3] presents a unified document structure extraction algorithm that is probability-based for scanned document image pages. He also developed a system that detects and recognizes special symbols (Greek letters, mathematical symbols, etc.) on technical document pages that are not handled by the current OCR (Optical Character Recognition) systems.

In [4], Nakagawa et al. proposes a mathematical knowledge browser which helps people to read mathematical documents. Using this browser, printed mathematical documents can be scanned and recognized by OCR. Then the meta-information (e.g. title, author) and the logical structure (e.g. section, theorem) of the documents are automatically extracted.

In respect to the document analysis tools, WISDOM++ [5] is a document processing system that operates in five steps: document analysis, document classification, document understanding, text recognition with optical character recognition (OCR), and text transformation into HTML/XML format.

In [6], Cohen and Jensen assume that structured documents are represented with the document object model. Their approach to information extraction is based on a DOM tree, which is an ordered tree where each node is either an element or a text node. An element node has an ordered list of zero or more child nodes, and contains a tag (such as "table", "h1", or "li") and attributes (such as "href" or "src"). A text node is normally defined to contain a single text string.

The next phase after the document analysis is knowledge extraction which is important from both general and specific points of view.

In [7], Henzinger and Lawrence discuss methods for extracting knowledge from the web by randomly sampling and analyzing hosts and pages, and by analyzing the link structure of the web. By this approach, a variety of interesting information can be extracted, such as the distribution of interest in different areas, the nature of competition in different categories of sites, and the degree of communications among different countries.

IKRAFT [8] is an interactive tool to elicit from users the rationale for choices and decisions as they analyze information used in building a knowledge base. Starting

from raw information sources, most of them originating on the Web, users are able to specify connections between selected portions of those sources.

H. Sakamoto et al. [9] show their recent results in knowledge discovery from semi-structured texts which contain heterogeneous structures represented by labeled trees. The aim of their study is to extract useful information from documents on the Web.

In [10], Crowder and Sim's goal is to capture relevant knowledge from legacy documents. Firstly, they converted the legacy documents to XML documents where the output is semantically tagged. Once in an XML form, the data can be easily transformed. They describe the development of tools to automate the process of converting legacy documents to XML documents. They also show that XML versions of legacy documents provide better results than a basic text search over the identical documents.

In the past decade, most work on extraction has been focused primarily on factual information. Only recent years have witnessed a growing interest in subjective texts such as evaluative ones. The general problem that Carenini et al consider in [11] is how to effectively extract useful information from large corpora of evaluative text.

W. R. Cyre [12] developed a tool for knowledge extraction. The process is to begin with a basic ontology and extract Conceptual Graphs from text in the domain of interest. During this process, the ontology is augmented by the knowledge engineer. In this approach, the user scans the text and creates conceptual graphs from sentences or other expressions, and joins the individual graphs into a knowledge-base.

The approach presented by Vargas-Vera et al [13] describes a Semantic Annotation Tool for extraction of knowledge structures from web pages through the use of simple user-defined knowledge extraction patterns.

3. Document properties

The document we targeted, the UML superstructure specification (version 2.1), is a large specification in pdf format with 771 pages. It has almost 2200 headings with a lot of nested lists, hyperlinks, figures, tables, etc.

What was our motivation for analyzing pdf documents? First of all, people do not have access to the original word-processor formats of the documents much of the time. When documents are published to the web, an explicit choice is usually made to render the result as pdf or html to guarantee that everyone can read it (without having to have Microsoft Word, Framemaker, etc.) and so that people can not so easily create a new version of the document that appears to be an official version. Moreover, pdf format has some useful features that make it semi-structured; for example it often contains "bookmarks" created from headings to enable a user to navigate a document. However, a computer can also easily use this information to extract the structure.

Figure 1, shows sample bookmarks of the UML specification. The general structure of this document consists of parts, chapters, sections, subsections and keyword-headed sub-subsections. The names of some of these correspond to concepts such as 'Abstraction' and 'Associations'.

