

Andreas Deininger

# Nutzungsmöglichkeiten der Extensible Markup Language (XML) und Stylesheet Language Transformations (XSLT) für die Darstellung und Transformation landwirtschaftlicher Informationen

Das World Wide Web Consortium hat als neue Seitenbeschreibungssprache für Dokumente, welche über das Internet verbreitet werden, die Extensible Markup Language (XML) vorgeschlagen. Nachfolgend wird dieses Sprachmodell vorgestellt und aufgezeigt, wie XML-Daten transformiert und in eigenen Applikationen verwendet werden können.

## 1 Einführung

In weniger als einem Jahrzehnt hat sich das Internet als Mittel der Informationsübertragung etabliert und dabei weite Bereiche der Informationstechnologie revolutioniert. Einer der wichtigsten, wenn nicht sogar der wichtigste Internet-basierte Dienst ist dabei das World Wide Web (WWW), welcher speziell für den Austausch von Dokumenten in elektronischer Form über die Datenwege des Internet entwickelt wurde. Für die Codierung der Dokumente kommt dabei eine ursprünglich von Tim Berners-Lee am CERN entwickelte Seitenbeschreibungssprache, die sog. hyper text markup language (HTML) zum Einsatz. Bereits 1993 entwickelt, liegt HTML mittlerweile in der Spezifikation 4.01 vor, welche vom W3C-Consortium im Dezember 1999 publiziert wurde (W3C 1999, [21]). Im Lauf der Entwicklung wurden in HTML, welches ursprünglich lediglich zur Beschreibung der Dokumentstruktur gedacht war, immer mehr Sprachelemente eingeführt, welche eine Aussage über das konkrete Aussehen und die Formatierung des Dokumentes treffen. Das W3C-Consortium versuchte zunächst dieser Entwicklung, etwa durch die Einführung von Cascading Stylesheets (CSS) (W3C 1999, [17, 20]), Einhalt zu gebieten, hat sich dann aber gänzlich von HTML abgewandt und im Februar 1998 ein komplett neues Sprachmodell für die Codierung von Dokumenten als Empfehlung vorgeschlagen, die Extensible Markup Language (XML) (W3C 1998, [16]). Nach den Vorstellungen des W3C-Consortiums soll diese Sprache langfristig HTML als lingua franca das WWW ablösen. Nachfolgend soll das Grundkonzept dieser Sprache dargestellt werden. Ferner soll kritisch an Anwendungsbeispielen aus der Landwirtschaft untersucht werden, welche Vorteile sich tatsächlich aus der Nutzung dieser neuen Sprache ergeben können.

## 2 Darstellung von Daten mittels Extensible Markup Language

Ein wesentliches Charakteristikum ist die Flexibilität und Erweiterungsfähigkeit dieser Sprache, die als eine Art Meta-Sprache zu verstehen ist, also lediglich eine Aussage darüber trifft, wie ein Dokument generell aufgebaut sein muss. Es wird also lediglich ein Sprachrahmen zur Verfügung gestellt, der es dem Benutzer ermöglichen soll, ein für seine Belange relevantes Sprachmodell zu definieren und damit seine vorliegenden Daten zu codieren.

### 2.1 Generelle Charakteristika von XML-Dokumenten

XML-Dokumente bestehen aus Elementen, welche Attribute enthalten können. Die Elementnamen werden als sog. Tags zwischen eckige Klammern gesetzt, wobei in wohlgeformten XML-Dokumenten für jedes Element ein öffnendes und ein schließendes Tag vorhanden sein muss. Zwischen dem öffnenden und dem schließenden Tag selbst steht der Elementinhalt. Dem Element können ein oder mehrere Attribute beigefügt werden, für die jeweils ein Wert anzugeben ist. Für einen Artikel „Schraube“, für den zusätzlich der Lagerbestand als Attribut definiert ist, könnte die Definition wie folgt aussehen: <Artikel Lagerbestand="524">Schraube</Artikel>. Selbstverständlich können Elemente selbst wiederum Elemente enthalten, was den Aufbau komplexer Dokumente erlaubt.

### 2.2 Darstellung eines Tränkeplanes in XML als Anwendungsfall

In der landwirtschaftlichen Praxis sind prozessrechnergesteuerte Tränkeautomaten für die Aufzucht von Saugkälbern weit verbreitet (BÜSCHER und KÄCK 1995). Bei solchen Automaten wird die Höhe der Tränkegabe, die Konzentration

der Tränke, sofern Milchaustauscher vertränkt wird, sowie das Verhältnis von Frischmilch und Milchaustauscher, sofern beide Komponenten zum Einsatz kommen, in einem sog. Tränkeplan definiert. Solch ein Tränkeplan kann sehr einfach in einem XML-Dokument codiert werden, wie nachfolgend dargestellt:

**Listing 1: Tränkeplan als XML-Dokument**

```

1 <?xml version="1.0" encoding='ISO-8859-1' ?>
2 <!DOCTYPE Tränkeplan SYSTEM "planattr.dtd">
3 <Tränkeplan Tränkepreis="0.50">
4   <Mengenplan>
5     <Startperiode Länge="3" Startwert="6" Endwert="6" />
6     <Periode Länge="14" Endwert="8" />
7     <Periode Länge="18" Endwert="8" />
8     <Periode Länge="92" Endwert="2" />
9   </Mengenplan>
10  <Konzentrationsplan MAT-Preis="300">
11    <Startperiode Länge="10" Startwert="90" Endwert="120" />
12    <Periode Länge="67" Endwert="120" />
13  </Konzentrationsplan>
14  <Milchanteilplan Milchwert="120">
15    <Startperiode Länge="5" Startwert="100" Endwert="100" />
16    <Periode Länge="7" Endwert="100" />
17  </Milchanteilplan>
18 </Tränkeplan>

