2 WordML

Die *Wordprocessing Markup Language* (kurz WordML) bildet die Schnittstelle zwischen der XML-Welt und dem bisherigen proprietären Doc-Format von MS Word.

Schnittstelle

In WordML lassen sich, wie wir bereits im letzten Kapitel gezeigt haben, alle Word-Dokumente abspeichern und ebenso wieder öffnen. Da sie anders als das Doc-Format nicht in einem Binärcode abgespeichert werden, sind WordML-Dokumente mit jedem Texteditor lesbar. Daraus ergeben sich neue Möglichkeiten zur Generierung von Word-Dokumenten und zum automatischen »Taggen« von Inhalten. Beides wird in den folgenden Kapiteln thematisiert und erläutert. Dieses Kapitel soll hierfür die Grundlage bilden, weshalb wir uns mit dem Aufbau und den Inhalten solcher WordML-Dokumente genauer beschäftigen werden.

Vgl. Kapitel 7, S. 131

Die Struktur von WordML ist in einem Schema definiert und ist Teil einer ganzen Schemafamilie von Office 2003. Sie kann kostenlos zusammen mit einer weiterführenden Dokumentation unter folgender Adresse heruntergeladen werden: www.microsoft.com/office/xml.

Schemas

Nach der Installation befinden sich die Schemas in verschiedenen Ordnern. Auf die Namen dieser Schemas wird im Folgenden immer wieder verwiesen werden. Hinzuweisen sei hier vor allem auch auf die Referenz aller Elemente und Attribute, die einem oft weiterhelfen kann. Die Einführung, die die Dokumentation bietet, ist hingegen eher knapp gehalten und oftmals wenig hilfreich.

WordML und die anderen Office-Schemas enthalten alle Strukturen, die ein Office-Dokument aufweisen kann. Ein Word-Dokument kann bekanntermaßen auch Excel-Tabellen umfassen, was dazu führen kann, dass sehr viele komplexe Strukturen innerhalb eines Word-Dokumentes vorkommen können.

In diesem Buch konzentrieren wir uns allerdings auf die WordML-Elemente und deren Verwendung. Da es sehr viele Elemente gibt und ein WordML-Dokument, das unter Verwendung von `Speichern unter` erstellt wurde, sehr viele Metainformationen enthält (u.a. Name des Autors, Erstellungsdatum, Formatvorlagen), ist es aus didaktischer Sicht besser, sich den einzelnen Aspekten näher zu widmen. In den nächsten Abschnitten werden nach einer Einführung in die Grundstruktur eines WordML-Do-

kumentes verschiedene, häufig vorkommende Strukturen, die der Darstellung dienen, wie Tabellen, Absätze etc., im Detail erläutert.

2.1 Der Grundaufbau

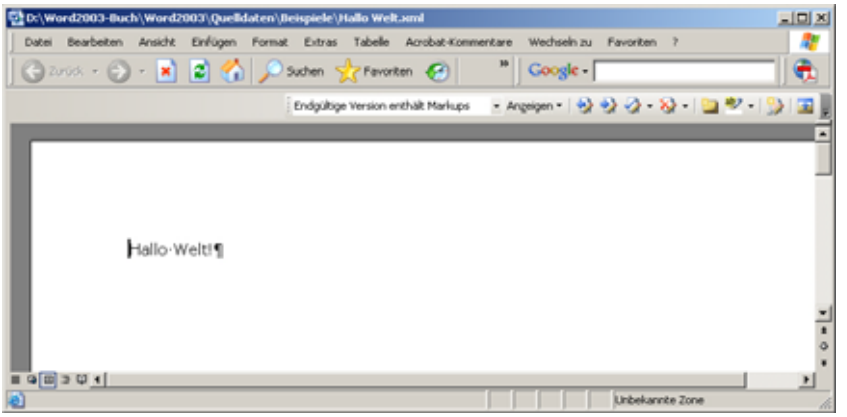
Das folgende *Hallo Welt*-Beispiel wurde manuell erzeugt und stellt die kürzeste Variante eines WordML-Dokumentes mit einem Absatz dar:

```
<?xml version="1.0"?>
<?mso-application progid="Word.Document"?>                    ❶
<w:wordDocument xmlns:w="http://schemas.microsoft.com/office/  ❷
    word/2003/wordml">
   <w:body>                                                   ❸
      <w:p>                                                   ❹
         <w:r>                                                ❺
            <w:t>Hallo Welt!</w:t>                            ❻
         </w:r>
      </w:p>
   </w:body>
</w:wordDocument>
```

Word 2003 XML Viewer

❶ Die PI (Processing Instruction) dient dem Betriebssystem dazu, das Dokument als WordML-Dokument zu erkennen. Statt eines XML-Editors oder des Internet Explorer wird nun Word 2003 gestartet. Außerdem ermöglicht die PI dem Internet Explorer,[1] das WordML-Dokument als Word-Dokument gerendert darzustellen, statt es wie ein beliebiges XML-Dokument in der Hierarchiedarstellung anzuzeigen.

Abb. 2-1
Anzeige im Internet Explorer



❷ Das Wurzelelement mit der Namensraumdeklaration für WordML.

❸ `<w:body>` enthält den Inhalt eines WordML-Dokumentes.

[1] Für die Anzeige von WordML-Dokumenten mit dem Internet Explorer müssen Sie, falls Word 2003 nicht vorhanden ist, einen Viewer installieren. Diesen können Sie kostenlos auf der englischsprachigen Seite von Microsoft finden. Sie müssen hierzu die Suchbegriffe »Word 2003 XML Viewer« eingeben.

❹ <w:p> (*paragraph*) ist das Absatzelement in WordML.
❺ <w:r> steht für *Run-Text*, also den Fließtext eines Absatzes.
❻ <w:t> (*text*) enthält den eigentlichen textlichen Inhalt.