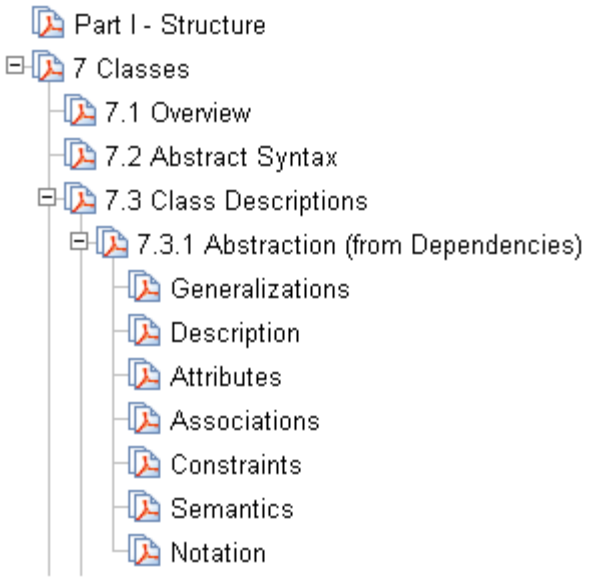

Figure 1. Bookmarks of the UML superstructure

Our ultimate goal is to extract the document's logical structure. As we mentioned, many key concepts of the targeted specification are expressed in this structure. By extracting the structure and representing it as XML, we can form a good infrastructure for the subsequent KB creation steps.

4. Document transformation

We approached the structure extraction problem as a two-stage problem. In this section we describe the first step: Transforming the raw input into a format more amenable to analysis. The second step, extracting and refining the structure, is the topic of Section 5.

To extract the logical structure of the document, we experimented with transformations using various existing tools to see to what extent each could facilitate the extraction process.

Since our targeted document is a large specification, we started with much smaller documents. Firstly we performed various conversions using a simplified sample file which had similar properties to the target document. Then we analyzed a single chapter, Chapter 7, before

moving on to process the first 219 pages, which covered the first 9 chapters.

4.1. Conversion experiments

Table 1 shows the tools we used for conversion and the formats we experimented with. We applied different tools in this respect such as Adobe Acrobat Professional 7.8, Microsoft Word 2003, Stylus Studio 2006 XML Enterprise Suite, and ABBYY PDF Transformer 1.0. In Table 1 we exclude transformations with similar results.

Table 1. Different conversions

Input Format (Size KB)	Tools for Conversions	Output Format (Size KB)
DOC (34.5)	Microsoft Office Word 2003	**TXT** (2.81)
DOC (34.5)	Microsoft Office Word 2003	**RTF** (55)
DOC (34.5)	Microsoft Office Word 2003	**HTML** (40.7)
DOC (34.5)	Microsoft Office Word 2003	**XML** (55)
DOC (34.5)	Adobe Acrobat Professional 7.8	**PDF** (19) with Bookmarks
DOC (34.5)	Adobe Acrobat Professional 7.8	**PDF** (15.9) without Bookmarks
PDF (19) with Bookmarks	Adobe Acrobat Professional 7.8	**HTML** (6.38)
PDF (15.9) without Bookmarks	Adobe Acrobat Professional 7.8	**HTML** (5.15)
PDF (19) with Bookmarks	Adobe Acrobat Professional 7.8	**XML** (9.92)
PDF (15.9) without Bookmarks	Adobe Acrobat Professional 7.8	**XML** (8.30)
PDF (19) with Bookmarks	ABBYY PDF Transformer 1.0	**HTML** (19.2)
PDF (19) with Bookmarks	ABBYY PDF Transformer 1.0	**TXT** (2.82)

In the next section, we define our criteria for choosing the best transformation; subsequently we evaluate these conversions according to these criteria.

4.2. Criteria

Since the ultimate goal of the paper is to extract the document's logical structure and convert it to XML we are most interested in an output format from the Table 1 which can most facilitate this.

