DeTable - Wrapper für komplexe Tabellen
DeTable - wrapper for complex tables

Günther SOMMER
9625837
E175
TU Wien

DeTable - Wrapper für komplexe Tabellen
DeTable - wrapper for complex tables

Inhalt

Die Aufgabe dieser Arbeit (und des dabei entstandenen Programms) ist eine Umwandlung von komplexen HTML-Tabellen in ein XML Format. Diese Tabellen enthalten auch sogenannte „verbundene“ Zellen.

Diese Information soll aufgebrochen und die darin enthaltene Information in die aufgebrochenen Zellen gespeichert werden. Dazu ist es erforderlich, die ganze Datei einzulesen und den HTML-Sourcecode zu parsen, inklusive aller Texte und Attribute. Vor allem die Attribute COLSPAN und ROWSPAN der Tabellen-Tags (<TD> oder <TH>) sind von besonderer Bedeutung, da diese die Verbindung von Zellen für den HTML-Browser angeben.

Das Programm bricht die Zelle in die durch ROWSPAN und COLSPAN angegebene Anzahl von Zellen auf und klont die darin enthaltene Information. Desweiteren kann in Abhängigkeit der Header-Zellen (die entweder konfiguriert oder mittels <TH> Tag gekennzeichnet wurden) bei gewissen Spalten das Klonen unterbunden werden.

Besonders zu erwähnen ist, dass das Programm nicht nur mit Tabellen in Rechteckform umgehen kann (alle Zeilen haben die gleiche Anzahl von Zellen), sondern die Länge jeder Zeile unterschiedlich sein kann. Dies wird durch die Verwendung von verschachtelten dynamischen Listen erreicht.

Das Ergebnis des Programms kann wiederum in eine HTML-Datei, aber auch in eine XML-Datei (die DTD liegt bei) gespeichert werden.

Abstract

The main purpose of this paper (and the developed program) is a wrapper for complex html tables to a XML format. This tables also have so called „spanned“ cells.

The information in spanned cells should be disassembled and the information shall be stored in the disassembled cells. For this, it is necessary to read the html-file and to parse the full HTML-source code including all texts and attributes. The focus is on the special attributes COLSPAN and ROWSPAN of the table-tags (<TD> and <TH>), because they indicate spanned cells. The program breaks up the spanned cell into a amount of cells defined by COLSPAN and ROWSPAN and clones the information into this broken up cells.

Also in dependency of the header cells (configured via command line arguments or detected by the <TH> tag) the cloning can be suppressed for some cells.

A good feature of the program is, that not only rectangular tables can be used (every row has the same amount of cells), but also rows with different length can be parsed. This is done by use of nested dynamic lists.

The output of the program can be stored in a html or XML file (the defining DTD is also in the package).

Inhaltsverzeichnis

1.Problembeschreibung..... 4
1.1. Herausforderungen.....4
2.Einführung.....4
2.1. XML und HTML..... 4
2.2. Information Extraction.....5
2.3. HTML – Syntaxfehler.....5
2.4. Änderungen der Seitenstruktur..... 6
3.Existierende Lösungen..... 6
3.1. Lixto Visual Wrapper..... 6
3.2. XWRAP Elite..... 6
3.3. XFetch-Wrapper..... 7
3.4. Übersicht.....7
3.5. Beurteilung.....7
4.Die Lösung mit DeTable..... 8
4.1. Eingabedaten.....8
4.2. Parametrisierung..... 8
4.3. Tokenizer..... 8
4.4. Token Peeling..... 9
4.5. Parser..... 10
4.6. Analyse..... 14
4.7. HTML-Ausgabe.....15
4.8. XML-Ausgabe..... 15
5.Zusammenfassung und Ausblick.....17
6.Literaturverzeichnis.....17
7.Beispiel.....18
7.1. Bildschirmausgabe.....18
7.2. Input.....18
7.3. Ergebnis (HTML)..... 19
7.4. Ergebnis (XML).....19
8.Anhang 21
8.1. Konfigurationsparameter..... 21
8.2. Readme..... 21

1. Problembeschreibung

Die Aufgabenstellung war, aus einer HTML-Tabelle Information auszulesen, dabei die Struktur zu analysieren und diese zu verstehen, damit „verbunden“ Tabellen-Zellen in einzelne Zellen aufgebrochen werden können. Idealerweise sollten die so verarbeiteten Daten in einem XML-Format ausgegeben werden können.

Die Beispieldateien enthielten medizinische Therapievorschläge bzw. Medikamentationen. In dieser Arbeit wird eine solche Tabelle als Beispiel verwendet.

1.1. Herausforderungen

Um die Aufgabenstellung umzusetzen, mussten folgende Herausforderungen gelöst werden:

- Entwicklung eines Parsers für HTML-Tabellen mit Erkennung für Strukturinformation
- Erkennen von vertikalen und/oder horizontal verbundenen Zellen, deren Aufbrechen und das Klonen der darin enthaltenen Information
- Verwendung eines Single-Pass Verfahrens und damit einer dynamischen Tabellenstruktur
- Fehlertoleranz gegenüber vergessenen Tags
- Erkennung von Titel-Zellen

2. Einführung

2.1. XML und HTML

HTML und XML besitzen den gemeinsamen Stammvater SGML. Beide Sprachen verpacken daher ihre Strukturierungsinformation in sogenannten „Tags“.

HTML ist auch sehr stark auf die Formatierung und Präsentation des Inhaltes ausgerichtet, während XML keine impliziten Anzeige-Attribute und Tags enthält. XML ist auf strukturierte und hierarchische Abspeicherung von Information ausgerichtet und enthält keine direkten Formatierungsmöglichkeiten.

Daher kommen auch die verschiedenen Einsatzgebiete von HTML und XML. HTML wird zur Darstellung von Informationen im WWW verwendet, während XML zur Speicherung von Daten verwendet wird.

Oft werden die in HTML angezeigten Informationen aus XML-Datensätzen generiert. Dies ist eine einfach zu realisierende Lösung, da vom hierarchisch strukturierten in ein unstrukturiertes Format umgewandelt wird. Die Strukturinformation wird zur grafischen Präsentation benötigt, nur zum kleinen Teil zur Verarbeitung.

Obwohl dieser Weg einfach ist, ist die Umkehrung wesentlich problematischer. Die gleiche Information kann auf verschiedenste Wege dargestellt werden, gleich aussehen und trotzdem mit verschiedenen Technologien in HTML gelöst werden, wie z. B. Tabellen, Frames, CSS, etc. Für den Leser der Webseite ist kein Unterschied bestimmbar, da die verschiedenen Lösungen gleich dargestellt werden, für den Parser sind diese Lösungen aber sehr unterschiedlich und brauchen verschiedene Strategien zur Umwandlung.

Das Programm zur Umwandlung kann nicht die Darstellung am Bildschirm als Ausgangspunkt nehmen, sondern muss die Informationen aus dem Seitenquelltext in HTML entnehmen und daraus erkennen, welche Informationen zusammenhängen, ohne deren graphische Darstellung zu kennen.