```

Zeile 1 weist das Dokument als XML-Dokument aus und spezifiziert die Codierung, um eine korrekte Umsetzung der Umlaute in dem Dokument zu gewährleisten. Zeile 2 deklariert das Dokument als Tränkeplan, welcher anhand einer DOCTYPE-Definition validiert werden kann, welche lokal auf dem System in der Datei planattr.dtd hinterlegt ist. Von Zeile 3 ab wird der Tränkeplan definiert, welcher aus einem Mengenplan (Zeilen 4-9), einem Konzentrationsplan (Zeilen 10-13), und einem Milchanteilplan (Zeilen 14-17) besteht. Jeder Plan besteht aus mindestens einer Startperiode sowie aus beliebig vielen weiteren Perioden. Für diese Perioden sind deren Länge sowie die Planwerte zu Periodenbeginn und -ende (erstes lediglich im Falle der Startperiode) in Form von Attributen zu definieren, welche den Elementdefinitionen für die Perioden hinzugefügt werden. Auch der Preis für die Tränke und den Milchaustauscher sowie der Milchwert sind als Attribute den Elementen Tränkeplan, Konzentrationsplan bzw. Milchanteilplan hinzugefügt. Alternativ hätten diese Werte auch als eigenständige Elemente definiert werden können, etwa in der Form `<MAT-Preis>230</MAT-Preis>`. Daraus wird ersichtlich, dass das vorgestellte XML-Dokument nicht die einzig mögliche Repräsentation eines Tränkeplans darstellt, man hätte auch ohne weiteres sämtliche Attribute in Elementform codieren können. Dadurch wäre derselbe Informationsgehalt in einer anderen, allerdings umfangreicheren Form codiert worden.

## 2.3 Definition von XML-Dokumenten

XML-Dokumente können sich im Wesentlichen durch zwei Eigenschaften auszeichnen (GOOSENS und RAHTZ 1999):

- Wohlgeformtheit: Diese sehr allgemeine, für jedes XML-Dokument zwingend erforderliche Eigenschaft besagt lediglich, dass die Elementdefinitionen korrekt erfolgt sind sowie dass keine ungültige Schachtelung der einzelnen Elemente vorliegt.

- Gültigkeit: Über die Wohlgeformtheit hinaus sollte ein XML-Dokument gewissen Ansprüchen an seinen Inhalt genügen: im obigen Beispiel etwa sollte sichergestellt sein, dass für den Tränkeplan wiederum zumindest ein Mengenplan definiert wurde, dass für jede Periode die Attribute Länge sowie Eckwert(e) spezifiziert wurden usw. Um ein Dokument daraufhin überprüfen zu können, muss die überprüfende Instanz auf einen Satz von Regeln zurückgreifen können. Das Aussehen dieser Regeln soll nachfolgend für das Beispiel des in der Datei plan.xml codierten Tränkeplans erläutert werden. Dabei ist zu beachten, dass verschiedene Modelle zur Formulierung der Regeln für die Überprüfung zum Einsatz kommen können. Nachfolgend ist die Validierung des XML-Dokuments anhand einer Dokumententyp-Definition sowie anhand des XML-Schema-Modells ausführlicher erläutert.

### 2.3.1 Definition mittels DOCTYPE-Anweisungen

Die traditionelle Methode zieht zur Überprüfung der Gültigkeit eines Elements die Anweisungen in der DOCTYPE-Definition zu Rate, welche sowohl intern in der XML-Datei aufgeführt als auch, wie in unserem Fall, in einem separaten File namens plan.dtd hinterlegt werden kann. Der Inhalt dieser Datei ist nachfolgend wiedergegeben:

**Listing 2: Doctype-Definition einer Tränkeplan-Instanz**

```

1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2 <!-- DTD Tränkeplan "plan.dtd" -->
3 <!ELEMENT Tränkeplan (Mengenplan | (Mengenplan,
4   Konzentrationsplan), (Mengenplan, Konzentrati-
5   onsplan, Milchanteilplan)>
6 <!ATTLIST Tränkeplan
7   Tränkepreis CDATA #REQUIRED>
8 <!ELEMENT Mengenplan (Startperiode, (Perio-
9   de?, Periode?, Periode?, Periode?))>
10 <!ELEMENT Konzentrationsplan (Startperiode,
11   (Periode?, Periode?, Periode?, Periode?))>
12 <!ATTLIST Konzentrationsplan
13   MAT-Preis CDATA #REQUIRED>
14 <!ELEMENT Milchanteilplan (Startperiode, (Perio-
15   de?, Periode?, Periode?, Periode?))>
16 <!ATTLIST Milchanteilplan
17   Milchwert CDATA #REQUIRED><!ELEMENT Startpe-
18   riode EMPTY>
19 <!ATTLIST Startperiode
20   Länge CDATA #REQUIRED
21   Startwert CDATA #REQUIRED
22   Endwert CDATA #REQUIRED>
23 <!ELEMENT Periode EMPTY>
24 <!ATTLIST Periode
25   Länge CDATA #REQUIRED
26   Endwert CDATA #REQUIRED>

```

Zeile 3 von Listing 2 legt den Inhalt des Elements Tränkeplan fest: Dieser darf lediglich aus einem Mengenplan, einer Kombination aus Mengen- und Konzentrationsplan oder aus je einem Mengen-, Konzentrations- und Milchanteilplan bestehen. Darüber hinaus dürfen keine weiteren Elemente enthalten sein. Die Zeilen 4 und 5 definieren die Attributliste des Elements Tränkeplan: Zu jedem Tränkeplan muss zwingend (#REQUIRED) ein Attribut namens Tränkepreis definiert werden, dessen Inhalt beliebige Zeichen (Character DATA) sein dürfen. In Zeile 6 wird festgelegt, dass ein Mengenplan, wie ein Konzentrations- und ein Mengenplan auch, zwingend aus einer Startperiode sowie aus bis zu 4

weiteren Perioden bestehen darf. Das '?' hinter dem Element Periode erlaubt für dieses Element das null- oder einmalige Auftreten innerhalb des übergeordneten Elements. Schließlich werden in den Zeilen 12-20 noch die Elemente Startperiode und Periode selbst mit den dazugehörigen Attributen deklariert.