Da in diesem Beispiel keine Formatvorlagen angegeben wurden, wird der Absatz *HalloWelt!* in der Standardeinstellung mit der Standardschrift, Standardschriftgröße etc. angezeigt.

HalloWelt!

Zum Vergleich folgt ein *Hallo Welt*-Beispiel, das nicht manuell erstellt wurde, sondern bei dem der Text in ein leeres Word-Dokument eingegeben und mit der Speichern unter-Option als XML (WordML) abgespeichert wurde.

Vgl. Kapitel 1.4, S. 16

Da das WordML-Dokument äußerst lang ist, werden hier nur Auszüge daraus abgedruckt.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?mso-application progid="Word.Document"?>
<w:wordDocument xmlns:w="http://schemas.microsoft.com/office/word/      ❶
    2003/wordml" xmlns:v="urn:schemas-microsoft-com:vml"
    xmlns:w10="urn:schemas-microsoft-com:office:word" xmlns:sl="http://
    schemas.microsoft.com/schemaLibrary/2003/core" xmlns:aml="http://
    schemas.microsoft.com/aml/2001/core" xmlns:wx="http://
    schemas.microsoft.com/office/word/2003/auxHint" xmlns:o="urn:schemas-
    microsoft-com:office:office" xmlns:dt="uuid:C2F41010-65B3-11d1-
    A29F-00AA00C14882" w:macrosPresent="no" w:embeddedObjPresent="no"
    w:ocxPresent="no" xml:space="preserve">
   <o:DocumentProperties>                                               ❷
      <o:Title>Hallo Welt</o:Title>
      <o:Author>Montero</o:Author>
      <o:LastAuthor>Montero</o:LastAuthor>
      <o:Revision>3</o:Revision>
      <o:TotalTime>0</o:TotalTime>
      <o:Created>2005-12-20T14:31:00Z</o:Created>
      <o:LastSaved>2005-12-20T14:31:00Z</o:LastSaved>
      <o:Pages>1</o:Pages>
      <o:Words>1</o:Words>
      <o:Characters>11</o:Characters>
      <o:Lines>1</o:Lines>
      <o:Paragraphs>1</o:Paragraphs>
      <o:CharactersWithSpaces>11</o:CharactersWithSpaces>
      <o:Version>11.6359</o:Version>
   </o:DocumentProperties>
   <w:fonts>                                                            ❸
      <w:defaultFonts w:ascii="Times New Roman" w:fareast="Times New Ro-
          man" w:h-ansi="Times New Roman" w:cs="Times New Roman"/>
      <w:font w:name="Verdana">
         <w:panose-1 w:val="020B0604030504040204"/>
         <w:charset w:val="00"/>
         <w:family w:val="Swiss"/>
         <w:pitch w:val="variable"/>
```

```
      <w:sig w:usb-0="20000287" w:usb-1="00000000" w:usb-2="00000000"
          w:usb-3="00000000" w:csb-0="0000019F" w:csb-1="00000000"/>
   </w:font>
</w:fonts>
<w:lists>                                                              ❹
   <w:listDef w:listDefId="0">
      <w:lsid w:val="376C6A7C"/>
      <w:plt w:val="HybridMultilevel"/>
      <w:tmpl w:val="D6A4D526"/>
      <w:lvl w:ilvl="0" w:tplc="5D06413E">
         <w:start w:val="1"/>
         <w:pStyle w:val="Style3"/>
         <w:lvlText w:val="%1."/>
         <w:lvlJc w:val="left"/>
         <w:pPr>
            <w:tabs>
               <w:tab w:val="list" w:pos="720"/>
            </w:tabs>
            <w:ind w:left="720" w:hanging="360"/>
         </w:pPr>
      </w:lvl>
      <w:lvl w:ilvl="1" w:tplc="04070019" w:tentative="on">
         <w:start w:val="1"/>
   ...
   </w:list>
</w:lists>
<w:styles>                                                             ❺
   <w:versionOfBuiltInStylenames w:val="4"/>
   <w:latentStyles w:defLockedState="off" w:latentStyleCount="156"/>
   <w:style w:type="paragraph" w:default="on" w:styleId="Normal">
      <w:name w:val="Normal"/>
      <w:autoRedefine/>
      <w:rsid w:val="00881D85"/>
      <w:pPr>
         <w:jc w:val="both"/>
      </w:pPr>
      <w:rPr>
         <w:rFonts w:ascii="Verdana" w:h-ansi="Verdana"/>
         <wx:font wx:val="Verdana"/>
         <w:sz w:val="22"/>
         <w:sz-cs w:val="24"/>
         <w:lang w:val="DE" w:fareast="DE" w:bidi="AR-SA"/>
      </w:rPr>
   </w:style>
   <w:style w:type="paragraph" w:styleId="Heading1">
      <w:name w:val="heading 1"/>
      <wx:uiName wx:val="Heading 1"/>
      <w:basedOn w:val="Normal"/>
      <w:next w:val="Normal"/>
      <w:rsid w:val="008D1824"/>
      <w:pPr>
```

