Präsentationen in XML

Thomas A. Schmitz

XML ist ein Standard im Informationsaustausch. ConTEXt enthält ausgeklügelte und mächtige Werkzeuge zur Verarbeitung der vielseitigen Auszeichnungssprache. Diese Anleitung zeigt, wie man Präsentationen in XML definieren und dann mit ConTEXt verarbeiten kann.

XML verwenden

Grundprinzipien

XML bietet einen guten Kompromiss in Sachen Lesbarkeit für Menschen und Programme. Das Format ist ziemlich einfach, seine Prinzipien können in wenigen Sätzen beschrieben werden:

- XML-Dokumente bestehen aus Inhalt und Markup; wer bereits eine Markupsprache (wie TEX oder HTML) beherrscht, sollte keine großen Schwierigkeiten damit haben.
- Als Auszeichnungssprache muss XML ein paar Zeichen reservieren, um Auszeichnungen und Text unterscheiden zu können. In XML gibt es nur drei Arten von Sonderzeichen: spitze Klammern `<` `>` begrenzen ›Tags‹ (Elemente), die Zeichen `&` und `;` begrenzen Zeichen-›Entitys‹, und die geraden Anführungszeichen `"` und `'` begrenzen Attributwerte.
- XML-Dokumente sind (zumindest im Prinzip) in sich abgeschlossen: Sie beschreiben ihre eigene Struktur, d. h. sie definieren alle erlaubten Tags und deren erlaubten Inhalt.
- XML-Dokumente zeichnen sich durch eine hierarchische Struktur aus: Es gibt genau ein Wurzelelement (›Root‹); alle Elemente werden durch öffnende und schließende Tags begrenzt.
- Jedes öffnende Tag hat ein zugehöriges schließendes; Elemente müssen in der richtigen Reihenfolge verschachtelt werden und dürfen sich nicht überschneiden.

Im Gegensatz zu vielen anderen Markupsprachen definiert XML nur diese *syntaktischen* oder *strukturellen* Regeln, keine *semantischen*: Es gibt keine vordefinierten Tags und fast keine vordefinierten Entitys. Das hat den Vorteil, dass man seine eigenen Tags definieren und sie den Anforderungen des eigenen Dokuments anpassen kann. Der Nachteil ist, dass man seine eigenen Tags definieren muss und sie den

Anforderungen des eigenen Dokuments anpassen muss. Das bedeutet, dass man sich über die Struktur seiner Informationen sorgfältig Gedanken machen sollte.

Sollte man XML verwenden?

XML bietet eine Reihe von Vorteilen, durch seine eigenen Qualitäten und durch die Art, wie es verwendet wird:

- Es ist extrem anpassungsfähig und kann für verschiedene Zwecke auf sehr unterschiedliche Weise verwendet werden. Es ist für stark strukturierte und verschachtelte Daten ebenso geeignet wie für »gemischte« Inhalte mit viel Text.
- Es ermutigt[1] die Trennung von Inhalt und Darstellung und damit das Schreiben von »sauberem« Code.
- XML kann von sehr vielen Programmen gelesen und verarbeitet werden. Fast jede Programmiersprache bietet Parser und Konverter dafür. Zusätzlich gibt es mächtige Werkzeuge für XML-basierte Workflows.
- Es gibt eine Reihe von Ausgabemöglichkeiten. Mit den richtigen Werkzeugen kann man PDFs zur Druckausgabe produzieren oder verschiedene Formate für das Web.
- XML bietet einen guten Kompromiss zwischen Lesbarkeit und Informationsdichte.
- Man könnte argumentieren, dass die etwas langatmige Syntax die Lesbarkeit erhöhe: Das explizite Öffnen und Schließen von Tags ist leichter verständlich als z. B. die geschweiften Klammern von TEX, bei denen man nicht immer gleich sieht, welche Gruppe sie schließen.

Das sind starke Argumente, aber XML ist keine Wunderwaffe und nicht per se ›besser‹ als TEX-Syntax.