### 2.3.2 Definition mittels XML Schemas

Die oben erläuterte DOCTYPE-Syntax erweist sich als einfaches Werkzeug zur Definition von XML-Dokumenten, hat aber Grenzen hinsichtlich ihrer reglementierenden Vorgaben. Als Beispiel wäre etwa die Beschränkung der Anzahl der Perioden eines Planes auf 4 Stück in den Zeilen 6, 7 und 10 zu nennen. Es ist unschwer einzusehen, dass eine Restriktion auf 100 Elemente zu überlangen, fehlerträchtigen Dokumenten führen würde. Ferner ist zu beachten, dass die obige DOCTYPE-Definition nicht als wohlgeformtes XML-Dokument validiert werden kann, da die XML-Syntax keine Verwendung findet. Aus all diesen Gründen hat das W3C-Konsortium im Februar 2000 einen Entwurf für ein neues Validierungsmodell für XML-Dokumente vorgelegt, das sog. XML-Schema-Modell (W3C 2000, [22]). Eine Validierungsregel nach diesem Modell für das obige XML-Dokument ist nachfolgend □ aus Platzgründen in verkürzter Form □ wiedergegeben:

Listing 3: XML-Schema zur Definition einer Tränkeplan-Instanz

```

1  <schema>
2  <element name="Tränkeplan">
3  <complexType content="elementOnly">
4  <choice>
5  <element ref="Mengenplan"/>
6  <group>
7  <element ref="Mengenplan"/>
8  <element ref="Konzentrationsplan" minOccurs="1" maxOccurs="1"/>
9  </group>
10 <group>
11 <element ref="Mengenplan"/>
12 <element ref="Konzentrationsplan" minOccurs="1" maxOccurs="1"/>
13 <element ref="Milchanteilplan" minOccurs="1" maxOccurs="1"/>
14 </group>
15 </choice>
16 <attribute name="Tränkepreis" type="atttprice" use="default" value="50"/>
17 </complexType>
18 </element>
19 <element name="Mengenplan">
20 <complexType content="elementOnly">
21 <element ref="Startperiode"/>
22 <element ref="Periode" minOccurs="0" maxOccurs="4"/>
23 </complexType>
24 </element>
25 (...)
61 <simpleType name="Periodenlänge" base="nonNegativeInteger">
62 <minInclusive value="2"/>
63 <maxExclusive value="99"/>
64 </simpleType>
65 <simpleType name="Planwert" base="decimal">
66 <minInclusive value="0"/>
67 <maxExclusive value="255"/>
68 </simpleType>
69 </schema>

```

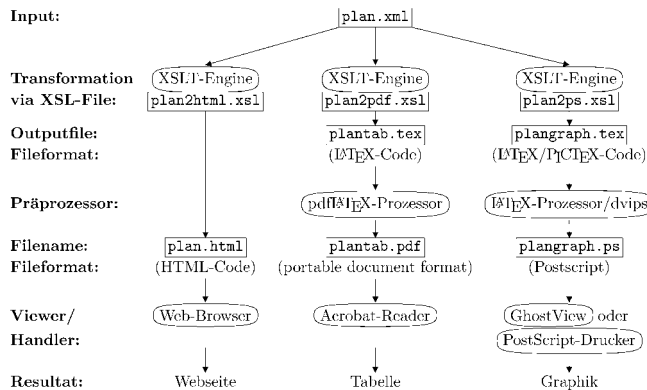
In den Zeilen 2-18 erfolgt die Definition des Elements Tränkeplan. Dieser kann entweder nur aus einem Mengenplan (Zeile 5), einem Mengen- und einem Konzentrationsplan (Zeile 7+8) oder aus einer Kombination von Mengen-, Konzentrations- und Milchanteilplan bestehen (Zeilen 11-13); ferner wird in Zeile 16 das jetzt optionale Attribut Tränkepreis definiert, welches, falls nicht spezifiziert, mit dem Vorgabewert 50 (Pf/l Tränke) besetzt wird. Die abschließenden Zeilen 61 bis 69 demonstrieren die Vorzüglichkeit der XML-Schema-Sprache gegenüber der in Listing 2 verwendeten DocType-Definiton: über die Definition der beiden neuen, auf dem einfachen Typ 'nonNegativeInteger' beruhenden Datentypen Periodenlänge bzw. Planwert wird sichergestellt, dass lediglich Zahlen innerhalb bestimmter Grenzen als zulässige Werte innerhalb des XML-Dokuments validiert werden.

## 3 Transformation von Daten mittels Extensible Stylesheet Language Transformations

### 3.1 Generelle Darstellung der XSL- Transformationsprache

Ein wesentliches Kennzeichen des XML-Konzepts ist die vollständige Trennung von Form und Inhalt in den verwendeten Dateien. Die XML-Datendatei soll sich dabei lediglich auf die Speicherung und Auszeichnung der einzelnen Elemente beschränken. Sollen diese Daten dann in irgendeiner Form, etwa in Form von Graphiken oder Tabellen, dargestellt werden, so muss eine Transformation in das benötigte Output-Format vorgenommen werden. Zur Durchführung dieser Transformation wurde vom W3-Consortium eine XML-basierte Transformationssprache in Form der Extensible Stylesheet Language Transformations (XSLT) vorgeschlagen (W3C 1999, [19]). Diese Sprache gestattet es, XML-Dokumente in das gewünschte Output-Format zu überführen, wobei im Rahmen dieses Transformationsprozesses über die Auswahl und Formatierung einzelner Elemente aus der XML-Datei bestimmt werden kann. Dadurch erweist sich XSLT als ein sehr flexibles Werkzeug zur Erzeugung der zu generierenden Output-Datei. In Tabelle 1 ist dieses Konzept tabellarisch für den bereits in Kapitel 2.2 vorgestellten, in XML-codierten Tränkeplan erläutert. Ausgehend von dem XML-Input in Form der Datei plan.xml (Listing 1) werden beispielhaft drei verschiedene Transformationen beschrieben, um mit den Daten des XML-Files den codierten Tränkeplan entweder als Webseite, als Tabelle oder als Graphik darzustellen. Jede dieser Darstellungsmethoden verlangt dabei seitens der den Output generierenden Instanz ein eigenes Fileformat. Im Falle der Webseite wird von dem Web-Browser eine HTML-Seite benötigt, und für die Darstellung der Tabelle soll eine Datei im portable document format (pdf) zum Einsatz kommen, welches mittels des Acrobat-Readers gelesen werden kann. Die graphische Darstellung des Tränkeplans schließlich soll als PostScript-Datei erfolgen, welche von einem postscriptfähigen Drucker ausgedruckt werden oder mit Hilfe des Programms Ghostview am Bildschirm betrachtet werden kann. Im günstigsten Falle, wie etwa im Falle der Webseite, können die benötigten Ausgabeformate (hier in Form der HTML-Datei) direkt über eine Transformation aus der XML-Datei gewonnen werden. Diese Transformation wird von einem XSL-Transformations-Prozessor durchgeführt, der seine Transformationsanweisungen aus einer in der XSLT-Sprache codierten Datei, hier plan2html.xml (vgl. Listing 4) erhält.