2.2. Information Extraction

Der Prozess der „Information Extraction“ ist laut Peshkin und Pfeffer [4] das Ausfüllen von Vorlagen mit vorher unbekannten Text, der aber zu einem vordefinierten Gebiet zugehörig ist. In unserem Falle ist die „Vorlage“ eine XML-Datei mit bekannter DTD und der „unbekannte Text“ eine HTML-Datei mit einer Tabelle.

Für dieses Wrapping (in [5] dargestellt als Mediation) sind zwei Schritte lt. [6] notwendig:

- Extraktion der Daten, „Unwrapping“ – Aus dem HTML-Quelltext müssen die echten Informationen von den Darstellungsinformationen getrennt werden. Dies kann mit Hilfe von Strukturinformationen von HTML selbst durchgeführt werden, da z. B. Attribute der einzelnen Tags nur Darstellungsinformationen, aber nie Nutzdaten enthalten.

- Integration, „Wrapping“ – Die so gewonnenen Daten werden nun in ein internes Datenmodell (das vorgegeben sein kann, aber auch aus den Daten generiert wird) abgelegt, und es wird versucht die Daten zu normalisieren und redundante Daten zu eliminieren. Die im Datenmodell abgespeicherten Daten werden nun in strukturiertem Format in XML übertragen und abgespeichert. XML eignet sich hervorragend dafür, da es selbst keine Struktur vorgibt, aber dem Datenmodell entsprechend strukturiert werden kann (und auch generische Werkzeuge zur Überprüfung der Konsistenz und der Syntax erlaubt, die das Datenmodell nicht kennen müssen).

Die Daten nach ihrere Strukturiertheit unterschieden werden:

1. Strukturierte Daten: Das sind relationale Daten. Es gibt einen auswertbaren Zusammenhang zwischen den Daten und dem Aufbau der Datenbank (Beispiel: „Josef“ im Feld „Vorname“). Typisches Beispiel dafür sind relationale Datenbanken.
2. Teil-strukturierte Daten: Diese Daten sind nicht in einem strukturierten Datenformat, sie enthalten aber in Schlagwörtern die Informationen. Beispiele hierfür sind Job-Anzeigen oder Immobilen-Anzeigen in Zeitungen, aber auch gut strukturierte HTML-Seiten (z. B. Ebay).
3. Unstrukturierte Daten: Das ist normal geschriebener Text. Zur Extraktion muss der Inhalt des Textes verstanden werden. In diese Kategorie fallen z. B. Zeitschriftenartikel.

Dieses Projekt konzentriert sich vor allem auf die Extraktion aus Tabellen in HTML-Dateien, und diese fallen unter „Teil-strukturierte Daten“.

2.3. HTML – Syntaxfehler

Eine zusätzliche Schwierigkeit stellen vor allem im HTML Syntaxfehler dar. Moderne Browser sind darauf ausgelegt sehr viele Syntaxfehler zu ignorieren und ein möglichst gutes Ergebnis zu erreichen. Dies führt leider auch dazu, dass ein guter Teil der im Internet zu findenden WWW-Seiten fehlerhafte Syntax enthält. Vor allem Wrapper sind auf fehlerhafte Seiten empfindlich, da diese nach definierbaren Regelsätzen arbeiten müssen und nicht eine „best-effort“ Strategie verwenden können, wo der menschliche Leser die Zusammenhänge erkennen kann. Die häufigsten Fehler sind vergessene oder falsch geschachtelte Tags, fehlerhafte oder nicht ganz normkonforme Attribut-Beschreibungen. Ein Studie von 1995 [7] hat festgestellt, dass mehr als 40% der Webseiten nicht HTML-konform sind. Die meisten Fehler sind fehlende HTML, HEAD und BODY Tags, danach folgen aber schon falsch verschachtelte Tags oder fehlende End-Tags (beide jeweils ca. 30% der Seiten).

Diese Probleme können durch Wissen um den Aufbau von HTML gelöst werden, ähnlich wie es von den Browsern durchgeführt wird. Dabei werden logische Schlussfolgerungen gezogen, wie dass bei einem neuen Feld-Tag und dem davor fehlenden Schluss-Tag das Schluss-Tag als implizit gilt und damit intern das Start-Tag als Schluss-Tag und Start-Tag gehandhabt wird.

2.4. Änderungen der Seitenstruktur

Ein weiteres Problem beim Wrapping stellen Änderungen der Seitenstruktur da. Oft sind die HTML-Dateien von Webseiten im Laufe längerer Zeit nicht konstant. Es werden neue Elemente hinzugefügt, das grafische Layout geändert oder Umstrukturierungen vorgenommen.

Hartkodierte Wrapper [8] (diese sind sehr schnell implementierbar) verlassen sich auf einen festen Seitenaufbau mit gleichbleibender Abfolge von Elementen an fixen Positionen. Sie müssen also bei jeder Änderung angepasst werden. Hier sind Wrapper im Vorteil, welche die HTML-Struktur von Dokumenten analysieren können und die Ausgabe nur dann ändern, falls sich die Strukturierung der Nutzdaten geändert hat und nicht nur Layout-Änderungen vorgenommen wurden. Auch „unbekannte“ neue Dateien können für hartkodierte Wrapper möglicherweise unübersetzbar sein.

3. Existierende Lösungen

Es sollen hier nun einige HTML-XML Wrapping Tools dargestellt werden, diese verfolgen verschiedene Ansätze, um die Umwandlung durchzuführen.

3.1. Lixto Visual Wrapper[1]

Lixito [1] ist ein auf die Übernahme von dynamischen HTML-Seiten ausgerichtetes Werkzeug. Um die Definition des Regelsatzes für den Benutzer so einfach wie möglich zu machen, werden mit Hilfe eines visuellen Tools die für die Umsetzung interessanten Bereiche einer WWW-Seite (Tabellen, Frames, etc.) ausgewählt. Dadurch wird ein Regelsatz in der im Tool integrierten Programmiersprache Elog generiert. Der Wrapper kann das HTML-Dokument entweder als hierarchisches Dokument oder als flach durchsuchbares Textdokument behandeln. Die Verfeinerung der Regeln passiert dann in einem iterativen Prozess, der solange wiederholt wird, bis der Benutzer mit dem Ergebnis zufrieden ist.

Als Vorteil dürfte sich auf jeden Fall die kurze Einarbeitungszeit und auch die einfache Bedienbarkeit erweisen. Leider lässt sich zum Wrapping-Verfahren selbst nicht allzu viel in Erfahrung bringen, womit auch unklar ist, welche Stärken aber auch Schwächen dieses Tool besitzt.

3.2. XWRAP Elite[2]

XWRAP [2] ist ein Tool, das auf das Umsetzen von WWW-Seiten im Internet ausgerichtet wurde. Es kann eine URL der zu umwandelnden Seite angegeben werden, welches von XWRAP geladen wird. Danach wird die Seite heuristisch analysiert und es werden mehrere mögliche Varianten zur Auswahl gestellt. Wenn der Prozess abgeschlossen ist, generiert das Tool eine fertige Java-Klasse, die mittels Parameter eine gegebene Webseite mit dem zuvor generierten Regelsatz in XML umwandelt. Es können auch dynamische Teile für die URL definiert werden.

Vorteilhaft ist die einfache Benutzung und die direkte Umsetzung, sowie das Generieren der JAVA-Klasse, was eine einfache Einbindung in Projekte erlaubt. Nachteilig ist, dass nur via HTTP-Server publizierte HTML-Seiten verwendet werden können und dass keine manuelle Bearbeitungsmöglichkeit des Regelsatzes vorhanden ist.