To select the best conversion, we defined a set of criteria based on the experiences we gained during our experiments. These criteria are as follows:

(a) Generality: A format should enable the design of a general extraction algorithm for processing other electronic documents.

(b) Low volume: We should avoid a format which contains of a lot of extra unneeded material that is not related to the document content. This includes information related to the font, style, and position of words, lists, paragraphs, etc.

(c) Clean & understandable: Even if a format results in small files, it still might not be adequate; it should also be clean and understandable. For instance, formats which cleanly mark constructs such as paragraphs with a single marker and use carriage returns judiciously are easier to work with than formats that don't do this. For instance, some formats marked constructs with multiple markers and were not even consistent about this.

(d) Similarity to XML: We prefer a format which has a similar structure to XML, such as XML itself or HTML, because our final goal is to extract the logical structure in the XML style.

(e) Having good Clues: A format should use markers which provide accurate and good clues for processing and finding the logical structure, such as meaningful keywords with respect to the headings: "LinkTarget", "DIV", "Sect", "Part", etc.

Sometimes, formats which contain a lot of extra information, such as font, size, style and position of each part of the document, are more valuable for processing while in some cases documents which are absolutely pure without any extra characters are useful. Hence, we would like to compromise among different kinds of formats to satisfy our criteria. In the next part, we evaluate the presented transformations to define the best candidate.

4.3 First stage of evaluation

To narrow down the list of possible transformations to use, we evaluated every transformation in Table 1 according to how they satisfy the above criteria. We performed all the presented conversions on the UML superstructure specification. Our observations are as follows:

DOC and RTF formats are messy. They even code figures among the contents of the document while some formats such as HTML or XML put all the figures in a separate folder in an image format. In addition, they store information related to the font, size, style, etc of each heading, paragraph, sentence and even words beside them. This information is not useful for us because they vary from document to document, contradicting the generality property and increasing the potential for noise during processing. On the other hand, if we extract

HTML or XML formats from DOC/RTF, the results also tend to have the same properties.

TXT format is very simple but does not give us any clues for processing and you may not even find the beginning of the chapters, headings, tables, etc. Therefore, it does not have a suitable structure for analyzing.

PDF is complex itself, but after a conversion into HTML or XML by Adobe Acrobat Professional 7.8, the result is very nice, especially in the case of PDF files which have bookmarks. They are clean, low sized, with tagging structure and useful clues for processing. They can even satisfy the generality property as we describe it later on.

Therefore, our finalist candidates are HTML and XML formats extracted by Adobe Acrobat professional 7.8 from the PDF file with bookmarks. In the next section, we compare these two options.

4.4 Second stage of evaluation

To further narrow our choice of transformation, we analyzed the following sample parts of our target document using the two finalist candidates. These cover an array of possible structures that appear repeatedly in the UML specification:

(1) Sample paragraphs
(2) Sample figures (e.g. figure 7.25)
(3) Sample tables (e.g. table 2.1)
(4) Complex tables which have phrases, figures and hyperlinks in their cells (e.g. table 12.1)
(5) Complex nested lists which have complicated hierarchy structures (e.g. part 2.3)

After many assessments, we found out that the XML format is the best candidate for processing; it is more understandable and simple for analysis. Moreover, in the XML style, each tag is in a line, so we can analyze the document line by line which is easier in compare to the HTML format in which we have to explore the document character by character.

5. Extracting and refining the document structure

After following the steps described in Section 4, we have an XML document that is reasonably clean. However aspects of the document structure still need to be extracted. That is the topic of this section.

We discuss three implementation approaches to finalizing our extraction of structure. We evaluate our methods and the reasons of failures in the first two techniques. After that, we present our successful practice for the logical structure extraction.