Tab.1: Transformation von XML-Daten in verschiedene Ausgabeformate



In den beiden anderen beschriebenen Anwendungsfällen kann die Output-Datei nicht direkt von dem XSLT-Prozessor generiert werden, sondern wird von einem Präprozessor erzeugt, welcher als Input die im Rahmen des Transformationsprozesses quasi als Zwischenstufe erzeugte Outputdatei des XSLT-Prozessors verarbeitet. Durch das beschriebene Vorgehen wird ein wesentlicher Vorteil des Konzeptes der Trennung von Inhalt und Form deutlich. Soll die Darstellungsweise der Datei geändert werden, so sind lediglich Änderungen an der XSLT-Datei erforderlich, soll eine Datei in einem weiteren Ausgabeformat erzeugt werden, so ist lediglich eine zusätzliche Transformationsdatei zu erstellen. In beiden Fällen bleiben die Plandaten in der XML-Datei dabei unberührt.

### 3.2 Beispielhafte Anwendungsmöglichkeiten für Transformationen

#### 3.2.1 Transformation von XML-Daten in HTML-Dateien

Einen sehr häufigen Anwendungsfall stellt die Transformation von intern gespeicherten XML-Dateien in das HTML-Format zur Darstellung der Daten in einem Webbrowser dar. Diese Transformation soll beispielhaft für die in Kapitel 2.2 beschriebene Datei plan.xml erläutert werden. Die Transformation selbst wird von dem XSL-Transformations-Prozessor vorgenommen, der seine Anweisungen für die Auswahl und Formatierung der einzelnen Elemente über die nachfolgend wiedergegebene Datei plan2html.xsl erhält:

Listing 4: XSLT-Instruktionen zur Transformation in HTML-Code

```

1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xslb="http://www.w3.org/TR/WD-xsl">
3   <xsl:template match="/">
4     <xslb:eval no-entities="true"/>
5     <html>
6       <head>
7         <title>Tränkeplan - XML-Markup</title>
8         <style type="text/css">
9           div.c2 {text-align: center}
10          hl.c1 {color: red}
11        </style>
12      </head>
  
```

```

13   <body>
14     <div class="c2">
15       <h1 class="c1">Tränkeplan</h1>
16       <table border="1" width="75%" cellpadding="0" cellspacing="0">
17         <tr bgcolor="white">
18           <th align="center" colspan="11"><b>Mengenplan</b></th>
19         </tr>
20         <tr bgcolor="yellow">
21           <td align="center"><b>Tage</b></td>
22           <td align="center">P1:</td>
23           <td align="center">
24             <xsl:value-of select="Tränkeplan/Mengenplan/Startperiode/@Länge" />
25           </td>
26           <td align="center">P2:</td>
27           <td align="center">
28             <xsl:value-of select="Tränkeplan/Mengenplan/Periode[1]/@Länge"/>
29           </td>
30           <td align="center">P3:</td>
31           <td align="center">
32             <xsl:value-of select="Tränkeplan/Mengenplan/Periode[2]/@Länge"/>
33           </td>
34         </tr>
35         (... )
36         <td align="center" colspan="6">
37           <xsl:if
38             test="Tränkeplan/Konzentrationsplan/Startperiode/@Länge[. = 0] or Tränkeplan/Milchanteilplan/Startperiode/@Länge[. != 0]"> <xsl:value-of select="Tränkeplan/@Tränkepreis"/> Pf / l
39             Milchtränke
40           </xsl:if>
41         </td>
42       </tr>
43     </table>
44     <br /> <b>Tränkeplanrechner -
45     </b>Markup: XML/XSLT <P>Copyright 2000
46     <a href="mailto:deininger@uni-kassel.de">Andreas Deininger</a></P>
47   </div>
48 </body>
49 </html>
50 </xsl:template>
51 </xsl:stylesheet>
  
```

Zeile 1 weist das Dokument als XML-Dokument aus, in Zeile 2 werden die verschiedenen Namensräume spezifiziert, um innerhalb des Dokuments zwischen Ausgabeanweisungen im HTML-Format (etwa Zeilen 5-23) und Anweisungen an den XSL-Prozessor, gekennzeichnet durch ein vorangestelltes "xsl:" (etwa Zeilen 3, 24) unterscheiden zu können. Die weiteren Zeilen spezifizieren zunächst das allgemeine HTML-Gerüst der Datei (Zeilen 5 ff.) und anschließend die Darstellung der Planwerte in Tabellenform (Zeilen 16 ff.). Die Zelleninhalte der Tabelle werden dann dynamisch mit den Inhalten der XML-Plandatei gefüllt. Zeile 24 etwa liefert den Wert des Attributs "Länge" der Startperiode des Mengenplans. Die Mächtigkeit der XSLT-Sprache wird in Zeile 262 ff. demonstriert. Eine if-Anweisung gestattet es, den Tränkepreis nur dann in der Tabelle mit auszugeben, wenn, wie im Falle des Milch- oder Kombiautomaten, tatsächlich Frischmilch vertränkt wird, andernfalls bleibt diese Tabellenzeile leer.