1 http://www.lixto.com
2 http://www.cc.gatech.edu/projects/disl/XWRAPElite

3.3. XFetch-Wrapper[3]

XFetch-Wrapper (von Republica) dient dazu, aus irgendeinem Eingabe-Datenformat (also nicht nur HTML, sondern auch CSV, EDIFACT und andere Formate) eine XML-Ausgabe zu generieren. Dazu wird eine eigene Beschreibungssprache „DEL“ (Data Extraction Language) verwendet und die damit generierten Regeln zu einer Dokumentenart in einem Regelsatz gespeichert. DEL hat einen an XSLT angelehnten Aufbau und verwendet auch reguläre Ausdrücke.

Dieses Produkt dürfte sich sehr gut eignen, wenn nicht SGML-abstammende Sprachen in XML übersetzt werden sollen, und es kann mittels einer mächtigen und spezialisierten Beschreibungssprache verwendet werden. Nachteilig ist sicher, dass es keine Art von visuellem Umsetzungstool gibt, vor allem im HTML-Bereich.

3.4. Übersicht

Ein sehr gute Aufzählung von freien und kommerziellen Wrapping-Tools ist unter folgendem Link zu finden:

http://www.wifo.uni-mannheim.de/~kuhlins/wrappertools/

Desweiteren erläutert [3] einige Tools und kategorisiert diese auch nach ihrere Funktionsweise.

3.5. Beurteilung

Auf die vorliegende Problemstellung sind diese Programme nur bedingt anwendbar. Die Umwandlung von HTML in XML wäre durchführbar. Das Problem ist aber das zusätzliche Aufbrechen von verbundenen Tabellen, das einen Umbau des HTML-Quelltextes selbst erfordert, wofür diese Tools nicht entwickelt worden sind.

Daher musste ein Tool entwickelt werden, das die HTML-Tabelleninformation aus den bestehenden Dateien extrahiert und parsen kann, die verbundenen Zellen erkennt, sie als solche intern markiert und diese Information auch bei der Ausgabe in XML oder HTML verwertet, in dem zum Beispiel die enthalten Information vervielfacht werden.

3 http://www.x-fetch.com

4. Die Lösung mit DeTable

Nachfolgend wird nun das von mir entwickelte Programm sowie die Umsetzung der Problemstellung beschrieben. Es werden die einzelnen im Programm durchgeführten Schritte dargestellt und erklärt.

4.1. Eingabedaten

Um den Bau eines Wrappers zu erleichtern, wurden einzelne Beschränkungen der Eingabedaten vorgenommen.

Die Information selbst wird in Tabellen abgespeichert und kann aus den Tabellen ausgelesen werden. Die Tabellen können verbunden werden, dh. mittels HTML-Tags wird angegeben, dass der folgende Feldtext für eine Überzelle aus z. B. 3 mal 4 Zellen gilt.

4.2. Parametrisierung

Als erstes werden die Konfigurationsparameter eingelesen. Unabhängig von der Position werden die einzelnen Parameter abgearbeitet und die einzelnen Variablen der Klasse zugewiesen. Es kann auch ein Parameter für eine andere Konfigurationsdatei angeben werden (und diese könnte in ihren Parametern wieder eine weitere Konfigurationsdatei beinhalten). Die Parameter aus der Datei werden zur Parameterliste hinzugefügt. So lassen sich verschiedene Parameterkonfigurationen mit gemeinsamen und speziellen Teilen bilden.

4.3. Tokenizer

Aufgabe des Tokenizer ist es, die HTML-Datei in einzelne Token zu übersetzen. Als Token wird entweder ein Tag (Beispiel <BLABLA>) oder der Text zwischen zwei Tags gesehen (<...>BLABLABLA</...>). Dies ist notwendig für die weitere Verarbeitung und die Erkennung der Tabellen-Tags.

Die einzelnen Token werden in einer dynamischen Liste gespeichert.

Das ganze erfolgt in einem mehrstufigen Prozess, der in dem folgenden Flussdiagramm dargestellt wird.

Es wurden auch Vorkehrungen getroffen, um fehlerhafte HTML-Tags besser verarbeiten zu können.

Beispiel:

<TD>abc</TD><TD>def</TD> - korrekt

<TD>abc</TD<TD>def</TD> - fehlerhaft, nach dem ersten /TD fehlt das Zeichen „>“

<TD>abc</TD>TD>def</TD> - fehlerhaft, vor dem zweiten TD fehlt das Zeichen „<“

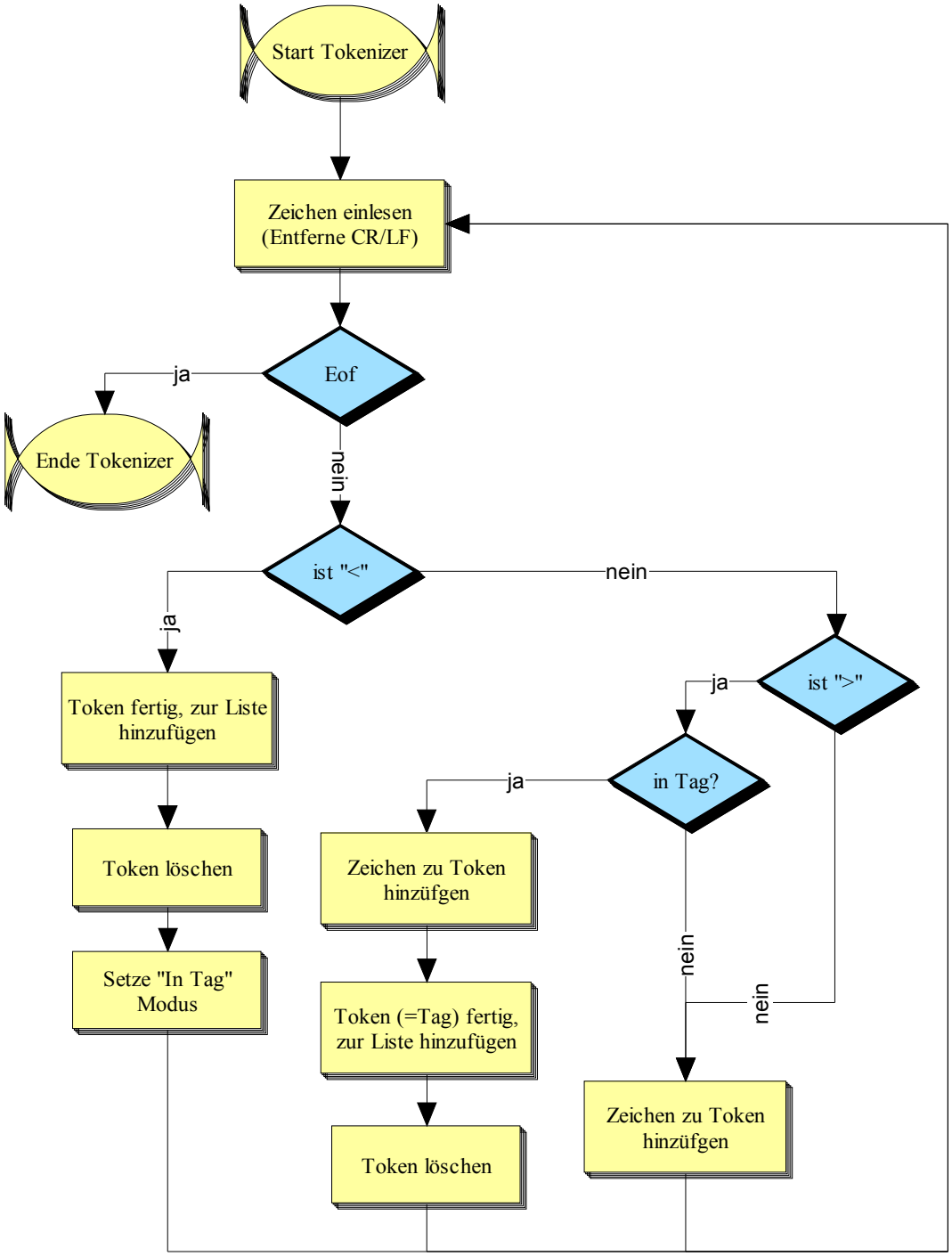
Da in HTML keine geschachtelten Tags in der Weise „<<TD>TD>“ zulässig sind, kann das obere Problem vom Tokenizer einfach gelöst werden.