```
            <w:pStyle w:val="Heading1"/>
            <w:keepNext/>
            <w:spacing w:before="240" w:after="60"/>
            <w:outlineLvl w:val="0"/>
         </w:pPr>
         <w:rPr>
            <w:rFonts w:ascii="Arial" w:h-ansi="Arial" w:cs="Arial"/>
            <wx:font wx:val="Arial"/>
            <w:b/>
            <w:b-cs/>
            <w:kern w:val="32"/>
            <w:sz w:val="32"/>
            <w:sz-cs w:val="32"/>
         </w:rPr>
      </w:style>
      ...
   </w:styles>
   <w:shapeDefaults>
      <o:shapedefaults v:ext="edit" spidmax="2050"/>
      <o:shapelayout v:ext="edit">
         <o:idmap v:ext="edit" data="1"/>
      </o:shapelayout>
   </w:shapeDefaults>
   <w:docPr>                                                          ❻
      <w:view w:val="print"/>
      <w:zoom w:percent="100"/>
      <w:doNotEmbedSystemFonts/>
      <w:attachedTemplate w:val=""/>
      <w:defaultTabStop w:val="708"/>
      <w:hyphenationZone w:val="425"/>
      <w:punctuationKerning/>
      <w:characterSpacingControl w:val="DontCompress"/>
      <w:optimizeForBrowser/>
      <w:validateAgainstSchema/>
      <w:saveInvalidXML w:val="off"/>
      <w:ignoreMixedContent w:val="off"/>
      <w:alwaysShowPlaceholderText w:val="off"/>
      <w:compat>
         <w:breakWrappedTables/>
         <w:snapToGridInCell/>
         <w:wrapTextWithPunct/>
         <w:useAsianBreakRules/>
         <w:dontGrowAutofit/>
      </w:compat>
   </w:docPr>
   <w:body>                                                           ❼
      <wx:sect>
         <w:p>
            <w:r>
               <w:t>Hallo Welt!</w:t>
            </w:r>
```

```
            </w:p>
            <w:sectPr>                                                    ❽
               <w:pgSz w:w="11906" w:h="16838"/>
               <w:pgMar w:top="1417" w:right="1417" w:bottom="1134"
                   w:left="1417" w:header="708" w:footer="708"
               w:gutter="0"/>
               <w:cols w:space="708"/>
               <w:docGrid w:line-pitch="360"/>
            </w:sectPr>
         </wx:sect>
      </w:body>
</w:wordDocument>
```

❶ Das Wurzelelement mit den Namensraumdeklarationen. Es werden immer alle Office-Namensräume eingetragen, auch wenn beispielsweise keine eingebetteten Grafiken im Word-Dokument vorkommen.
❷ Die Dokumenteigenschaften enthalten alle für das Office-Dokument wichtigen Informationen, u.a. den Namen des Erstellers, letzte Speicherung und den Namen der Datei.
❸ Die Vorgabewerte von Schriften.
❹ Die Vorgaben für Listen. Im vorliegenden Beispiel sind diese bis einschließlich der neunten Ebene definiert.
❺ Die Formatvorlagen für Absätze und Tabellen.
❻ `<w:docPr>` enthält eine Vielzahl von Kindelementen. Diese legen verschiedene Eigenschaften für das gesamte Dokument fest. Unter anderem wird dort angegeben, in welcher Ansicht das Dokument nach dem Öffnen angezeigt werden soll – hier in der Druckansicht (`<w:view w:val="print">`) – oder ob XML-Dateien (XML-Daten, die einem eigenen Schema folgen) auch nicht valide abgespeichert werden dürfen (`<w:saveInvalidXML w:val="off"/>`).
❼ Das `<w:body>`-Element enthält die inhaltlichen Informationen, wie die Texte, Tabellen, Listen etc.
❽ `<w:sectPr>` definiert mit seinen Kindelementen die Eigenschaften der Seiten, wie Seitengrößen (`<w:pgSz>`), Kopf- und Fußbereiche (`<w:pgMar>`) etc.

Vgl. Kapitel 7, S. 131

Eine solche Flut an Informationen mag abschreckend wirken, sollte es aber nicht. Zum einen spielen viele dieser Elemente für die gewöhnliche Arbeit mit WordML keine Rolle, zum anderen kann einem Word selbst helfen, diese zu erstellen. Möchten Sie beispielsweise wissen, wie eine Fußnote in WordML ausgezeichnet wird, brauchen Sie lediglich in einem leeren Word-Dokument eine Fußnote zu erzeugen und dieses Dokument als WordML abzuspeichern. Der Quellcode dieses WordML-Dokumentes zeigt Ihnen die Struktur, die Word benötigt, um eine solche Fußnote anzeigen zu können.

Damit Sie sich in einem WordML-Dokument gut zurechtfinden, wird Ihnen dieses Kapitel neben der Grundstruktur die wichtigsten Zusammenhänge zeigen.

Grundstruktur

Die allgemeine Grundstruktur eines WordML-Dokumentes mit den wichtigsten Elementen lässt sich wie folgt beschreiben:

- Das einzig zulässige Wurzelelement heißt `<w:wordDocument>`, es deklariert alle Namensräume und besitzt einige Kindelemente:
  - Das optionale Element `<o:DocumentProperties>` enthält Metainformationen über das Dokument, wie Autor, Titel etc.
  - Das optionale Element `<w:fonts>` definiert die Vorgabewerte zu den verwendeten Schriftarten.
  - Das optionale Element `<w:lists>` enthält alle Informationen zu den Listen.
  - Das optionale Element `<w:styles>` beinhaltet die Formatvorlagen zu den Absätzen, inzeiligen Auszeichnungen etc.
  - Das optionale Element `<w:docPr>` definiert den Seitenaufbau und enthält die Angaben zu den Rändern, den Fußnoten, dem Kopfbereich, die Seitenbreite etc.
  - Das Element `<w:body>` enthält den eigentlichen Inhalt des Dokumentes.

2. 2 Das Wurzelelement

Das Wurzelelement hat eine Reihe von Attributen und Namensraumdeklarationen. Außer der Namensraumdeklaration für WordML sind alle übrigen optional und könnten, wenn kein weiterer Namensraum verwendet wurde, weggelassen werden.

Vgl. Kapitel 2.1, S. 28

Sollte das Dokument gemischten Inhalt haben, d.h., es wurde Word als XML-Editor verwendet und es wurden XML-Daten innerhalb des WordML-Dokumentes abgespeichert, so ist die proprietäre Deklaration (URI und Namensraum) hier anzugeben.

Vgl. Kapitel 2.11, S. 73

Die Präfixe, die hier vorgegeben werden, wie z.B. `w` für WordML, sind zwar den XML-Regeln folgend frei wählbar, sollten aber dennoch in dieser Form übernommen werden. Word 2003 reagiert an vielen Stellen ansonsten recht eigenwillig, und es sollte daher möglichst vermieden werden, diese internen Word-Regeln zu verletzen.

Der folgende Code zeigt das Wurzelelement <w:wordDocument> mit den Angaben zu den Namensräumen:

```
<w:wordDocument xmlns:w="http://schemas.microsoft.com/office/word/2003/
    wordml" xmlns:v="urn:schemas-microsoft-com:vml" xmlns:w10="urn:sche-
    mas-microsoft-com:office:word" xmlns:sl="http://
    schemas.microsoft.com/schemaLibrary/2003/core" xmlns:aml="http://
    schemas.microsoft.com/aml/2001/core" xmlns:wx="http://
    schemas.microsoft.com/office/word/2003/auxHint" xmlns:o="urn:schemas-
    microsoft-com:office:office" xmlns:dt="uuid:C2F41010-65B3-11d1-
    A29F-00AA00C14882" w:macrosPresent="no" w:embeddedObjPresent="no"
    w:ocxPresent="no" xml:space="preserve">
```

Die Namenräume haben die folgenden Bedeutungen:

- `xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"`
  Deklariert mit dem Präfix w die WordML-Elemente und -Attribute. Der Name des zugrunde liegenden Schemas ist `wordnet.xsd` [2].

- `xmlns:v="urn:schemas-microsoft-com:vml"`
  Deklariert mit dem Präfix v die VML-Elemente und -Attribute, die zum Einbinden von Grafiken dienen. Das dazugehörige Schema ist leider nicht Teil des Referenzpaketes.

- `xmlns:w10="urn:schemas-microsoft-com:office:word"`
  Deklariert mit dem Präfix `w10` die Elemente und Attribute, die zur Positionierung von Grafiken dienen. Der Name des zugrunde liegenden Schemas lautet `w10.xsd` .

- `xmlns:sl="http://schemas.microsoft.com/schemaLibrary/2003/core"`
  Deklariert mit dem Präfix `sl` jene zwei Elemente und jene drei Attribute, die zur Einbindung eigener Schemas dienen. Der Name des zugrunde liegenden Schemas lautet `xsdlib.xsd`.

- `xmlns:aml="http://schemas.microsoft.com/aml/2001/core"`
  Deklariert mit dem Präfix `aml` die Elemente und Attribute, die zur Änderungsverfolgung, Kommentierung und zur Erstellung von Verweisen dienen. Das zugrunde liegende Schema hat den Namen `aml.xsd`.

- `xmlns:wx="http://schemas.microsoft.com/office/word/2003/auxHint"`
  Deklariert mit dem Präfix `wx` die Elemente und Attribute, die als Hinweise für die Konvertierung des Dokumentes in andere Formate wie z.B. HTML dienen. Die Elemente werden in Word selbst nicht ausgewertet. Der Name des zugrunde liegenden Schemas ist `wordnetaux.xsd`.

---

[2] Sie finden alle Schemas in der bereits erwähnten Dokumentation zu WordML (Microsoft Office 2003 XML Reference Schemas).

- `xmlns:o="urn:schemas-microsoft-com:office:office"`

  Deklariert mit dem Präfix `o` die Elemente und Attribute, die nähere Auskunft über die Datei geben. Die Elemente enthalten Angaben zum Autor, zur letzten Speicherung, das Erstellungsdatum etc. Dieses Schema wird ebenso in Excel verwendet. Der Name des zugrunde liegenden Schemas lautet `office.xsd`.

- `xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"`

  Deklariert mit dem Präfix `dt` ein Attribut, das den Datentyp eines Wertes festlegt. Der Name des zugrunde liegenden Schemas ist `dt.xsd` und es befindet sich im Ordner für Excel-Schemas.

2.3 Der Body-Bereich

Dokumentrumpf

Das `<w:body>`-Element kann unter anderem aus beliebig vielen Sections (`<wx:sect>`), Absätzen (`<w:p>`) oder Tabellen (`<w:tbl>`) bestehen. Die Absätze können wiederum Run-Text-Elemente (`<w:r>`) enthalten, die ihrerseits den textlichen Inhalt (`<w:t>`) auszeichnen.

Die `<wx:sect>`-Elemente werden wie alle Elemente mit dem Namensraum `wx` von Word 2003 nicht ausgewertet, können aber insbesondere zur hierarchischen Gliederung sehr hilfreich sein.

Vgl. Kapitel 2.11, S. 73

Es sind zudem alle Elemente eines propriäteren Schemas (sofern vorhanden) erlaubt.

Das `<w:body>`-Element bildet somit einen Rahmen um den Inhaltsteil des Word-Dokumentes, einen Einfluss auf das Layout des Dokumentes hat es nicht.

2.4 Absätze

Absätze werden mit dem Element `<w:p>` dargestellt. Das optionale Kindelement `<w:pPr>` weist dem Absatz über Referenzierung eine Formatvorlage zu. Wird kein `<w:pPr>`-Element eingesetzt, so verwendet Word 2003 die Standardformatvorlage `Normal`.