Es gibt eine Reihe von Nachteilen, die man nicht vernachlässigen kann, wenn man XML mit ConTEXt verwendet:

- Man muss eine weitere Syntax lernen.
- XML fügt eine weitere Ebene zwischen Quelle und Ausgabe ein; man muss XML-Elemente auf die richtigen ConTEXt-Befehle abbilden.
- Das Schreiben von Makros wird verkompliziert: Man muss immer zweimal über die Syntax und Regeln nachdenken, zuerst, wie man sein Vorhaben in XML strukturiert, und dann, wie man die XML-Struktur in ConTEXt umsetzt.
- Es kann unheimlich frustrieren, wenn etwas nicht funktioniert: Die zusätzliche Übersetzung von XML nach TEX macht die Fehlersuche schwierig und ist oft eine Problemquelle.

[1] Ich spreche von ›ermutigen‹, weil man auch in XML fürchterliche Dokumente erstellen kann, die optisches Erscheinungsbild und Struktur vermischen.

Alles in allem empfehle ich, sich die Sache genau zu überlegen. Wer nur einzelne Dokumente verfasst, die sich in Struktur, Stil und Format unterscheiden, oder wer nur die PDF-Ausgabe braucht, sollte XML lieber vermeiden.

Es ist aber nützlich, wenn

- die Struktur der Dokumente sich vorhersehbar wiederholt,
- sich diese Struktur in wiederholten Stilelementen äußert,
- man verschiedene Ausgabeformate aus einer Quelle erzeugen möchte,
- man den Inhalt eventuell auf andere Weise oder mit anderen Anwendungen verarbeiten möchte.

XML mit ConTEXt verarbeiten

Der Editor

Die erste Frage, die viele Leute stellen, wenn sie von XML hören, ist: Welchen Editor sollte ich dafür verwenden? Darauf gibt es keine allgemeingültige Antwort, sie hängt von verschiedenen Faktoren ab: dem Betriebssystem, der Art der XML-Dokumente (enthalten sie hauptsächlich Text, oder sind es eher Datenbanken?) und der bevorzugten Arbeitsweise (möchte man die Tags und Attribute sehen oder sollen sie irgendwie interpretiert werden?).

Es gibt ein paar Spezialwerkzeuge zur Bearbeitung von XML, das bekannteste ist Oxygen (`www.oxygenxml.com`), ein kommerzielles Programm, das für die meisten Plattformen verfügbar ist. Die meisten Universaleditoren bieten zumindest eine Syntaxhervorhebung.

Meine bevorzugte Lösung ist ziemlich einfach: Emacs ist mein Editor der Wahl, und es gibt dafür einen guten XML-Bearbeitungsmodus[2] mit Syntaxhervorhebung, Einrückung, Validierung und sogar automatischer Vervollständigung, wenn man ein Schema hinterlegt hat.

Man sollte einfach verschiedene Editoren ausprobieren, bis man etwas gefunden hat, mit dem man bequem arbeiten kann.

XML in ConTEXt

XML mit ConTEXt zu verarbeiten ist schon lange möglich. Im alten ConTEXt MkII war das noch ziemlich kompliziert, aber mit ConTEXt MkIV, das auf LuaTEX basiert, hat sich die Unterstützung dramatisch verbessert, und das ist auch schon mehr als zehn Jahre her. Bei ConTEXt LMTX soll es noch besser werden.

[2] `www.thaiopensource.com/nxml-mode`

Die XML-Datei wird in Lua geparst und im Speicher gehalten. Dadurch hat man jederzeit Zugriff auf jedes Element. Das macht es einfach, Teile der Eingabe mehrfach zu verwenden, auszuwählen, zu filtern, zu verändern oder zu prüfen. So kann man Makros schreiben, die das *n*-te Element des Typs ‹x› verarbeiten, oder ein Element des Typs ‹x› nur, wenn es den Text *z* enthält; oder man kann den Inhalt eines Elements mit verschiedenen Werten vergleichen und in Abhängigkeit davon etwas damit anstellen.