Der Tokenizer liest die Datei Zeichen für Zeichen. Wenn der Tokenizer das Zeichen „<“ liest, dann sind alle Zeichen, die seit dem letzen „>“ oder „<“ gelesen wurden, ein neues Token.

Das zweite Problem wird anders gelöst. Der Tokenizer setzt ein Flag für den InTag-Modus wenn er ein „<“ gelesen hat. Mit dem Zeichen „>“ wird der InTag-Modus beendet. Ist der Tokenizer nicht im inTag-Modus, so wird das Zeichen ignoriert.

Der Ablauf wurde in Form eines Ablaufdiagrammes in Abb. 1 dargestellt.

Abb. 1 - Ablauf Tokenizer



4.4. Token Peeling

Diese Routine trifft noch einige Vorbereitungen, um den darauffolgenden Parser zu unterstützen. Insgesamt wird versucht, den HTML-Output dem HTML-Input so ähnlich wie möglich zu gestalten.

Mit der Voraussetzung, dass die Information nur in den Tabellen enthalten ist, werden die Teile vor und nach der Tabelle in separaten Listen gespeichert und nicht zur Umwandlung herangezogen. Das führt zu einer Vereinfachung des Parsen.

Die Routine schält den HTML-Teil vor dem TABLE-Tags der Token-Liste und nach dem </TABLE> in zwei andere Listen:

PreTable Liste – die Token vor der Tabelle

PostTable Liste – die Token nach der Tabelle

Die beiden neuen Listen werden vom Parser nicht verwendet und erst wieder im HTML-Output unverändert hinzugefügt.

Übrig bleibt nach dieser Routine nur mehr die Liste der Token innerhalb der Tabelle.

4.5. Parser

Der Parser ist das Herzstück der Applikation. Im folgenden wird der Ablauf erklärt, zusätzlich geben Abb. 2 und 3 den Ablauf als Ablaufdiagramm wieder.

Der Parser führt unter anderem auch eine Überprüfung auf einen weiteren (nested) <TABLE> Tag durch. Wird dieser erkannt, wird die Applikation abgebrochen.

Beim Durchlauf des Parsers ist die vertikale und horizontale Größe der Tabelle nicht bekannt. Der Ablauf wurde auf ein „1-Pass-Verfahren“ (nur ein Durchlauf des Parsers) Verfahren ausgelegt, um Ausführungszeit zu sparen. Daher verwaltet der Parser die Zellen in verschachtelten Listen.

Der Aufbau dieser Listen sieht folgendermaßen aus:

Tabellen-Liste – enthält x Zeilen-Listen

Zeilen-Liste – enthält a Zellen

Zelle

Zelle

Zeilen-Liste – enthält b Zellen

Zelle

Zelle

Zelle

Wie zu sehen ist, kann der Parser auch Zeilen mit verschiedener Anzahl von Zellen verarbeiten.

Der Parser führt intern mit, an welcher Position er sich in der neuen Tabelle befindet. Bei einer neuen Zelle oder Zeile werden die Zähler in X und Y-Richtung geändert und es wird überprüft, ob die entsprechenden Listenobjekte schon existieren, falls nicht, werden sie dynamisch angelegt.

Der HTML-Code einer Tabelle sieht ungefähr folgendermaßen aus:

<TABLE>

<TR>	***<TH>Spalte 1</TH>***	***<TH>Spalte 2</TH>***	***<TH>Spalte 3</TH>***	***</TR>***
<TR>	<TD>Zelle 1</TD>	<TD>Zelle 2</TD>	<TD>Zelle 3</TD>	</TR>
<TR>	<TD>Zelle 4</TD>	<TD>Zelle 5</TD>	<TD>Zelle 6</TD>	</TR>
<TR>	<TD>Zelle 7</TD>	<TD>Zelle 8</TD>	<TD>Zelle 9</TD>	</TR>

</TABLE>

Für die weitere Betrachtung ist wichtig, dass die einzelnen Zellentags (<TD> oder <TH>) noch weitere Attribute beinhalten können.

Vor allem zwei Attribute sind von Interesse

COLSPAN x – gibt an, dass eine Zelle mit x Zellen in horizontaler Richtung verbunden werden soll

ROWSPAN y – gibt an, dass eine Zelle mit y Zellen in vertikaler Richtung verbunden werden soll

Kombination – auch eine Kombination von ROWSPAN und COLSPAN ist zulässig, das ergibt dann einen zweidimensionalen Verbund

Diese geben dem Parser an, dass er diese Zelle aufbrechen soll, und die Zelle in die aufgebrochenen Zellen klonen soll. Die Anzahl der zu erstellenden Zellen wird mittels COLSPAN mal ROWSPAN berechnet. In den internen Datenstrukturen werden die Zellen gesetzt, als würde in den nächsten Zeilen an dieser Position Zellenwerte aus den Tokens gelesen werden. Dadurch verhält es sich für den Rest des Programmes transparent.

Intern wird, wie oben genannt, die aktuelle Position in der internen Tabelle gespeichert. Wird auf eine neue Zelle zugegriffen, wird vorher überprüft ob diese Zelle leer ist. Ist das nicht der Fall, so wird so weit nach rechts in der Zeile gegangen, bis eine leere Zelle kommt, oder die Zeile zu Ende ist (wodurch dann eine neue Zelle am Ende der Zeile erstellt wird).