```
<w:styles>
...
<w:style w:type="paragraph" w:styleId="berschrift1">                    ❶
   <w:name w:val="heading 1"/>
   <wx:uiName wx:val="Überschrift 1"/>
   <w:basedOn w:val="Standard"/>
   <w:next w:val="Standard"/>
   <w:rsid w:val="008D1824"/>
   <w:pPr>
      <w:pStyle w:val="berschrift1"/>
      <w:keepNext/>
```

```
      <w:spacing w:before="240" w:after="60"/>
      <w:outlineLvl w:val="0"/>
   </w:pPr>
   <w:rPr>
      <w:rFonts w:ascii="Arial" w:h-ansi="Arial" w:cs="Arial"/>
      <wx:font wx:val="Arial"/>
      <w:b/>
      <w:b-cs/>
      <w:kern w:val="32"/>
      <w:sz w:val="32"/>
      <w:sz-cs w:val="32"/>
   </w:rPr>
</w:style>
...
<w:p>
   <w:pPr>
      <w:pStyle w:val="berschrift1"/>                                    ❷
   </w:pPr>
   <w:r>
      <w:t>Das ist eine Überschrift der ersten Ebene</w:t>
   </w:r>
</w:p>
<w:p>
   <w:r>
      <w:t>Dieser Absatz hat die Formatvorlage Standard</w:t>                  ❸
   </w:r>
</w:p>
```

Vgl. Kapitel 2.10, S. 68

❶ Das Attribut `w:styleId` hat als Wert eine ID für die betreffende Formatvorlage. `<wx:uiName>` entspricht dem in Word angezeigten Namen der Formatvorlage. Dieser Name stimmt oft nicht mit dem Wert der ID überein, da dort alle Umlaute, Leerzeichen und Sonderzeichen entfernt werden. Im unseren Fall wird aus `Überschrift 1` der ID-Wert `berschrift1`. Das Attribut `w:type` deklariert den Typ der Formatvorlage, dessen Wert `paragraph` sie als Absatzformatvorlage ausweist.

❷ Das Element `<w:pStyle>` ist ein Kindelement von `<w:pPr>` und weist dem Absatz über den Attributwert von `w:val` eine Vorlage zu.

❸ Absätze ohne eine zugeordnete Formatvorlage und ohne weitere Angaben zu deren Eigenschaften mittels `<w:pPr>` werden mit der Standardformatvorlage formatiert.

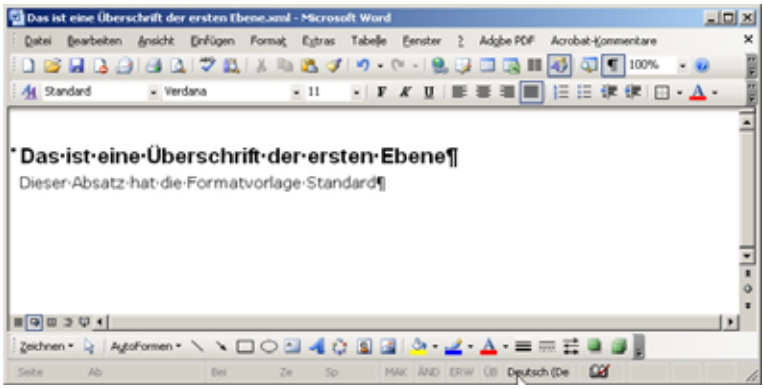


Abb. 2-2
Absätze

Absatzeigenschaften

Das Element `<w:pPr>` hat eine Reihe von optionalen Kindelementen, welche die Absatzeigenschaften im Detail bestimmen. In der Praxis sollten diese bei der Erstellung von Stylesheets zur Visualisierung nicht verwendet werden. Es ist einfacher und übersichtlicher, die Absatzeigenschaften wenn möglich über Formatvorlagen zuzuweisen.

Hier alle Kindelemente von `<w:pPr>` in der DTD-Schreibweise:

```
(pStyle?, (keepNext | keepLines | pageBreakBefore | framePr | widowControl
    | listPr | supressLineNumbers | pBdr | shd | tabs | suppressAutoHyphens
    | kinsoku | wordWrap | overflowPunct | topLinePunct | autoSpaceDE |
    autoSpaceDN | bidi | adjustRightInd | snapToGrid | spacing | ind |
    contextualSpacing | suppressOverlap | jc | textDirection | textAlign-
    ment | outlineLvl | divId | cnfStyle | rPr | sectPr | aml:annotation)
    *)
```

Die meisten der erlaubten Eigenschaften lassen sich über das Absatzformular steuern. Dieses erscheint unter `Format -> Absatz`.

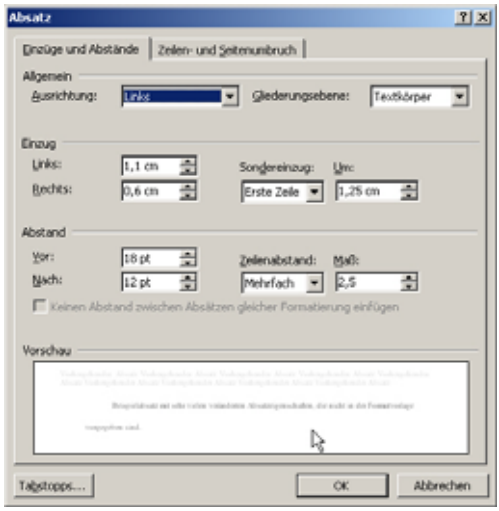


Abb. 2-3
Die Steuerung der Absatzeigenschaften