Um XML mit ConTEXt zu verarbeiten, braucht man neben der XML-Datei eine Stilvorlage (Environment), die an einem Ort liegen muss, wo ConTEXt sie findet (z. B. im gleichen Verzeichnis wie die XML-Datei). Sie enthält zwei Arten von Anweisungen:

- Einstellungen für jedes XML-Element, das man verarbeiten möchte, und
- Einstellungen für Layout und Aussehen des Dokuments – so wie bei jedem anderen ConTEXt-Projekt.

Wenn wir eine Datei `document.xml` und eine Stilvorlage `style.tex` haben, erzeugt der Aufruf:

```
context --environment=style document.xml
```

eine Datei namens `document.pdf`.

Präsentationen in XML

Um zu lernen, wie man XML in ConTEXt verarbeitet, gehen wir von einem realistischen Beispiel aus. Ich stelle den Arbeitsablauf vor, den ich nun seit 2017 an meiner Universität verwende:

Während des Semesters halte ich jede Woche eine 90-minütige Vorlesung. Ich schreibe den Text der Vorlesungen in XML, und ich habe mich entschieden, dass meine Quelldatei sowohl den Text meiner Vortragsfolien als auch mein Skript enthalten soll. Das hat den Vorteil, dass ich den Text mit anderen Anwendungen verarbeiten kann, um z. B. eine Website für den Kurs zu erzeugen, und dass ich meine Folien, Handouts für die Studierenden und das Skript für mich selbst aus den gleichen Quellen erzeugen kann, einfach mit unterschiedlichen Stilvorlagen.

Für die Präsentation verwende ich das `simpleslides`-Modul[3], das Aditya Mahajan und ich vor einigen Jahren entwickelt haben. Damit ist es einfach, Präsentationsfolien mit einer ansprechenden Gestaltung zu produzieren, ohne dass man sich allzusehr um Einstellungen und Layout kümmern muss; die ›Schwerarbeit‹ wird von dem Modul übernommen.

[3] Die Installation ist im Wiki beschrieben: `https://wiki.contextgarden.net/Modules`

Die Struktur

Wie wir bereits gesehen haben, sind wir frei, die Elemente unserer XML-Datei selbst zu definieren. Was wäre dann eine gute Struktur für meinen Kurs? Die Datei soll die Präsentationen für ein ganzes Semester enthalten. Fangen wir mit dem Wurzelelement an, das wir etwas einfallslos ‹document› nennen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE presentation>
<document>

</document>
```

Dieses Wurzelelement bekommt Kindelemente namens ‹presentation› für jede Vorlesung. Um jeder Präsentation einen eindeutigen Bezeichner zu geben, ergänzen wir ein Attribut day mit dem Datum der Vorlesung. Damit sieht die Struktur so aus:

```
<document>

  <presentation day="21_09_21">

  </presentation>

  <presentation day="21_09_28">

  </presentation>

</document>
```

Jede Präsentation hat ein paar Metainformationen wie Titel und Datum, die auf der Titelfolie erscheinen sollen. Und natürlich hat sie einen Inhalt: die einzelnen Folien, aus denen die Präsentation besteht. Daraus ergibt sich die Struktur für unsere erste Präsentation:

```
<document>

  <presentation day="21_09_21">
    <title>Beispiel einer Präsentation in XML</title>
    <author>A. U. Thor</author>
    <date>21. 9. 2021</date>
    <content>
      <slide>

      </slide>
      <slide>
```

```
      ‹/slide›
    ‹/content›
  ‹/presentation›

‹/document›
```

Jede einzelne Folie hat drei Teile: einen Titel, einen Inhalt, der auf der Folie zu sehen ist, und einen Kommentar, der nicht auf die Folie soll, sondern ins Vorlesungsskript.

Der Inhalt der Folien kann natürlich irgendetwas sein, was man da haben möchte. Der Einfachheit halber beschränken wir uns hier auf ein paar häufige Standardelemente: normalen Text, eine Aufzählung, ein Bild und schließlich eine Tabelle. Wir verwenden diese hier nur einzeln, damit die Beispiele nicht zu komplex werden.

Aber zunächst beschäftigen wir uns damit, wie man eine Stildatei zur Verarbeitung dieser XML-Elemente verfasst:

Die Stilvorlage

Unsere Datei `style.tex` beginnt mit folgenden Zeilen:

```
\startxmlsetups xml:presentationsetups
  \xmlsetsetup{#1}{*}{-}
  \xmlsetsetup{#1}{document|
                   presentation|
                   title|
                   author|
                   date|
                   content|
                   slide|
                   slidetitle|
                   slidecontent}
              {xml:*}
\stopxmlsetups

\xmlregistersetup
  {xml:presentationsetups}
```

Wir definieren und registrieren ein ›Setup‹ für unsere XMLVerarbeitungsanweisungen namens `presentationsetups`. Unsere Datei enthält zwar den Text für Folien *und* Skript, aber für die Präsentation wollen wir nur die Inhalte der Folien verarbeiten. Dazu müssen wir genau definieren, welche Elemente verwendet werden sollen. Zuerst weisen wir ConTEXt an, alle Elemente fallenzulassen, das macht die Anweisung `\xmlsetsetup{#1}{*}{-}`. Danach listen wir die Elemente auf, die wir verarbeiten

wollen. Später müssen wir jedes neue Element zu dieser Liste hinzufügen. Die Liste enthält jetzt also alle bisher bekannten Elemente außer ‹slidecomment›, dessen Inhalt wir ja nicht auf den Folien haben wollen.

Die Anweisung für das Wurzelelement ‹document› ist einfach: ConTEXt soll den Inhalt einfach zur weiteren Verarbeitung ›spülen‹ (flush):

```
\startxmlsetups xml:document
  \xmlflush{#1}
\stopxmlsetups
```

Dies ist der erste und wichtigste Befehl, den man zur Verarbeitung von XML lernen muss: `\xmlflush{#1}` bedeutet »nimm den Inhalt des aktuellen Elements und verarbeite ihn«. Unter ›verarbeiten‹ verstehen wir dabei: Wenn es Text ist, setze ihn; wenn weitere XML-Elemente enthalten sind, verarbeite sie laut den zugehörigen Anweisungen.

Was machen wir mit den einzelnen Präsentationen? In unserem Beispiel sind sie mit einem Datum bezeichnet, und wir möchten eine einzelne Präsentation für jede Vorlesung bekommen. Wie erreichen wir das? Ich schlage vor, wir verwenden das Attribut `day` als Namen für einen ConTEXt-Modus:

```
\startxmlsetups xml:presentation
  \startmode[\xmlatt{#1}{day}]
  \setupTitle[
    title={\xmltext{#1}{title}},
    author={\xmltext{#1}{author}},
    date={\xmltext{#1}{date}}]
  \placeTitle
  \xmltext{#1}{content}
  \page
  \stopmode
\stopxmlsetups
```

Zuerst ziehen wir den Wert aus dem `day`-Attribut (das macht `\xmlatt{#1}{day}` und starten einen `mode` damit. Für unsere Präsentation mit dem Attributwert `"21_09_21"` definiert das einen Modus dieses Namens, den wir dann auf der Kommandozeile verwenden können, um nur die Präsentation dieses einen Tages zu erzeugen:

```
context --environment=style --mode=21_09_21 document.xml
```

Dann sehen wir uns die Kindelemente unseres ‹presentation›-Elements an:

`\xmltext{#1}{title}` holt den Inhalt des ‹title›-Elements, der dann dem Befehl `\setupTitle` aus dem `simpleslides`-Modul als Parameter übergeben wird. Auf diese Weise werden Titel, Autor und Datum verarbeitet; der Befehl `\placetitle` erzeugt un-

sere Titelfolie. Schließlich überreichen wir das ‹content›-Element der Satzmaschine zur weiteren Behandlung.

Das war die allgemeine Struktur unserer Präsentation. Betrachten wir nun die einzelnen Folien. Die erste bekommt nur einen Titel und etwas Text. So könnte das XML dazu aussehen:

```
‹slide›
  ‹slidetitle›Grundlagen‹/slidetitle›
  ‹slidecontent›
    ConTeXt ist eine Markupsprache und ein Verarbeitungssystem für Dokumente,
    basierend auf dem Sastzsystem TeX. Es wurde mit dem gleichen Ziel der
    allgemeinen Verwendbarkeit entwickelt wie LaTeX, um einen vereinfachten
    Zugang zum hochqualitativen Textsatz zu ermöglichen, den TeX bietet.
    Während LaTeX aber die Schreibenden von Layout und typografischen Details
    fernhält, nimmt ConTeXt den entgegengesetzten Weg, indem es strukturierte
    Schnittstellen zu Typographie, Hintergründen, Hyperlinks, Darstellung,
    Farben und bedingter Ausführung anbietet.
  ‹/slidecontent›
  ‹slidecomment›
    Hier stehen ein paar Kommentare zu dieser Folie.
    Sie erscheinen nicht in der Präsentation, sondern
    nur im Vortragsskript.
  ‹/slidecomment›
‹/slide›
```

Das ist nicht besonders kompliziert; wir müssen nur Regeln schreiben, die den Inhalt der Elemente ‹slidetitle› und ‹slidecontent› an ConTEXt übergeben:

```
\startxmlsetups xml:slide
  \xmldoiftext{#1}{/slidetitle}{%
  \SlideTitle{
    \xmltext{#1}{slidetitle}%
  }}
  \start
    \xmltext{#1}{slidecontent}
    \par
  \stop
\stopxmlsetups
```

In der ersten Zeile des Setups prüfen wir, ob das Element ‹slidetitle› überhaupt einen Inhalt hat, schließlich haben nicht alle Folien einen Titel; wenn ja, dann wird er mit dem Befehl `\SlideTitle` aus dem `simpleslides`-Modul gesetzt.

Der Inhalt der Folie steht in einer `\start`...`\stop`-Umgebung, so dass alle Schriftveränderungen innerhalb der Folie sich nicht darüber hinaus auswirken.

Der Inhalt der Folie wird dann wiederum einfach an ConTEXt zum Setzen weitergegeben. Für unsere einfache Folie reicht das.

Als nächstes kommt eine Folie mit einer Aufzählung. Hier ist das XML:

```
<slide>
  <slidetitle>Eine Aufzählung</slidetitle>
  <slidecontent>
    <numberedlist>
      <item>Erster Punkt</item>
      <item>Zweiter Punkt</item>
      <item>Dritter Punkt</item>
      <item>Vierter Punkt</item>
    </numberedlist>
  </slidecontent>
  <slidecomment>
    Mehr Anmerkungen.
  </slidecomment>
</slide>
```

Für diesen Fall müssen wir die Einstellungen für zwei neue XML-Elemente definieren, `<numberedlist>` und `<item>` – und nicht vergessen, diese Namen in die Elemente-Liste am Anfang der Stilvorlage einzutragen.

Für die Aufzählung verwenden wir einfach eine `itemize`-Umgebung:

```
\startxmlsetups xml:numberedlist
  \startitemize[n]
    \xmlflush{#1}
  \stopitemize
\stopxmlsetups

\startxmlsetups xml:item
  \startitem
    \ignorespaces\xmlflush{#1}
  \stopitem
\stopxmlsetups
```

Zuerst geben wir an, dass ConTEXt den Inhalt des Elements `<numberedlist>` innerhalb einer `itemize[n]`-Umgebung »flushen« soll, und dann jedes einzelne `<item>` in einer `item`-Umgebung.

Unsere nächste Folie soll ein Bild zeigen. Das `simpleslides`-Modul enthält bereits Code, um Bilder ansprechend einzubinden, also müssen wir das nur noch in XML abbilden und an ConTEXt weitergeben. So könnte das aussehen:

```
<slide>
  <slidecontent>
    <includeimage type="vertical"
    resource="cow" height="0.4">
      Bildunterschrift
    </includeimage>
  </slidecontent>
  <slidecomment>
    Muh!
  </slidecomment>
</slide>
```

Und hier ist das Setup für die XML-Struktur:

```
\startxmlsetups xml:includeimage
  \IncludePicture[\xmlatt{#1}{type}]
    [\xmlatt{#1}{resource}]
    [height=\xmlatt{#1}{height}\textheight]
  {\xmlflush{#1}}
\stopxmlsetups
```

Wir erinnern uns: `\xmlatt{#1}{type}` bedeutet »der Wert des Attributs `type` des aktuellen XML-Elements«. Ein XML-Element kann mehrere Attribute haben, und mit diesem ConTEXt-Befehl können wir sie abfragen. Oh, und nicht vergessen, das neue Element `<includeimage>` in die Liste am Anfang der Stilvorlage einzutragen!

Schließlich hätten wir gerne eine Tabelle auf einer unserer Folien. Hierfür benutzen wir »Extreme Tables«, obwohl »Natural Tables« (alias »HTML-Tabellen«) ebensogut funktionieren würden. Um es ein bisschen anspruchsvoller zu machen, verwenden wir auch spalten- oder zeilenübergreifende Zellen.

So drücken wir das in XML aus:

```
<slide>
  <slidetitle>EIne Tabelle</slidetitle>
  <slidecontent>
  <table>
    <tablerow>
      <tablecell>Habt</tablecell>
      <tablecell>ihr</tablecell>
      <tablecell>schon</tablecell>
      <tablecell>gesehen,</tablecell>
    </tablerow>
    <tablerow>
    <tablecell>was</tablecell>
```

```
    <tablecell nx="2" ny="2">
      &CONTEXT;
    </tablecell>
    <tablecell>für</tablecell>
   </tablerow>
   <tablerow>
     <tablecell>all</tablecell>
     <tablecell>unsere</tablecell>
   </tablerow>
   <tablerow>
     <tablecell>wunderbaren</tablecell>
     <tablecell>Dokumente</tablecell>
     <tablecell>tun</tablecell>
     <tablecell>kann?</tablecell>
   </tablerow>
  </table>
  </slidecontent>
  <slidecomment>
    Even more notes.
  </slidecomment>
</slide>
```

Die Abbildung auf ConTEXt-Befehle ist ziemlich simpel:

```
\startxmlsetups xml:table
  \placefigure[here,force]{none}
   {\startembeddedxtable
    \xmlflush{#1}
  \stopembeddedxtable}
\stopxmlsetups

\startxmlsetups xml:tablerow
  \startxrow
    \xmlflush{#1}
  \stopxrow
\stopxmlsetups

\startxmlsetups xml:tablecell
  \startxcell[
    nx=\xmlattdef{#1}{nx}{1},
    ny=\xmlattdef{#1}{ny}{1},
    align=middle,
    top=\vss,
    bottom=\vss]
```

```
    \xmlflush{#1}
  \stopxcell
\stopxmlsetups
```

Hier sehen wir eine andere Methode, um XML-Attribute zu verarbeiten: `\xmlattdef ↪{\#1}{nx}{1}` bedeutet: »der Wert des Attributs `nx` des aktuellen Elements; wenn es das nicht gibt, nimm den Vorgabewert ›1‹«.

Außerdem haben wir eine XML-Entity verwendet: Wir wollen das Logo ConTEXt ordentlich gesetzt haben, also haben wir es in XML als `&CONTEXT;` ausgedrückt. Dafür brauchen wir eine Definition in unserer Stildatei:

```
\xmltexentity{CONTEXT}{\ConTeXt}
```

Fast geschafft!

Unsere Stilvorlage muss nun das Aussehen unseres Dokuments definieren. Wir machen es uns einfach und überlassen das Meiste dem `simpleslides`-Modul:

```
\usemodule[simpleslides][
  style=BigNumber,
  font=Helvetica,
  size=17pt]
```

Wenn wir unser XML nun mit dem bekannten Aufruf compilieren:

```
context --environment=style --mode=11_09_21 document.xml
```

bekommen wir eine Ausgabe, die aussieht wie Abb. 1 auf S. 13.

Das Skript

Das waren soweit die Einstellungen, um eine PDF-Präsentation aus unserer XML-Datei zu machen. Wir betrachten nun die Möglichkeiten, aus den gleichen Daten eine andere Ausgabe zu erzeugen.

Unsere erste Übung ist das Vorlesungsskript. Dabei wollen wir gewissermaßen das Gegenteil der Präsentation erreichen: Wir wollen nur den Inhalt der `‹slidecomment›`-Elemente und alles andere ignorieren. Nachdem wir (bisher noch) keine besonderen Elemente im Kommentarbereich verwenden, müssen wir nur dafür sorgen, dass der Inhalt gesetzt wird.

Als auffälliges Element fügen wir einen Folienzähler ein, damit wir nicht den Überblick verlieren, wo in unserer Präsentation wir uns gerade befinden.

Nachdem wir das Grundprinzip der XML-Verarbeitung verstanden haben, können wir uns die zweite Stilvorlage einfach so ansehen:

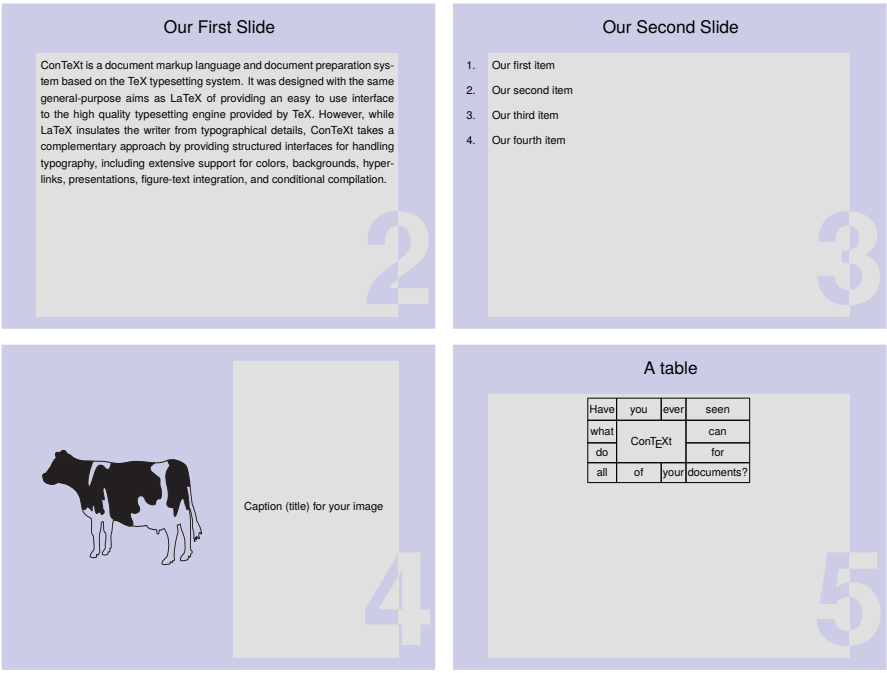

Abb. 1: Ausgabe unserer Beispielpräsentation (englische Version)

```
\startxmlsetups xml:manuscriptsetups
\xmlsetsetup{#1}{*}{-}
\xmlsetsetup{#1}{document|
                 presentation|
                 content|
                 slide|
                 slidecomment}{xml:*}
\stopxmlsetups

\xmlregistersetup{xml:manuscriptsetups}

\startxmlsetups xml:document
  \xmlflush{#1}
\stopxmlsetups

\startxmlsetups xml:content
  \xmlflush{#1}
```

```
\stopxmlsetups

\startxmlsetups xml:presentation
  \startsection[
    title={\xmltext{#1}{date}: \hfill
    \xmltext{#1}{title}},
    bookmark={\xmltext{#1}{date}:\xmltext{#1}{title}}]
    \xmlflush{#1}
  \stopxmlsetups

\startxmlsetups xml:slide
  \NewSlide \xmlflush{#1}
\stopxmlsetups

\startxmlsetups xml:slidecomment
  \xmlflush{#1}\par
\stopxmlsetups

\definecounter [SlideNumber] [way=bytext,prefix=no]

\setuplayout[
  marking=off,
  width=fit,
  height=fit,
  header=0.6cm,
  footer=0cm]