Die obige Tabelle würde (mit einer verbundenen Zelle) in HTML dann so aussehen:

<TABLE>

<TR>	***<TH>Spalte 1</TH>***	***<TH>Spalte 2</TH>***	***<TH>Spalte 3</TH>***	***</TR>***
<TR>	<TD>Zelle 1</TD>	<TD>Zelle 2</TD>	<TD>Zelle 3</TD>	</TR>
<TR>	<TD>Zelle 4</TD>	<TD COLSPAN=2 ROWSPAN=2>		</TR>
<TR>	<TD>Zelle 7</TD>	Zelle 5 </TD>		</TR>

</TABLE>

Nachdem Durchlauf des Parsers sollte aus der oberen Tabelle mit den verbundenen Zellen folgende Tabelle mit den aufgebrochenen Zellen generiert werden:

<TABLE>

<TR>	***<TH>Spalte 1</TH>***	***<TH>Spalte 2</TH>***	***<TH>Spalte 3</TH>***	***</TR>***
<TR>	<TD>Zelle 1</TD>	<TD>Zelle 2</TD>	<TD>Zelle 3</TD>	</TR>
<TR>	<TD>Zelle 4</TD>	<TD>Zelle 5</TD>	<TD>Zelle 5 – Klon</TD>	</TR>
<TR>	<TD>Zelle 7</TD>	<TD>Zelle 5 - Klon</TD>	<TD>Zelle 5 - Klon</TD>	</TR>

</TABLE>

Die „Zelle 5 – Klon“ - Zellen werden noch nicht mit Information gefüllt, sie werden nur als Klon-Zellen markiert. Die Befüllung mit Daten wird erst im „Analyse“-Schritt durchgeführt, da dies von mehreren Parametern abhängig ist.

Wie schon anfangs erwähnt hat HTML-Code oft Fehler. Ein typischer Fehler ist auch das Vergessen von Schluss-Tags. Der Parser ist so gebaut, dass er mit diesen vergessenen Tags umgehen kann.

Die Tags <TD> und <TH> werden gleichwertig behandelt, sie haben für den Parser keinen Unterschied, ausser einen Vermerk im internen Datenmodell, der für die Analyse benötigt wird.

Im korrekten Beispiel sind zwei Zellen in der ersten Reihe zu sehen:

<TR><TD>1</TD><TD>2</TD></TR>

<TR>....</TR>

Problemfall 1 – es fehlt ein Schluss-Tag einer Zelle

<TR><TD>1<TD>2</TD></TR>

<TR>...</TR>

Der Parser setzt wenn er ein „<TD>“ liest den „inCell-Modus“. Mit „</TD>“ wird der inCell-Modus beendet.

Wird ein „<TD>“ gelesen, während sich der Parser im InCell-Modus befindet, so verhält sich der Parser, als hätte er ein „</TD>“ gelesen. Danach führt er den Schritt für „<TD>“ aus.

Problemfall 2 - es fehlt der Schluss-Tag der Zelle, Variante 2

<TR><TD>1</TD><TD>2</TR>

<TR>...</TR>

Der Parser setzt, wenn er ein „<TD>“ liest, den „inCell-Modus“. Mit „</TD>“ wird der inCell-Modus beendet.

Wird ein „</TR>“ gelesen, während sich der Parser im InCell-Modus befindet, so verhält sich der Parser, als hätte er ein „</TD>“ gelesen. Danach führt er den Schritt für „</TR>“ aus.

Problemfall 3 - es fehlt der Schluss-Tag der Zelle, Variante 3

<TR><TD>1</TD><TD>2

<TR>...</TR>

Der Parser setzt wenn er ein „<TD>“ liest den „inCell-Modus“. Mit „</TD>“ wird der inCell-Modus beendet.

Wird ein „<TR>“ gelesen, während sich der Parser im InCell-Modus befindet, so verhält sich der Parser, als hätte er ein „</TD>“ gelesen. Danach erkennt er (den impliziten InRow-Modus) und er führt den Schritt für „</TR>“ aus. Danach kommt erst der eigentliche „<TR>“-Schritt.

Problemfall 4 - es fehlt der Schluss-Tag der Zeile

<TR><TD>1</TD><TD>2</TD>

<TR>...</TR>

Der Parser setzt, wenn er ein „<TR>“ liest, den „inRow-Modus“. Mit „</TR>“ wird der inRow-Modus beendet.

Wird ein „<TR>“ gelesen, während sich der Parser im InCell-Modus befindet, so verhält sich der Parser, als hätte er ein „</TR>“ gelesen. Danach führt er den Schritt für „<TR>“ aus.