5.1. First refinement approach: Grammars

In the first approach, we applied various parsing packages. We attempted to write a comprehensive grammar to parse the XML document; in particular, we needed to parse the internal structure of some of the tag-delimited data, such as the text of headings. The following are some examples of heading text that needed parsing:

7 Classes
7.1 Overview
7.2 Abstract Syntax
7.3.1 Abstraction

Although it should be fairly easy to write a parser to identify such elements as the section numbers and section titles, we encountered too many exceptions, resulting in the need for far too many rules and context-sensitive parsing.

5.2. Second refinement approach: Simple stack-based parsing written in Java

In this approach, we turned to writing simple java scanning code to scan for matching major tags, such as <Part>, <Sect> and <Div>, which Adobe Acrobat Professional 7.8 used to open and close each part, chapter, section, etc of the document. Consider the following simple structure of the document:

```
<Sect name="Generalization">
        <Sect name="Class-Ref">
                <Sect name="Name">
                </Sect>
                <Sect name="Package-Ref">
                </Sect>
        </Sect>
</Sect>
```

Using a straightforward stack-based parsing approach, we converted this into:

```
<Generalization>
        <Class-Ref>
                <Name>
                </Name>
                <Package-Ref>
                </Package-Ref>
        </Class-Ref>
</Generalization>
```

Unfortunately, after running the program for the different chapters and the whole document as well, it failed. We found out, there is a considerable amount of incorrect tagging. The tool opened each part, chapter, section, etc by "<Sect>" in a proper place of the document but it closed all of these tags by "</Sect>" in the wrong places. The problem was more crucial when we processed the whole document at once because of the accumulative mis-tagging. Here, a sample of this detection is presented:

<Sect number=" 7.3">
<Sect number="7.3.1">
</Sect>
<Sect number="7.3.2">
</Sect>
→ Correct place for closing <Sect number="7.3">
<Sect number="7.4">
</Sect>
</Sect>→ Wrong place

Therefore, we could not extract the logical structure by this simple approach and decided to develop a new program which is more powerful and capable of detecting such a wrong tagging. In the next part, our successful practice with corresponding results is provided.

5.3. Third implementation approach: leveraging the bookmarks

In the third approach, we wrote a java-based parser which focused on a keyword, "LinkTarget", which corresponds to the bookmark elements created in the previous transformation phase. This keyword is attached to each heading in the bookmark. Therefore, as a first step, we extracted all the lines containing the named keyword and put them in a queue: "LinkTargetQueue". We also defined the different type of headings in our document; you can see this classification in Table 2.

Table 2. Different kinds of headings

T	Sample Heading	Type
1	Part I - Structure	*Part*
2	7 Classes	*Chapter*
3	7.3 Class Descriptions	*Section*
4	7.3.1 Abstraction	*Subsection*
5	Generalization, Notation, etc	*Keyword*
6	Annex	*End part*
7	Index	*Last Part*

Then, we applied the following algorithm which takes the "LinkTargetQueue" as its input; each node of this queue is a line of the input XML file which has "LinkTarget" substring as a keyword.

Procedure DocumentStructureAnalysis(LinkTargetQueue)
F // a new XML file
L // a line: e.g.: <P id="LinkTarget_111914">7 Classes </P>
H // Heading: e.g.: 7 Classes
T // Type: e.g.: for the Chapters, $T_{Chapter} = 2$
$T_{\text{of the last member of the HeadingStack}} = 0$
HeadingStack = empty

While (LinkTargetQueue != empty) **do**
Get "L" from the LinkTargetQueue
Extract the heading "H" from the "L"
Define heading's type: "T"
While (T =< $T_{\text{of the last member of the HeadingStack}}$) **do**
Pop "H" and "T" from the HeadingStack
Close the suitable tag w.r.t the popped "T"
If (HeadingStack == empty)
Break this while loop
End if
End while
Push the new "H" and "T" in the HeadingStack
Open new tags w.r.t the pushed "H" & "T"
End while
While (HeadingStack != empty) **do**
Pop "H" and "T" from the HeadingStack
Close the suitable tag w.r.t the popped "T"
End while
Return "F"
End procedure

We extracted 2191 headings from the UML superstructure specification (version 2.1) and created a new XML file, Figure 2, for this document. We also tested the other documents and specifications such as UML Infrastructure (version 2.0); the extractions were well in all cases with 100% accuracy.