### 3.2.2 Tabellarische Datendarstellung mittels Transformation in LATEX-Format

Die Darstellung der Tränkepläne als Tabelle in einem Webbrowser bietet nur sehr eingeschränkte Gestaltungsmöglichkeiten für das Layout der Tabelle. Für eine Darstellung der Tabelle in gedruckter Form wären weitere Formatierungsmöglichkeiten wünschenswert, welche die HTML-Sprache nicht zur Verfügung stellt. Abhilfe kann die Nutzung des Satzsystems LATEX (KOPKA 1996, [9]) schaffen. Der benötigte Input für dieses Satzsystem kann problemlos von dem XSLT-Prozessor mittels der XSL-Stylesheet-Datei `plan2pdf.xml` generiert werden. Wird das von LATEX abgeleitete Programm `pdfLATEX` genutzt, so kann aus der Quelldatei auch Output in dem von der Firma Adobe entwickelten portable document format (pdf) (ADOBE SYSTEMS INC. 2000, [3]) erzeugt werden. In Tabelle 2 ist ein Ausdruck der erzeugten Tabelle mit den Planwerten zu finden.

Nachfolgend soll beispielhaft das Vorgehen bei der Erstellung dieser Tabelle vorgestellt werden. Als XSL-Prozessor soll dabei James Clark's `xt-XML-Parser` (CLARK 1999, [5]), ein vollständig in der Programmiersprache Java implementiertes Programm, verwendet werden. Der Aufruf zur Transformation könnte dann etwa wie folgt aussehen:

```
Set classpath=f:\programme\xt\xt.jar;f:\programme\jdk1.x\src\;
xt plan.xml plan2pdf.xml | pdflatex
```

Zunächst wird der Pfad der Java-Klassen zusätzlich zu den Standard-Klassen des Java-Development-Kit (SUN MICROSYSTEMS INC. [14]) um das Archiv `xt.jar` erweitert, welches die von dem `xt-Parser` benötigten Klassen enthält. Anschließend kann der Parser mit dem Kommando `xt` aufgerufen werden, wobei als Parameter das XML-Source-File `plan.xml` sowie das XSLT-Stylesheet-File `plan2pdf.xml` mit übergeben werden. Der vom `xt-Prozessor` generierte Output wird direkt dem `pdfLATEX-Programm` (TUG [15]) zugeführt, welches in Ermangelung eines mitgegebenen Dateinamens die erzeugte pdf-Datei unter dem Namen `texput.pdf` in dem Verzeichnis ablegt, aus dem heraus der Aufruf des `xt-Parsers` erfolgte.

Tab.2: Tabellarische Darstellung des Tränkeplans, Code mittels XSL generiert

Periode	Tränkepläne								
	Mengenplan			Konzentrationsplan			Milchanteilplan		
	50 Pf/l Tränkepreis			300 DM/dt MAT			Milchwert 120 g/l		
	Länge	Startwert	Endwert	Länge	Startwert	Endwert	Länge	Startwert	Endwert
	d	l	l	d	g/l	g/l	d	%	%
1	3	6	6.0	10	90	120	5	100	100
2	11		8.0	67		120	7		
3	18		8.0						
4	42		2.0						
5									
gesamt	70			90			110		

### 3.2.3 Graphische Datenausgabe mittels Transformation in PICTEX-Format

Neben der tabellarischen Darstellung, sei es im Webbrowser oder als pdf-File, kann auch die graphische Ausgabe der Pläne über die XSLT-Sprache realisiert werden. Im vorlie-

genden Beispiel wurde die Seitenbeschreibungssprache Postscript (Adobe Systems Inc. [2]) zur Darstellung der Graphik gewählt. Eine Entwicklungsumgebung zur Erzeugung von PostScript-Dateien stellt das Satzsystem LATEX (KOPKA 1996, [9]) dar, welches mittels Einbindung der Style-Files der PICTEX-Erweiterung (KOPKA 1996, [10]) zur Ausgabe von Graphiken befähigt ist. Die Konvertierung in das PostScript-Format erfolgt hierbei in einem zweistufigen Prozess: Zunächst wird von dem XSLT-Prozessor die XML-Datei mittels des Stylesheets `plan2ps.xml` transformiert, wobei als Output dieser Transformation eine Quelldatei für die Bearbeitung mittels des LATEX-Prozessors steht. Die Kompilation dieser Quelldatei `plangraph.tex` liefert eine Datei im `dvi-Format`, welche ihrerseits dann noch mittels des Programms `dvips` (ROCKICKI [13]) von Tomas Rockicki in eine PostScript-Datei umgewandelt werden muss. Diese Postscript-Datei kann dann entweder auf einem postscriptfähigen Drucker gedruckt werden oder mittels des Programms `GhostView` [1] direkt am Bildschirm betrachtet werden. Abbildung 1 zeigt die graphische Präsentation des Tränkeplans, wobei aus Platzgründen auf die Wiedergabe des Konzentrations- und Milchanteilplans verzichtet wurde.

## 4 Einbindung von XML Daten in Applikationen

Über das in Kapitel 3 erläuterte Verfahren der Transformation von XML-Daten hinaus wird sicherlich oftmals der Wunsch bestehen, vorhandene XML-Daten in bestehenden oder neu zu erstellenden Applikationen zu verwenden. Wie dies zu bewerkstelligen ist und welche Hilfsmittel hierfür einzusetzen sind, soll nachfolgend, wiederum anhand eines konkreten Beispiels, erläutert werden.