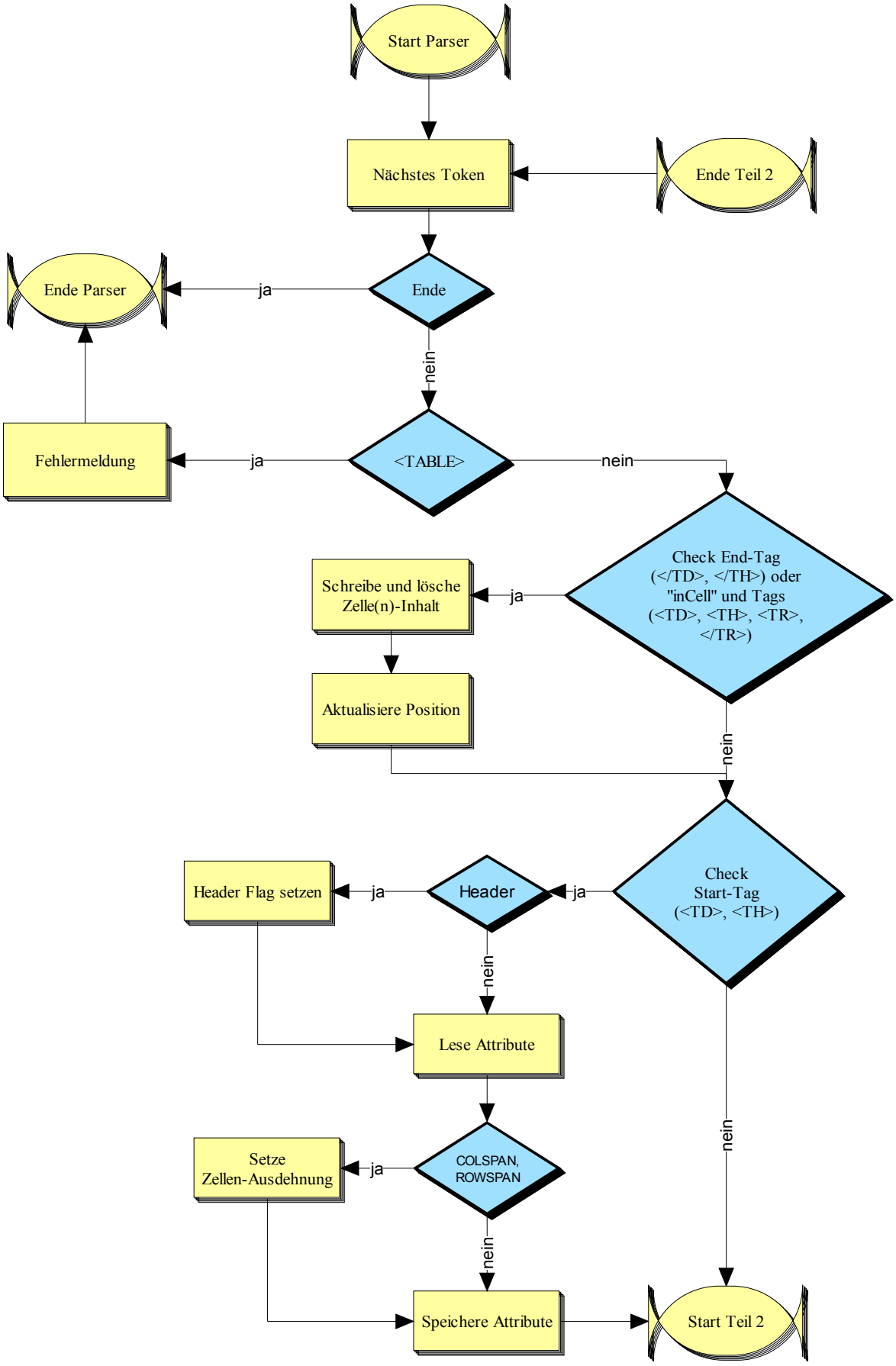


Abb. 2 – Ablauf Parser, Teil 1

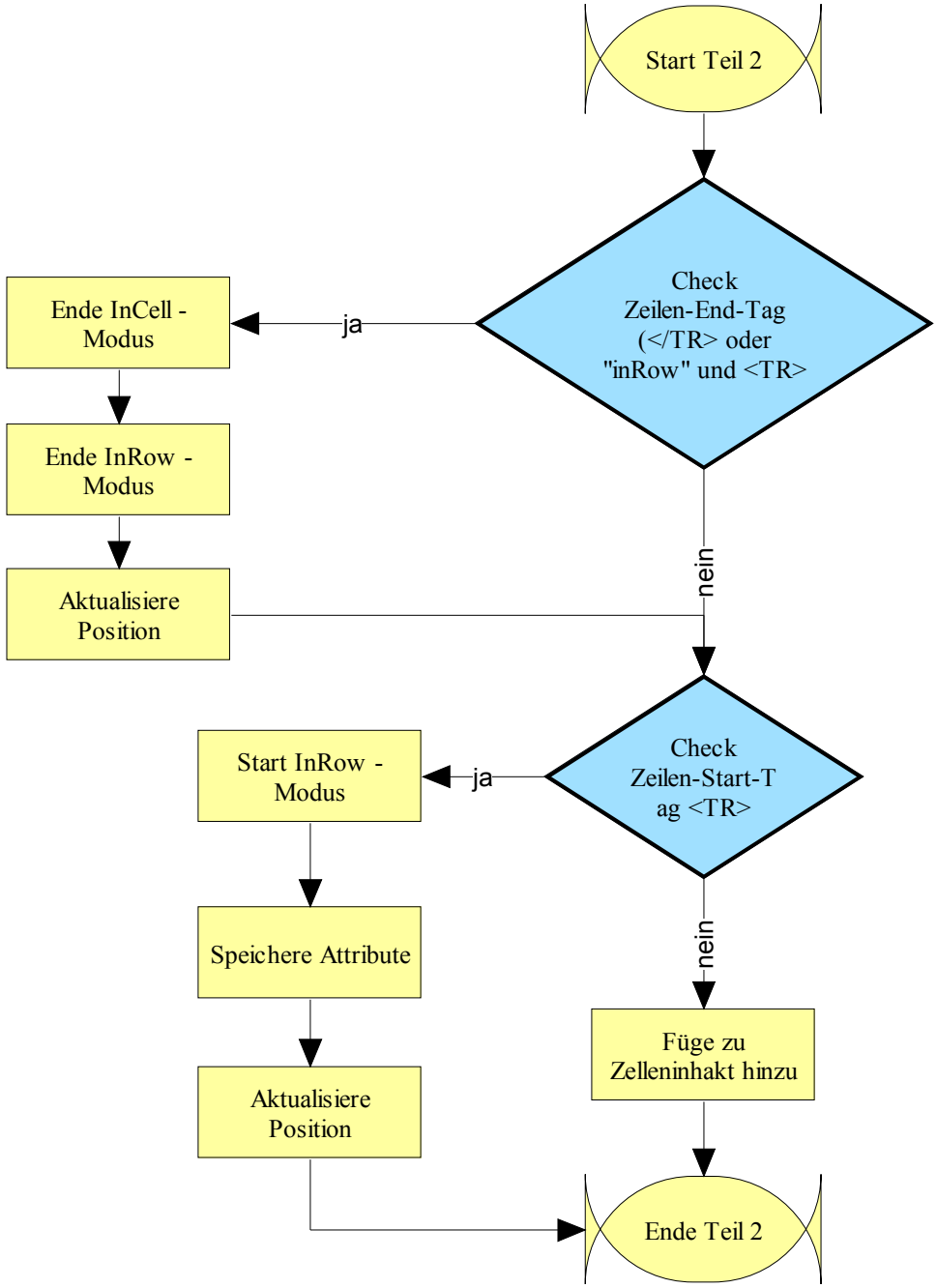


Abb. 3 – Ablauf Parser, Teil 2

4.6. Analyse

Die Aufgabe der Analyse-Routine ist es, die Headerzeile (die ja nicht unbedingt die erste Zeile sein muss) ausfindig zu machen. Die Headerzeile wird für die Unterdrückung des Klonens gebraucht.

Es wird die Tabelle im Speicher Zeile für Zeile durchsucht, bis eine Zelle mit gesetztem Header-Attribut gefunden wird. Sollte in einer weiteren Zeile ein Header-Eintrag auftauchen, so wird das Programm abgebrochen. Ebenso wird das Programm abgebrochen, wenn keine Header-Zeile gefunden wird.

Die Headerzeile kann in den beiden oben genannten Fällen manuell gesetzt werden bzw. die automatische Erkennung überstimmt werden.

Nach dem ersten Teil der Header-Erkennung erfolgt nun das Unterdrücken des Klonens für bestimmte Zellen.

Es wird beim Parsen gespeichert, ob die Zelle eine geklonte Zelle (durch ROWSPAN oder COLSPAN oder beides) ist oder ob es eine normale Zelle ist.

Danach wird fest gestellt, ob die Header-Spalte (zur aktuellen Spalte) ebenfalls geklont worden ist. Ist dies der Fall, so wird für die Zelle das Unterdrücke-Klonen-Flag gesetzt. Das bewirkt bei der Ausgabe, dass in diese Zelle nicht die Information der ursprünglichen Zelle geschrieben wird.

Sinn dieser Maßnahme ist es, bei bestimmten Anordnungen, die vor allem der Übersicht als auch der hierarchischen Strukturierung dienen, eine Möglichkeit zu haben, nicht alle Zellen zu füllen, vor allem wenn dies schon in der Überschrift angewendet wird.

4.7. HTML-Ausgabe

Das Programm versucht den Output der HTML-Datei dem Input möglichst ähnlich zu machen. Selbst die Attribute der Tabellen-Tags bleiben erhalten, ausgenommen die Attribute COLSPAN und ROWSPAN, die zu diesem Zeitpunkt nicht mehr von Interesse sind, da sie durch vorgehende Arbeitsschritte in einzelne Zellen umgewandelt wurden.

Der erste (und letze) Schritt ist die Ausgabe der Tags und Texte vor und nach der Tabelle, die in eigenen Listen gespeichert werden.

Dazwischen wird die dynamische Tabelle im Speicher ausgeben, je nach Parametrisierung mit oder ohne Unterdrückung der im Analyse Schritt markierten Zellen.