```
<w:p>
   <w:pPr>
      <w:keepNext/>                                                    ❶
      <w:keepLines/>                                                   ❷
      <w:pageBreakBefore/>                                             ❸
      <w:supressLineNumbers/>                                          ❹
      <w:suppressAutoHyphens/>                                         ❺
      <w:spacing w:before="360" w:after="240" w:line="600" w:line-     ❻
          rule="auto"/>
      <w:ind w:left="624" w:right="340" w:first-line="709"/>           ❼
      <w:jc w:val="left"/>                                             ❽
   </w:pPr>
   <w:r>
      <w:t>Beispielabsatz mit sehr vielen veränderten Absatzeigenschaften,
          die nicht in der Formatvorlage vorgegeben sind.</w:t>
   </w:r>
</w:p>
<w:p>
   <w:pPr>
      <w:spacing w:before="600" w:after="600"/>                        ❻
   </w:pPr>
   <w:r>
      <w:t>Ein Folgeabsatz mit einem größeren Abstand zu den Nachbarabsät-
          zen.</w:t>
   </w:r>
</w:p>
<w:p>
   <w:pPr>
      <w:ind w:left="1701"/>                                           ❼
   </w:pPr>
   <w:r>
      <w:t>Ein eingezogener Absatz.</w:t>
   </w:r>
</w:p>
<w:p>
   <w:pPr>
      <w:shd w:val="clear" w:color="auto" w:fill="8C8C8C"/>            ❾
   </w:pPr>
   <w:r>
      <w:t>Ein Absatz mit einer Hintergrundschattierung</w:t>
   </w:r>
</w:p>
```

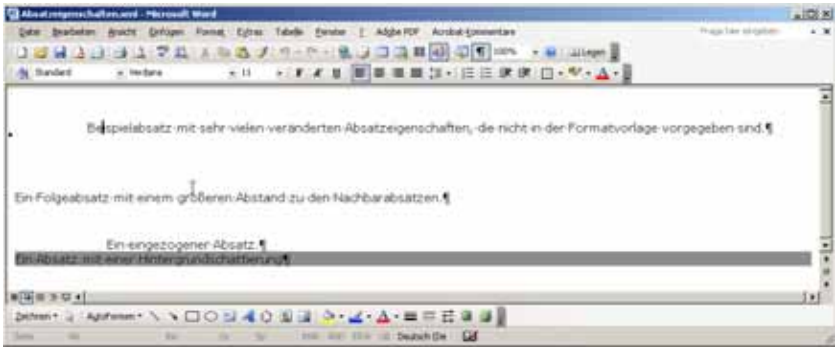


Abb. 2-4 *Absatzeigenschaften*

❶ Das Element `<w:keepNext>` verhindert einen Seitenumbruch zwischen diesem und dem folgenden Absatz oder Objekt. Es kann auch das Attribut `w:val` enthalten, das die Ausprägungen `on` und `off` besitzen kann und das Zusammenhalten mit dem nächsten Objekt an- oder ausschaltet.

❷ `<w:keepLines>` verhindert einen Seitenumbruch innerhalb des Absatzes. Es kann ebenso wie `<w:keepNext>` das Attribut `w:val` enthalten, das die Ausprägungen `on` und `off` haben kann, mit denen ein möglicher Seitenumbruch für den betreffenden Absatz zugelassen oder unterbunden werden kann.

❸ `<w:pageBreakBefore>` erzwingt einen Seitenumbruch vor dem Absatz. Auch hier ist ein Attribut `w:val` mit den Ausprägungen `on` und `off` erlaubt, mit denen ein Seitenumbruch vor dem betreffenden Absatz erzwungen oder gegebenenfalls gebilligt werden kann.

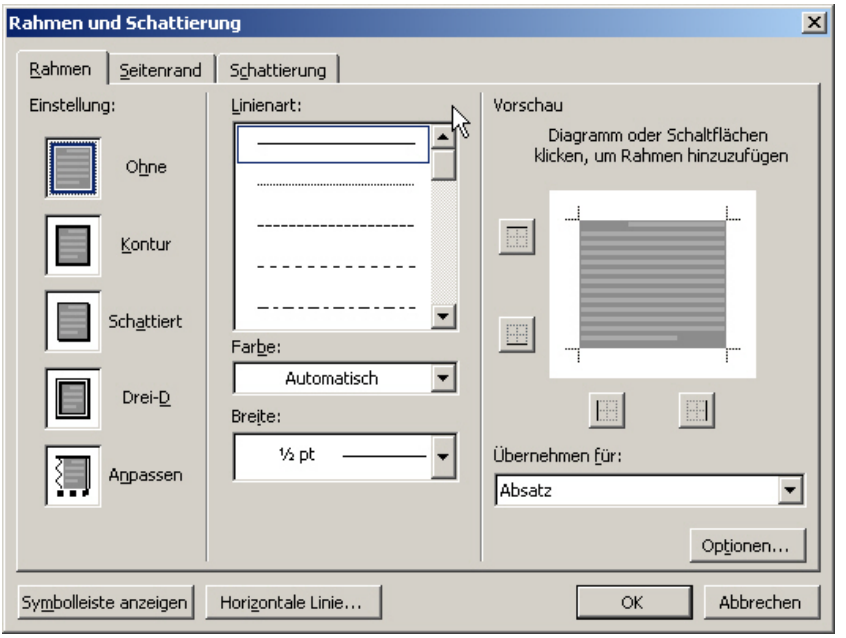
❹ `<w:suppressLineNumbers>` kann die in einem Dokument eingestellte Zeilennummerierung wieder ausschalten. Hat das Dokument keine Zeilennummerierung, wird es ignoriert. Es kann das Attribut `w:val` enthalten, das mit den Ausprägungen `on` und `off` die Zeilennummerierung an- bzw. ausschaltet.

❺ `<w:suppressAutoHyphens>` schaltet die automatische Silbentrennung in einem Absatz aus. Es kann das Attribut `w:val` enthalten, das mit den Ausprägungen `on` und `off` die Silbentrennung an- bzw. ausschaltet.

❻ `<w:spacing>` behandelt die Abstände vor und nach Absätzen sowie zwischen den Zeilen eines Absatzes. Das Attribut `w:before` definiert den Abstand vor einem Absatz in Twips (**tw**enti**e**ths of a **p**oint) die hier angegebenen 360 Twips entsprechen folglich 18pt. `w:after` gibt den Abstand nach einem Absatz in Twips an. Das Attribut `w:line` definiert den Zeilenabstand in Twips. Die hier angegebenen 600 Twips sind eine Umrechnung der Schriftgröße (12pt) multipliziert mit dem Zeilenabstand (2,5-fach) mal 20 (Twips). Das Attribut `w:line-rule` rechnet beim Wert `auto` die Abstände abhängig von den Schriftgrößen um. Wäre der Wert `exact`, so würden die in `w:line` angegebenen Maße in jedem Fall eingehalten.

❼ <w:ind> bestimmt die Einzüge eines Absatzes. Mit dem Attribut w:left kann man einen linken Einzug in Twips festlegen, mit dem Attribut w:right einen rechten und mit dem Attribut w:first-line kann man für einen hängenden Einzug sorgen.
❽ Mit <w:jc> lässt sich die Ausrichtung eines Absatzes festlegen. Die erlaubten Werte sind: left(links), center(zentriert), right(rechts) und both (Blocksatz).
❾ Das Element <w:shd> erlaubt eine Hintergrundfarbe, das Attribut w:fill legt diese als RGB-Hexadezimalwert fest. Das Attribut w:val erzeugt je nach Wert gepunktete Hintergründe. Beim Wert clear ist diese Eigenschaft ausgeschaltet. Die hier verwendeten Hintergrundfarbeinstellungen lassen sich über das Formular Format -> Rahmen und Schattierung ändern.

Abb. 2-5
Rahmen und Schattierung von Absätzen



Inzeiligen Formateigenschaften

Eine weitere Möglichkeit, Absätze zu formatieren, ist die Zuordnung von inzeiligen Formateigenschaften für den gesamten Absatz. Im folgenden Beispiel wird einem Absatz mit dem Element <w:rPr> eine eigentlich inzeilige Formatierung zugeordnet und an späterer Stelle z.T. wieder rückgängig gemacht. Diese und weitere inzeilige Formatierungen werden im folgenden Kapitel genauer betrachtet werden.