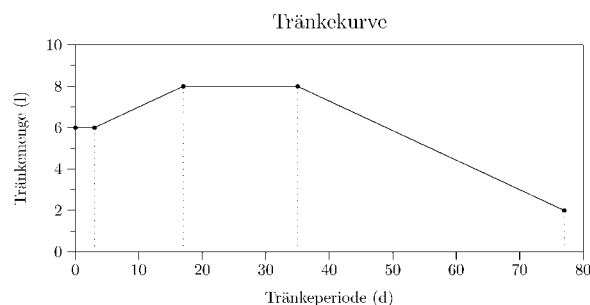


Abb.1: Graphische Darstellung des Tränkeplans, Code mittels XSL generiert

### 4.1 Das SAX-Application Programming Interface (API)

Sollen in selbst programmierten Anwendungen XML-Daten gehandhabt werden, so ist es ratsam, auf bereits vorgefertigte Routinen zur Manipulation dieser Daten zurückzugreifen. Solche Routinen werden für die verschiedensten Anwendungsbereiche in Form eines sog. Application Programming Interface (API) zur Verfügung gestellt. Im Falle von XML-Daten wurde von David Megginson ein Simple API for XML, kurz SAX benannt, entwickelt (MEGGINSON [12]). Dieses mittlerweile in der Version 2.0 verfügbare Interface ist komplett in der Programmiersprache Java imp-

lementiert und stellt insgesamt 17 Hauptklassen und 10 Helferklassen mit ihren entsprechenden Methoden zur Datenmanipulation zur Verfügung. Wie nachfolgend dargestellt, reichen für einfache XML-Anwendungen jedoch bereits wenige dieser Klassen aus, um die gewünschte Funktionalität zu erhalten.

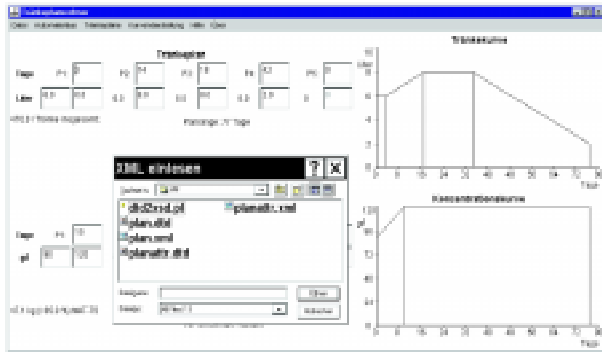


Abb.2: Tränkeplanrechner mit Dialog zum Speichern von XML-Dateien

#### 4.2 Speicherung von Tränkeplänen in XML als beispielhafter Anwendungsfall

Bei dem im Folgenden vorgestellten Programm, anhand dessen die Handhabung von XML-Daten innerhalb eigener Applikationen erläutert wird, handelt es sich um einen Tränkeplanrechner, welcher die Kalkulation und Berechnung von Tränkeplänen gestattet. Das Programm ist dabei sowohl zur Online-Nutzung im Browser (als sog. Applet) als auch als eigenständig lauffähiges Programm verfügbar und wurde detaillierter bereits von DEININGER (1998) beschrieben. Nachfolgend wird dargestellt, wie dieses Programm mittels der SAX-Schnittstelle um die Fähigkeit des Ladens und Speicherns von Tränkeplänen im XML-Format erweitert wurde. Abbildung 2 zeigt eine Bildschirmphotographie des Tränkeplanrechners mit einem geöffneten Dialogfenster zum Speichern eines Tränkeplans als XML-Datei. Die derart abgespeicherte Datei weist eine Struktur identisch zu Listing 1 auf, erweitert um die DOC-TYPE-Definition (vgl. Listing 2), so dass das XML-File als Standalone-XML-Datei weitergegeben werden kann. Umgekehrt können selbstverständlich auch bestehende, im XML-Format abgespeicherte Tränkepläne geladen werden, wobei die Planwerte dann übernommen werden und die entsprechenden Plangrößen kalkuliert und auf dem Bildschirm ausgegeben werden.

Das nachfolgend wiedergegebene Listing 5 zeigt die relevanten Programmteile, welche für das Öffnen einer bestehenden XML-Datei und das Einlesen der Planwerte programmiert wurden.

Listing 5: Java-Routine zum Einlesen von Tränkeplänen im XML-Format

```

6   import org.xml.sax.*;
7   import org.xml.sax.helpers.*;
   (...)
554 void openFile() {
555     FileDialog openxmlfd = new FileDialog(
        this, "XML einlesen", FileDialog.LOAD);
556     openxmlfd.show();

```

```

557     String xmlfileopen = openxmlfd.getFile();
558     if ( xmlfileopen == null )// Cancel
559         return;
560     try{
561         mysaxapp.einlesen(xmlfileopen);}
562     catch (java.lang.Exception e) {
563         System.out.println("Java Lang Except-
        tion!");
564     }
565     Tränkeinfo.preis.setText(String.valueOf
        (tkosten));
566     Konzentrationsinfo-
        preis.setText(String.valueOf(kkosten));
567     Milchanteillin-
        fo.preis.setText(String.valueOf(milkvalue));
568     update(true,true,true);
569 }
   (...)
1574 class mysaxapp extends DefaultHandler
1575 {
1576     int lcount=1, vcount=2, plannummer=0;
1577     public static void einlesen (String
        filename)
1578         throws Exception
1579     {
1580         XMLReader xr = new
        com.icl.saxon.aelfred.SAXDriver();
1581         mysaxapp handler = new
        mysaxapp();
1582         xr.setContentHandler(handler);
1583         xr.setErrorHandler(handler);
1584         FileReader r = new Fil-
        eReader(filename);
1585         xr.parse(new InputSource(r));
1586     }
1587     public void startElement (String uri,
        String name, String qName, Attributes atts)
1588     {
1589         if (name.equals("Tränkeplan")){
1590             Tränke.tkosten = Inte-
        ger.parseInt(atts.getValue(0));
1591         }
1592         if
        (name.equals("Konzentrationsplan")){
1593             Tränke.kkosten = Inte-
        ger.parseInt(atts.getValue(0));
1594             plannummer = 1; lcount=1;
1595             vcount=2;
1596         }
1597     }
   (...)
1622 }
1623 }