To trace the proposed algorithm, assume the following headings in the "LinkTargetQueue":

1 Heading → Chapter

2 Heading

2.1 Heading → Section

2.2 Heading

2.2.1 Heading → Subsection

2.2.2 Heading

2.3 Heading

3 Heading

$T_{Chapter}$ =2, $T_{Section}$ = 3, $T_{Subsection}$ = 4

The result would be as follow:

```
<Chapter number="1"
</Chapter>
<Chapter number="2">
        <Section number="2.1">
        </Section>
        <Section number="2.2">
                <Subsection number="2.2.1">
                </Subsection>
                <Subsection number="2.2.2">
                </Subsection>
        </Section>
        <Section number="2.3">
        </Section>
</Chapter>
<Chapter number="3">
</Chapter>
```

```
- <Book>
    <name>UML Specification</name>
    <P>text</P>
  - <Part number="1">
      <name>Structure</name>
      <P>text</P>
    - <Chapter number="7">
        <name>Classes</name>
        <P>text</P>
      - <Section number="7.3">
          <name>Class Descriptions</name>
          <P>text</P>
        - <Subsection number="7.3.1">
            <name>Abstraction</name>
            <Ref>Dependencies</Ref>
            <P>text</P>
          - <Generalizations>
              <P>text</P>
            </Generalizations>
          + <Description>
          + <Attributes>
          + <Associations>
          + <Constraints>
          + <Semantics>
          + <Notation>
          </Subsection>
        </Section>
      </Chapter>
    </Part>
  </Book>
```

Figure 2. Sample logical structure in the XML format

After extracting such a logical structure and creating the new XML file, we imported our document into the Protégé, as the targeted knowledge base system, using the commands available in its XML tab. In Figure 3, the logical structure model of the document is presented using the Jambalaya feature of Protégé 3.2.

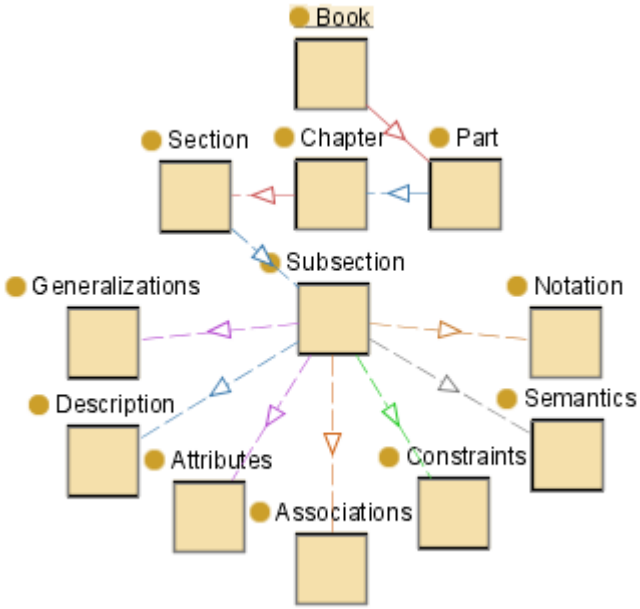

Figure 3. Logical structure model in the protégé 3.2

In the last section, we describe the second phase of the project as our future work and then provide some concluding remarks.

6. Future work and conclusion

In this paper we have described an approach to taking a raw pdf version of a published specification, and generating a clean XML document with meaningful tags.

The first phase was kind of *document analysis* to better understand the rational structure of the document and establish a good infrastructure for the second phase. We experimented with processing using a variety of tools and formats and concluded that generating pdf with bookmarks using Adobe Acrobat yields the best results.

We then applied different methods to form a new XML format which facilitated *document understanding*. This involved dealing with mis-tagging.