\setuphead[section][
  style=normal,
  number=yes,
  after={\resetcounter[SlideNumber]},
  expansion=yes,
  page=yes]

\setuppapersize[A6,landscape][A6,landscape]

\define\NewSlide%
  {\incrementcounter[SlideNumber]%
  \color[red]{[{\rawcounter[SlideNumber]}]}}

\setupinteraction[
  state=start,
  title={XML-Präsentationen},
  author={A. U. Thor}]
```

```
\placebookmarks[section][all]

\setupuserpagenumber[state=start,way=bysection]

\setupheadertexts[{\getmarking[sectionnumber]} – \pagenumber ]
```

Das meiste dürfte inzwischen ziemlich offensichtlich sein: Wir ›flushen‹ die Elemente, die wir gesetzt haben wollen. In diesem Fall möchten wir ein PDF für den gesamten Kurs, mit den einzelnen Präsentationen als Abschnitten (section). Diese Abschnitte beginnen auf einer neuen Seite und verwenden das ‹title›-Element als Titel. Wir fügen außerdem Lesezeichen (bookmarks) in unser PDF ein, damit sich die einzelnen Vorlesungen leichter finden lassen.

Wir haben einen Zähler `\SlideNumber`, der für jede Folie inkrementiert und in rot ausgegeben wird. Die Kopfzeile enthält die Nummer der Vorlesung im Kurs und die Seitenzahl innerhalb der Präsentation.

Der Rest des Codes dient der Optik des Skriptes, das im A6-Querformat ausgegeben wird – entweder zum Ausdruck auf Karteikarten oder zur Anzeige auf einem Tablet.

Handouts

Ursprünglich habe ich meine Folien genau so zum Download bereitgestellt, wie ich sie im Hörsaal gezeigt habe (siehe Abb. 1). Die Studierenden beschwerten sich aber zu Recht, dass dieses Format nicht zum Ausdrucken geeignet war. Also habe ich ein Environment definiert, das den Inhalt der Folien ohne farbigen Hintergrund ausgibt und jeweils vier Folien untereinander auf der linken Seite einer A4-Seite abbildet, damit auf der rechten Seite Platz für Notizen bleibt. So können die Studierenden ihre Anmerkungen auf den ausgedruckten Handouts direkt zu den Folien schreiben.

Die meisten Einstellungen für die Elemente sind die gleichen wie für die Folien. Da wir aber ohne das `simpleslides`-Modul arbeiten, musste ich ein paar Definitionen aus dem Modul kopieren, z. B. für die Bildplatzierung. Aber damit will ich niemanden langweilen. Das Wichtigste ist die Anordnung der Folien auf der Seite, die wir mit folgendem Code erreichen:

```
\setuppapersize[A7,landscape][A4]

\setuppaper[
  topspace=3mm,
  backspace=1.5mm,
  bottomspace=0mm,
  dx=0mm,
```

```
  dy=0mm,
  nx=1,
  ny=4]

\setuparranging[XY]
```

Damit bekommen wir die gewünschte Anordnung für den Ausdruck.

Fazit

Es dauert eine Weile, einen Workflow aufzusetzen, mit dem man Präsentationen aus XML erzeugen kann, und ich habe mehrere Anläufe gebraucht, um zu einer Form zu kommen, die mir hoffentlich auch in Zukunft dienen wird.

Wenn man alles beisammen hat, werden die Vorteile des XML-Formats deutlich: Es ist bequem, sowohl die Folien als auch die Notizen in einer Datei zu haben und die Daten damit einfach wiederverwenden zu können. Das Material steht damit auch für andere Ausgabeformate zur Verfügung; es wäre nicht schwierig, die Folien mit Hilfe einer XSL-Transformation ins Netz zu bringen. Und nachdem Inhalt und Form sauber getrennt sind, sollten die Daten auch noch verwendet werden können, wenn sich die zugrundeliegenden Mechanismen ändern.

(Der Artikel erschien auf englisch im Context Group Journal zum ConTEXt meeting 2019; für DTK übersetzt und bearbeitet von Henning Hraban Ramm.)