```

Da der Tränkeplanrechner vollständig in der plattform-unabhängigen Programmiersprache Java (FLANAGAN 1999) implementiert wurde, gestaltet sich die Nutzung der SAX-Klassen sehr einfach: Mit Zeile 6 und 7 werden die Haupt- und Helferklassen importiert, so dass im Programmablauf auf deren Methoden und Eigenschaften zugegriffen werden kann. In Zeile 554 bis 569 ist die Funktion `openFile` implementiert, welche aufgerufen wird, sobald der entsprechende Menüpunkt im Pulldown-Menü (Datei-Öffnen) aufgerufen wird. Dabei wird eine Dialogbox geöffnet, in welcher der Benutzer das zu öffnende File mit den XML-Daten auswählt (Zeilen 555+556). Anschließend werden die Daten dieses Files eingelesen. Hierzu dient der Aufruf der Methode `einlesen` (Zeile 561) der neu definierten Klasse `mysaxapp`, welche in den Zeilen 1574 bis 1623 spezifiziert ist. Bei dieser Klasse handelt es sich um eine von der Klasse `DefaultHandler` des SAX-API abgeleitete Klasse. Dadurch werden dieser Klasse die Methoden der übergeordneten `DefaultHandler`-Klasse vererbt. Am wichtigsten dabei ist die Methode `parse`, mit Hilfe derer das Eingelesene und

an die Methode übergebene XML-File gescannt werden kann (Zeile 1585). Da das SAX-API keinen eigenen Parser zur Verfügung stellt, erfolgt das Scannen des XML-Dokuments mit Hilfe des Ælfred-Parsers (MEGGINSON [11]), welcher als XML-Reader Objekt erzeugt und mit Hilfe des mit dem Parser mitgelieferten SAX-Treibers angesprochen wird (Zeile 1580). Um die Werte einlesen zu können, muss einzig die vererbte Funktion `startElement`, neu definiert werden (Zeile 1587-1622). Für jedes neu geparste Element wird dabei überprüft, ob dessen Name mit einem Elementnamen des Tränkeplans übereinstimmt (Zeile 1589, 1592). Gegebenenfalls wird dann der Wert dieses Elements eingelesen (Zeilen 1590, 1593). Auf diese Art und Weise werden im Laufe des Parse-Prozesses alle Planwerte des XML-Files in den entsprechenden Variablen abgespeichert. Sind alle Werte eingelesen, sind abschließend noch die Planwerte neu zu berechnen und neu auf dem Bildschirm auszugeben (Zeilen 565-568).

### 5 Schlussfolgerungen und Ausblick

Am Beispiel eines in XML codierten Tränkeplanes wurde die Leistungsfähigkeit des XML-Modells demonstriert und die sich bietenden Möglichkeiten erörtert. Hierbei wurde deutlich, dass bei der Abfassung des XML-Dokuments dem Autor über den Entwurf eigener Dokumententypen ein breiter Spielraum bei der Gestaltung des Dokumentes eingeräumt wird. Hierdurch erweist sich das XML-Sprachmodell in sehr weitem Umfang geeignet für die Codierung von Informationen, welche in landwirtschaftlichen Produktionsprozessen anfallen. Der Anwendungsbereich könnte dabei von der Festlegung und Erstellung von Viehregistern über Arbeitstagebücher bis hin zu Ackerschlagkarteien oder Flurstückskarten reichen. Würden solche Definitionen von XML-Dateien öffentlich publiziert und von der Mehrheit der landwirtschaftlichen Softwarehersteller akzeptiert, könnte damit eine programmübergreifende Verwendung der Daten ermöglicht werden. Dies würde neue Perspektiven hinsichtlich der Integration und Verzahnung der einzelnen Programme ermöglichen. Vergleichbare Ansätze in anderen Bereichen sind bereits vorhanden, etwa in Form der Mathematical Markup Language (MathML TM) (W3C 1999, [23]) oder der Synchronized Multimedia Integration Language (SMIL) (W3C 1998, [18]).

Inwieweit sich XML selbst durchsetzen wird, bleibt abzuwarten. Mit entscheidend für das weitere Schicksal dieser Sprache dürfte sein, ob die gängigen Webbrowser zukünftig XML besser unterstützen werden und inwieweit die Implementierungen zu den Spezifizierungen des WWW-Consortiums W3C kompatibel sein werden.

### 6 Literatur

- [1] Homepage for Ghostscript, Ghostview and GSview. — <http://www.cs.wisc.edu/~ghost/>.
- [2] ADOBE SYSTEMS INCORPORATED (1999): Postscript® Language Reference, Addison Wesley.
- [3] ADOBE SYSTEMS INCORPORATED (2000): PDF Reference Version 1.3, Second Edition, Addison Wesley.
- [4] BÜSCHER, W. und Käck, M. (1995): Prozeßrechnergesteuerte Tränkeautomaten zur Kälberfütterung - Arbeits- und Managementhilfe. Zeitschrift für Agrarinformatik 3: 113-116.