4.8. XML-Ausgabe

Die DTD des XML-Outputs hat folgendes DTD-Format:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT table (size,row+)>
<!ELEMENT size EMPTY>
<!ATTLIST size widthX CDATA "0" >
<!ATTLIST size widthY CDATA "0" >

<!ELEMENT row (cell*)>

<!ELEMENT cell (#PCDATA)>
<!ATTLIST cell isHeader (true|false) "false">
```

Die Tabelle (und nur die Tabelle, der restliche HTML-Teil vor / nach / in der Tabelle wird aufgrund der Nicht-Konvertierbarkeit verworfen) wird im XML-Element TABLE gespeichert.

Das TABLE Element besteht aus zwei Teilen:

SIZE

einer zwingenden Angabe der Größe der folgenden Tabelle im rechteckigen Maximalformat. Dies ist zum Parsen nicht unbedingt erforderlich, soll aber die Übersetzung in weiteren Programmen erleichtern, vor allem, wenn ein zweidimensionaler Array als Datenspeicher verwendet werden soll.

Size hat zwei Attribute:

WidhtX – als Ausdehnung in horizontaler Richtung des zweidimensionalen Felds

WidhtY – als Ausdehnung in vertikaler Richtung des zweidimensionalen Felds

Ein Beispiel:

```
<size widthX="3" widthY="3"/>
```

ROW

Enthält die „Nutzdaten“ der Tabelle in Reihenform. Dieses Element kann sich beliebig oft wiederholen

Beispiel:

```
<row>
<cell>...</cell>....<cell>....</cell>
</row>
```

Jede ROW besteht, wie schon gezeigt wurde, aus einzelnen Zellen.

CELL

Die Zellen enthalten die eigentliche Information. Zwischen dem Start- und dem End-Tag ist die eigentliche Nutzinformation als Text platziert.

Zusätzlich können sie mittels des Attributes isHeader als Headerzelle gekennzeichnet werden.

Beispiel:

<cell>normaler Inhalt</cell>
<cell isHeader=“true“>Kopf-Inhalt</cell>

Als Beispiel sei eine Tabelle aufgeführt:

A	***A***	***B***
1	2	3
4	5	6

Der obigen DTD-Definition zufolge sieht dann die entsprechende XML-Datei folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE table SYSTEM "detable.dtd">
<table>
<size widthX="3" widthY="3"/>
<row>
<cell isHeader="true">A</cell><cell isHeader="true"></cell><cell
isHeader="true">A</cell><cell isHeader="true">B</cell></row>
<row>
<cell>1</cell><cell>2</cell><cell>3</cell></row>
<row>
<cell>4</cell><cell>5</cell><cell>6</cell></row>
</table>
```

5. Zusammenfassung und Ausblick

Es konnte mit dem entwickelten Programm eine Lösung für die Umwandlung von komplexen HTML-Tabellen in XML gefunden werden. Es kann die Testdateien erfolgreich umsetzen.

Dieses Programm ist aber auf die Umwandlung von Tabellen beschränkt, und bietet keine Funktionalität zum Wrappen von anderen HTML-Texten. Damit ist der Einsatz dieses Tools nur auf einen eingeschränkten Aufgabenbereich möglich.

HTML hat in seiner Geschichte gezeigt, dass die HTML-Tags zum Großteil für das Layout der Seite verwendet werden. Dies führt dazu, dass nur ein geringer Teil der Seite aus dem eigentlichen gewünschten Text besteht (siehe auch [7] für die 1995 meist verwendeten HTML-Tags). Die Layout-Information lässt aber keinen eindeutigen Schluss auf die Strukturierung zu, da HTML nur wenig Layoutmöglichkeiten bietet und zur Gestaltung einer „ansprechenden“ Seite tief in die Trickkiste von HTML und Javascript gegriffen werden muss. Zum Zeitpunkt der Verfassung dieses Dokuments (2004) ist der HTML-Standard über längere Zeit unverändert geblieben. Es lässt sich im Moment nicht absehen, wohin sich der HTML-Standard in Zukunft weiterentwickeln wird,

Die Initiativen zum „barrierefreien Zugang“ zu Informationsinhalten dürften einen Vorteil in die Richtung der einfacheren Extraktion von Daten bringen (wenn auch als Nebeneffekt). Um die Verwendung von Programmen zum Vorlesen von HTML-Seiten (für Blinde) oder Programme zur Unterstüzung bei andere Behinderungen zu erleichtern, wird die Layoutinformation auf ein absolutes Minimum zurückgedrängt. Dies erhöht den Anteil der echten Information zur Layoutinformation und dürfte die Ergebnisse von Wrappern verbessern.

6. Literaturverzeichnis

[1] Robert Baumgartner, Sergio Flesca, Georg Gottlob; Visual Web Information Extraction with Lixto, The VLDB Journal, p. 119-128, 2001

[2] Ling Liu, Calton Pu, Wei Han; XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources, ICDE, p. 611-621, 2000

[3] A. Laender, B. Ribeiro-Neto, A. Silva, J. Teixeira; A Brief Survey of Web Data Extraction Tools, SIGMOD Record, vol. 31, 2, June, 2002

[4] L. Peshkin and A. Pfeffer; Bayesian Information Extraction Network, IJCAI 2003

[5] Y. Papakonstantinou, H. Garcia-Molina, J. Widom; Object Exchange Across Heterogeneous Information Sources, 11th Conference on Data Engineering, 1995

[6] B. Chidlovskii, U. M. Borghoff, P.-Y. Chevalier; Towards Sophisticated Wrapping of Web-based Information Repositories, Proceedings of 5th International RIAO Conf. 123-35, 1997

[7] A. Woodruff, P. M. Aoki, E. Brewer, P. Gauthier, L. A. Rowe; An Investigation of Documents from the World Wide Web, Computer Networks and ISDN Systems, Vol. 28, p. 963-980, 1996

[8] N. Kushmerick, D. S. Weld, R. B. Doorenbos; Wrapper Induction for Information Extraction, Intl. Joint Conference on Artificial Intelligence (IJCAI), p. 729-737, 1997

7. Beispiel

Nachfolgend soll das Programm an einer Beispieltabelle dargestellt werden. Das Programm wurde mit Standardparametern gestartet.

7.1. Bildschirmausgabe

```
C:\detable>java DeTable -i "acute asthma.html" -o test.html -x test2.xml


DeTable 1.1.3 - written by guenther sommer 2003-2004, contact gue-at-rettung.at
developed for the Institute for Software Technology and Interactive Systems
at the Vienna University of Technology
this app reads table in html file and splits up combined cells and puts it out
as html or xml
licensed under the GPL v2, for further information see www.fsf.org


reading configuration ...
reading input file ...
parsing table ...
writing html file ...
writing xml file ...
finished
```

7.2. Input

cause			*drug of choice*	*dosage*
adults	Gonococcus		Ceftriaxone	1g IM, single dose
				lavage infected eye
	Chlamydia		Azithromycin	1g orally single dose
			or	
			Doxycycline	100 mg orally twice a day for 7 days
children	Gonococcus	Children who weigh < 45 kg	Ceftriaxone	125 mg IM, single dose
		Children who weigh > 45 kg		same treatment as adults
	Chlamydia	Children who weigh < 45 kg	Erythromycin base	50 mg/kg/day orally in 4 divided doses for 10-14 days
		Children under 8 years old who weigh > 45 kg	Azithromycin	1 gm orally, single dose
		Children 8 years old or older	Azithromycin	1 gm orally, single dose
			or	
			Doxycycline	100 mg orally, twice a day for 7 days