Now that we have extracted the document's logical structure (document entity), we intend to focus on the hidden concepts found in the remaining natural language elements, and consequently perform knowledge acquisition from the UML specifications.

As a future work, we are interested to know what knowledge could be captured from each element of the

structure. We will capture lists of all words, bi-grams, tri-grams and quad-grams with their frequency of occurrence. The most frequent of these, after excluding those that are simply stop words, will give us a sense of the terminology and concepts in the document as a whole and present a sense of the key topics in each chapter, section and subsection. We would like to do related-phrases analysis for relationships between the concepts identified in the terminological analysis. For example, patterns such as "X is a kind of Y", "X has a Y", etc.

To conclude, we have made a first step towards our goal of creating a tool that will make complex specifications more understandable and navigable.

7. Acknowledgement

We highly appreciate the anonymous reviewers for their comments.

8. References

[1] S. Mao, A. Rosenfeld, and T. Kanungo, "Document Structure Analysis Algorithms: A Literature Survey", *in Proceedings of SPIE Electronic Imaging*, USA, 2003, pp. 197–207.

[2] S. Klink, A. Dengel, and T. Kieninger, "Document structure analysis based on layout and textual Features", *in Proceedings of International Workshop on Document Analysis systems*, Brazil, 2000, pp. 99-111.

[3] J. Liang, "Document Structure Analysis and Performance Evaluation", *PhD thesis, University of Washington*, Seattle, USA, 1999.

[4] K.Nakagawa, A.Nomura, and M.Suzuki, "Extraction of Logical Structure from Articles in Mathematics", *3rd International Conference on Mathematical Knowledge Management*, Bialowieja, Poland, 2004, pp. 276-289.

[5] O. Altamura, F. Esposito and D. Malerba, "Transforming paper documents into XML format with WISDOM++", *International Journal on Document Analysis and Recognition*, vol. 4, 2001, pp. 2-17.

[6] W. Cohen and L. Jensen, "A structured wrapper induction system for extracting information from semi-structured documents", *17th International Joint Conference on Artificial Intelligence, Workshop on Adaptive Text Extraction and Mining*, Seattle, USA, 2001.

[7] M. Henzinger and S. Lawrence, "Extracting knowledge from the World Wide Web", *in Proceedings of the National Academy of Science*, USA, 101: 5186-5191, 2004.

[8] Y. Gil and V. Ratnakar, "IKRAFT: Interactive Knowledge Representation and Acquisition from Text", *in Proceedings of the 13th International Conference on Knowledge Engineering & Knowledge Management*, Siguenza, Spain, 2002, pp. 27-36.

[9] H. Sakamoto, H. Arimura, and S. Arikawa, "Knowledge Discovery from Semi-structured Texts", *Progress in Discovery Science, Springer Berlin*, Heidelberg, 2002, pp. 586-599.

[10] R. Crowder and Y. W. Sim, "An Approach to Extracting Knowledge from Legacy Documents", *In Proceedings of International Design Engineering Technical Conferences & Computers and Information Engineering Conference*, Salt Lake City, USA, 2004, ASME DETC2004-57677, 7 pp.

[11] G. Carenini, R. T. Ng, and E. Zwart, "Extracting knowledge from evaluative text", *in Proceedings of the 3rd International Conference on Knowledge Capture*, Banff, Canada, 2005, pp. 11-18.

[12] W. R. Cyre, "Knowledge Extractor: A Tool for Extracting Knowledge from Text", *in Proceedings of Fifth International Conference on Conceptual Structures (ICCS)*, Seattle, USA, 1997, pp. 607-610.

[13] M. Vargas-Vera, E. Motta, J. Domingue, S. Buckingham Shum, and M. Lanzoni, "Knowledge Extraction by using an Ontology-based Annotation Tool", *In Proceedings of the Knowledge Markup and Semantic Annotation Workshop*, Victoria, Canada, 2001, pp. 5-12.