- [5] CLARK, J.(1999): XT: XSL Transformation engine, <http://www.jclark.com/xml/xt.html>.
- [6] DEININGER, A. (1998): Landtechnische Software online im WWW. Ein Tränkeplanrechner als beispielhafte Anwendung. Landtechnik 53: S. 330-331, <http://www.dainet.de/ktbl/lt/beitrag/lt5-98.htm>.
- [7] FLANAGAN, D. (1999): Java in a Nutshell. A Desktop Quick Reference. 3rd Edition, O'Reilly & Associates, Inc.
- [8] GOOSENS, M.; Rahtz, S. (1999): The LATEX Web Companion. Integrating TEX, HTML, and XML, Addison-Wesley.
- [9] KOPKA, H. (1996): LATEX Band 1, Einführung, 2. überarbeitete Auflage, Addison Wesley.
- [10] KOPKA, H. (1996): LATEX Band 2, Ergänzungen mit einer Einführung in MetaFont, 2. überarbeitete Auflage, Addison Wesley.
- [11] MEGGINSON, D.: Ælfred XML Parser, [http://www.opentext.com/services/content\\_management\\_services/aelfred.zip](http://www.opentext.com/services/content_management_services/aelfred.zip).
- [12] MEGGINSON, D.: SAX 2.0: The Simple API for XML, <http://www.megginson.com/SAX/>.
- [13] ROCKICKI, T.: The Official Dvips Home Page, <http://www.radical-eye.com/dvips.html>.
- [14] SUN MICROSYSTEMS, INC. (2000): Java TM 2 Platform Standard Edition Version 1.3., <http://java.sun.com/j2se/1.3/>.
- [15] TEX USERS GROUP (TUG): PDFTEX support, <http://www.tug.org/applications/pdftex/>.
- [16] WORLD WIDE WEB CONSORTIUM (W3C): Extensible Markup Language (XML) Version 1.0. W3C Recommendation, 10. Februar 1998, <http://www.w3.org/TR/REC-xml>.
- [17] WORLD WIDE WEB CONSORTIUM (W3C): Cascading Style Sheets, level 2 CSS2 Specification. W3C Recommendation, 12. Mai 1999, <http://www.w3.org/TR/REC-CSS2/>.
- [18] WORLD WIDE WEB CONSORTIUM (W3C): Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. W3C Recommendation, 15. Juni 1998, <http://www.w3.org/TR/REC-smil/>.
- [19] WORLD WIDE WEB CONSORTIUM (W3C): XSL Transformations (XSLT) Version 1.0. W3C Recommendation, 16. November 1999, <http://www.w3.org/TR/xslt>.
- [20] WORLD WIDE WEB CONSORTIUM (W3C): Cascading Style Sheets, level 1. W3C Recommendation, 17 Dec 1996, revised 11 Jan 1999, <http://www.w3.org/TR/REC-CSS1>.
- [21] WORLD WIDE WEB CONSORTIUM (W3C): HTML 4.01 Specification. W3C Recommendation, 24. Dezember 1999, <http://www.w3.org/TR/html4/>.
- [22] WORLD WIDE WEB CONSORTIUM (W3C): XML Schema Part 0:Primer. W3C Working Draft, 7. April 2000, <http://www.w3.org/TR/xml-schema-0/>.
- [23] WORLD WIDE WEB CONSORTIUM (W3C): Mathematical Markup Language (MathMLTM) 1.01 Specification. W3C Recommendation, Revision of 7 July 1999, <http://www.w3.org/TR/REC-MathML>.

### Nutzungsmöglichkeiten der Extensible Markup Language (XML) und Stylesheet Language Transformations (XSLT) für die Darstellung und Transformation landwirtschaftlicher Informationen (A. Deininger)

## Zusammenfassung

Im Februar 1998 wurde vom World Wide Web Consortium (W3C) mit der Extensible Markup Language (XML) ein neues Sprachmodell für die Codierung von Informationen zur Übertragung im Internet spezifiziert. Dieser Artikel befasst sich mit der Nutzung von XML in landwirtschaftlichen Anwendungen. Beispielhaft wird ein Tränkeplan für computergesteuerte Tränkeautomaten für Kälber in der XML-Sprache codiert und es werden Regeln für die Validierung der Daten entworfen und erläutert. Anschließend wird die Transformation der in dem XML-File gespeicherten Rohdaten in verschiedene Ausgabeformate, namentlich HTML, pdf und PostScrip, erläutert. Hierbei wird das für die Transformation verwendete Extensible Stylesheet Language Transformations- (XSLT) Sprachmodell vorgestellt und anhand eines Beispiels erläutert. Abschließend wird am konkreten Beispiel der Speicherung von Tränkeplänen in einem Tränkeplanrechner erörtert, wie XML-Daten mittels Nutzung des Simple Application Programming Interface for XML (SAX) in bestehenden oder neu zu erstellenden Anwendungsprogrammen verwendet und manipuliert werden können.

**Stichworte:** Internet, Seitenbeschreibungssprache, Extensible Markup Language (XML), Extensible Stylesheet Language Transformations (XSLT)

**Possibilities for using the Extensible Markup Language (XML) and Stylesheet Language Transformations (XSLT) for the presentation and transformation of agricultural informations** (A. Deininger)

## Summary

In February 1998 the World Wide Web Consortium (W3C) specified the extensible markup language (XML) as a new document markup language for the coding of documents transferred via the internet. This article discusses the use of XML in the field of agricultural applications. As an example, the coding of a feeding plan for automated calf feeding stations by the use of the extensible markup language is explained. Furthermore, the coding of rules which allow to validate XML data is explained and discussed. Afterwards, the transformation of XML data into different output formats, namely HTML, pdf and postscript, is illustrated. In that context, the model of extensible stylesheet language transformations (XSLT), which are used for the transformation of the xml data is introduced and exemplified. Finally, techniques for using and manipulating XML data in own applications are demonstrated. This is exemplified by showing how to open and save XML-files containing feeding plans in a feeding plan calculator, using the simple application programming interface for XML (SAX).

**Keywords:** Internet, document markup, extensible markup language (XML), extensible stylesheet language transformations (XSLT)

Dr. Andreas Deininger ist wissenschaftlicher Mitarbeiter am Fachgebiet Agrartechnik an tropischen und subtropischen Standorten der Universität Gesamthochschule Kassel, Nordbahnhofstrasse 1a, 37213 Witzenhausen. Spezialgebiete: Verfahrenstechnik der Tierproduktion, Mechanisierung der Innenwirtschaft und Prozesssteuerung in der Tierhaltung. Sie erreichen ihn unter der e-mail-Adresse [deininger@uni-kassel.de](mailto:deininger@uni-kassel.de), telefonisch unter 05542-98-1615 oder per Fax unter 05542-98-1520.