Neonates	Ophthalmia neonatorum (Caused by N. gonorrhoeae)	Ceftriaxone	25-50 mg/kg IV or IM, single dose, not to exceed 125 mg
	Chlamydia	Erythromycin	50 mg/kg/day orally in 4 divided doss for 10-14 days

7.3. Ergebnis (HTML)

cause	*cause*	*cause*	*drug of choice*	*dosage*
adults	Gonococcus		Ceftriaxone	1g IM, single dose
adults	Gonococcus			lavage infected eye
adults	Chlamydia		Azithromycin	1g orally single dose
adults	Chlamydia		or	or
adults	Chlamydia		Doxycycline	100 mg orally twice a day for 7 days
children	Gonococcus	Children who weigh < 45 kg	Ceftriaxone	125 mg IM, single dose
children	Gonococcus	Children who weigh > 45 kg		same treatment as adults
children	Chlamydia	Children who weigh < 45 kg	Erythromycin base	50 mg/kg/day orally in 4 divided doses for 10-14 days
children	Chlamydia	Children under 8 years old who weigh > 45 kg	Azithromycin	1 gm orally, single dose
children	Chlamydia	Children 8 years old or older	Azithromycin	1 gm orally, single dose
children	Chlamydia	Children 8 years old or older	or	or
children	Chlamydia	Children 8 years old or older	Doxycycline	100 mg orally, twice a day for 7 days
Neonates	Ophthalmia neonatorum (Caused by N. gonorrhoeae)		Ceftriaxone	25-50 mg/kg IV or IM, single dose, not to exceed 125 mg
Neonates	Chlamydia		Erythromycin	50 mg/kg/day orally in 4 divided doss for 10-14 days

7.4. Ergebnis (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE table SYSTEM "detable.dtd">
<table><size widthX="5" widthY="15"/>
```

<row><cell isHeader="true">cause</cell><cell isHeader="true"></cell><cell isHeader="true"></cell><cell isHeader="true">drug of choice</cell><cell isHeader="true">dosage</cell></row>

<row><cell>adults</cell><cell>Gonococcus</cell><cell></cell><cell>Ceftriaxone</cell><cell>1g IM, single dose</cell></row>

<row><cell>adults</cell><cell>Gonococcus</cell><cell></cell><cell></cell><cell>lavage infected eye</cell></row>

<row><cell>adults</cell><cell>Chlamydia</cell><cell></cell><cell>Azithromycin</cell><cell>1g orally single dose</cell></row>

<row><cell>adults</cell><cell>Chlamydia</cell><cell></cell><cell>or</cell><cell>or</cell></row>

<row><cell>adults</cell><cell>Chlamydia</cell><cell></cell><cell>Doxycycline</cell><cell>100 mg orally twice a day for 7 days</cell></row>

<row><cell>children</cell><cell>Gonococcus</cell><cell>Children who weigh < 45 kg</cell><cell>Ceftriaxone</cell><cell>125 mg IM, single dose</cell></row>

<row><cell>children</cell><cell>Gonococcus</cell><cell>Children who weigh > 45 kg</cell><cell></cell><cell>same treatment as adults</cell></row>

<row><cell>children</cell><cell>Chlamydia</cell><cell>Children who weigh < 45 kg</cell><cell>Erythromycin base</cell><cell>50 mg/kg/day orally in 4 divided doses for 10-14 days</cell></row>

<row><cell>children</cell><cell>Chlamydia</cell><cell>Children under 8 years old who weigh > 45 kg</cell><cell>Azithromycin</cell><cell>1 gm orally, single dose</cell></row>

<row><cell>children</cell><cell>Chlamydia</cell><cell>Children 8 years old or older</cell><cell>Azithromycin</cell><cell>1 gm orally, single dose</cell></row>

<row><cell>children</cell><cell>Chlamydia</cell><cell>Children 8 years old or older</cell><cell>or</cell><cell>or</cell></row>

<row><cell>children</cell><cell>Chlamydia</cell><cell>Children 8 years old or older</cell><cell>Doxycycline</cell><cell>100 mg orally, twice a day for 7 days</cell></row>

<row><cell>Neonates</cell><cell>Ophthalmia neonatorum (Caused by N. gonorrhoeae)</cell><cell></cell><cell>Ceftriaxone</cell><cell>25-50 mg/kg IV or IM, single dose, not to exceed 125 mg</cell></row>

<row><cell>Neonates</cell><cell>Chlamydia</cell><cell></cell><cell>Erythromycin</cell><cell>50 mg/kg/day orally in 4 divided doss for 10-14 days</cell></row>

</table>

8. Anhang

8.1. Konfigurationsparameter

-? oder --help
zeigt den Hilfetext an, wird auch angezeigt, wenn kein Parameter übergeben wird

-c DATEI oder –config DATEI
es lässt sich hiermit eine Konfigurationsdatei namens DATEI angeben

-r ZAHL oder –row ZAHL
damit lässt sich die Reihe ZAHL als Kopfzeile erzwingen. Normalerweise wird die Kopfzeile automatisch anhand des <TH> Tags erkannt. Wenn es keine Kopfzeile gibt, oder eine andere gewünscht wird, kann der Parameter verwendet werden. Die Nummerierung startet mit 1.

-s oder --no-supression
Wie weiter unten erfolgt mittels diesem Flag eine Deaktivierung der Klon-Unterdrückung von Zellen, wenn die entsprechenden Zellen der Kopfzeile den gleichen Inhalt haben. Die restliche Umwandlung bzw. Bearbeitung des Programms bleibt davon unangetastet.

-i DATEI oder --input-file DATEI
Die HTML-Datei mit dem Namen DATEI wird als Eingabedatei verwendet. Dieser Parameter ist zwingend.

-h DATEI oder --html-file DATEI
Die Datei mit dem Namen DATEI wird als HTML-Ausgabedatei verwendet. Entweder dieser oder der Parameter -x bzw. --xml-file sollten angegeben werden, da das Programm sonst keinen Output erzeugt.

-x DATEI oder --xml-file DATEI
Die Datei mit dem Namen DATEI wird als XML-Ausgabedatei verwendet. Entweder dieser oder der Parameter -h bzw. --html-file sollten angegeben werden, da das Programm sonst keinen Output erzeugt.

8.2. Readme

Nachfolgende die README.TXT der Applikation:

1. Generic

this application reads table in html file and splits up combined cells and puts it out as html or xml

this application was developed for the Institute for Software Technology and Interactive Systems at the Vienna University of Technology in 2003-2004 by Guenther SOMMER (gue-at-rettung.at).

for questions contact kaiser-at-ifs.tuwien.ac.at

this application is licensed unde the GPL v2, for further information see www.fsf.org

2. Compiling / Installation
This software was written with JDK 1.3.1, but should run with 1.4.2 too (tested)
It needs only the jdk-libraries and no external ones.

To compile:
javac DeTable.java

3. Running
Execute it with:
java DeTable
and you get a list of configuration options.

Have fun and live long and prosper
Guenther Sommer