```
<w:p>
   <w:pPr>
      <w:rPr>
         <w:b />
      </w:rPr>
   </w:pPr>
   <w:r>
      <w:t>Absatz mit </w:t>
   </w:r>
   <w:r>
      <w:rPr>
         <w:b w:val="off"/>
         <w:i/>
      </w:rPr>
      <w:t>inzeiligen</w:t>
   </w:r>
   <w:r>
      <w:rPr>
         <w:b w:val="off"/>
      </w:rPr>
      <w:t> Eigenschaften</w:t>
   </w:r>
</w:p>
```

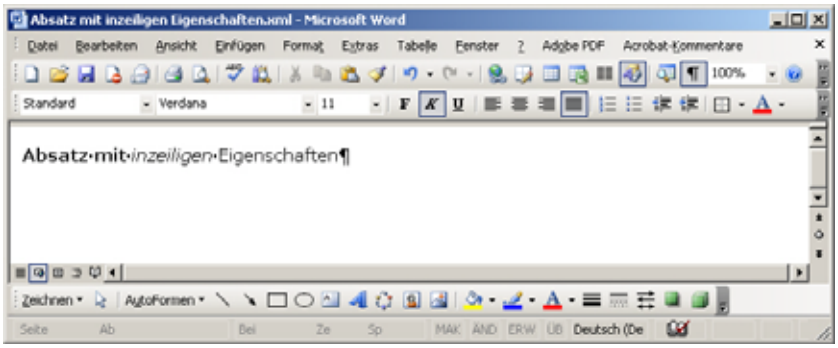


Abb. 2-6
Inzeilige Auszeichnung im Absatz

2.5 Inzeilige Elemente

Textformatierung

Inzeilige Elemente, Formatierungen und der Text befinden sich innerhalb des Run-Text-Elementes `<w:r>`. Das `<w:r>`-Element verfügt über eine ganze Reihe von Kindelementen. In der DTD-Syntax lassen sich diese wie folgt darstellen:

```
(rPr?, (aml:annotation | br | t | delText | instrText | delInstrText | no-
    BreakHyphen | softHyphen | annotationRef | footnoteRef | endnoteRef |
    separator | continuationSeparator | footnote | endnote | sym | pgNum |
    cr | tab | pict | fldChar | ruby | wx:t+)
```

Das Element <w:rPr> übernimmt an dieser Stelle eine ähnliche Funktion wie das <w:pPr>-Element für die Absätze. Seine Funktion ist es, als Containerelement alle Formatierungseigenschaften zu bündeln. So lassen sich Formatvorlagen zuordnen, aber auch lokale Formatierungen festlegen.

Im folgenden Beispiel wurden sowohl Formatvorlagen als auch lokale Formateigenschaften verwendet, optisch sehen beide inzeiligen Auszeichnungen gleich aus:

```
...
   <w:style w:type="character" w:styleId="Hervorhebung">                ❷
      <w:name w:val="Emphasis"/>
      <w:basedOn w:val="Absatz-Standardschriftart"/>
      <w:rsid w:val="00810D0D"/>
      <w:rPr>
         <w:i/> <w:i-cs/>
      </w:rPr>
   </w:style>
</w:styles>
...
<w:p>
   <w:r>
      <w:t>Dies ist ein </w:t>                                               ❹
   </w:r>
   <w:r>
      <w:rPr>                                                           ❶
         <w:rStyle w:val="Hervorhebung"/>                               ❷
      </w:rPr>
      <w:t>Absatz</w:t>
   </w:r>
   <w:r>
      <w:t> mit </w:t>
   </w:r>
   <w:r>
      <w:rPr>                                                           ❶
         <w:i/>                                                         ❸
      </w:rPr>
      <w:t>inzeiligen</w:t>
   </w:r>
   <w:r>
      <w:t> Auszeichnungen.</w:t>                                         ❹
   </w:r>
</w:p>
```

❶ Das Element <w:rPr> definiert die inzeiligen Eigenschaften des folgenden Textes.

❷ Das Element `<w:rStyle>` ordnet dem Run-Text eine inzeilige Formatvorlage zu. Das Attribut `w:val` enthält die RefID zu dieser Formatvorlage.
❸ Das Element `<w:i>` formatiert den Text kursiv. Es kommt dann zum Einsatz, wenn der Button `k` in der Formatierleiste verwendet wird.
❹ Texte ohne inzeilige und ohne Absatzformatierung werden mit der Formatvorlage `Standard` formatiert.

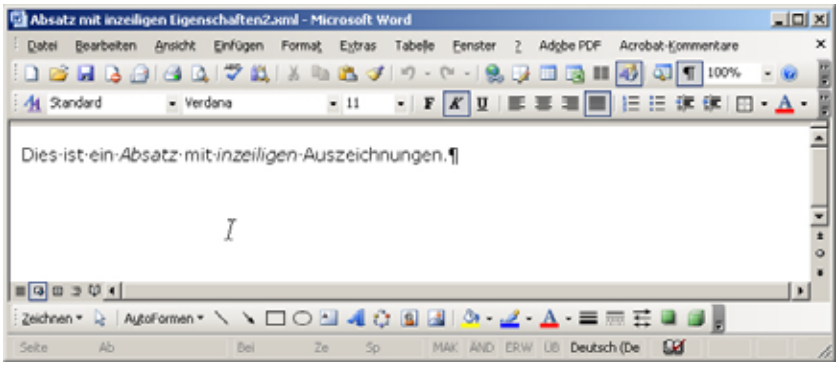


Abb. 2-7
Inzeilige Formatierung

Die Run-Text-Formatierungseigenschaften entsprechen im Wesentlichen den Einstellungsmöglichkeiten des Zeichenfensters. Sie gelangen zu diesem Menü über `Format –> Zeichen`.

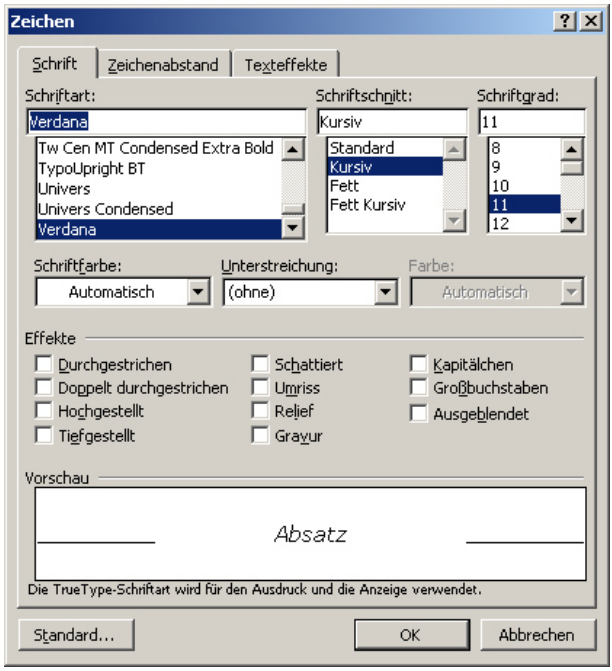


Abb. 2-8
Das Zeichenfenster

Das <w:rPr>-Element verfügt über eine Reihe von Kindelementen. In der DTD-Syntax lassen sich diese wie folgt darstellen:

```
(rStyle | rFonts | wx:font | wx:sym | b | b-cs | i | i-cs | caps | smallCaps
    | strike | dstrike | outline | shadow | emboss | imprint | noProof |
    snapToGrid | vanish | webHidden | color | spacing | w | kern | position
    | sz | sz-cs | highlight | u | effect | bdr | shd | fitText | vertAlign
    | rtl | cs | em | hyphen | lang | asianLayout | specVanish | aml:an-
    notation)+
```

```
<w:p>
   <w:r>
      <w:t>In dieser Zeile kommen </w:t>
   </w:r>
   <w:r>
      <w:rPr>
         <w:b/>                                                    ❶
      </w:rPr>
      <w:t>fett</w:t>
   </w:r>
   <w:r>
      <w:t>, </w:t>
   </w:r>
   <w:r>
      <w:rPr>
         <w:i/>                                                    ❷
      </w:rPr>
      <w:t>kursiv</w:t>
   </w:r>
   <w:r>
      <w:t> und </w:t>
   </w:r>
   <w:r>
      <w:rPr>
         <w:u w:val="single"/>                                     ❸
      </w:rPr>
      <w:t>unterstrichen</w:t>
   </w:r>
   <w:r>
      <w:t> vor. Alles </w:t>
   </w:r>
   <w:r>
      <w:rPr>
         <w:b/>
         <w:i/>
         <w:u w:val="single"/>
      </w:rPr>
      <w:t>gleichzeitig.</w:t>
   </w:r>
</w:p>
```

7 Stylesheet-Entwicklung: Von XML zu WordML

Eine zentrale Einsatzmöglichkeit der XML-Fähigkeit von Word 2003 ist die automatisierte Erzeugung von Word-Dokumenten aus XML-Quellen mittels XSLT.

Roundtrip

Diese Möglichkeit wird in der Praxis oft angewendet, um XML-Daten für Autoren aufzubereiten und als Word-Dokumente zur Verfügung zu stellen. Zusammen mit Stylesheets zum automatisierten Auszeichnen von WordML bildet dieser Workflow einen sogenannten Roundtrip. Das heißt, Word-Daten können in XML überführt und anschließend den Autoren wieder in Word zur Verfügung gestellt werden.

In diesem Kapitel werden die dazu benötigten Workflows dargestellt und einige Tricks gezeigt, wie eine solche Transformation von XML zu WordML möglichst einfach zu bewerkstelligen ist. Wir werden hierfür in den nächsten Kapiteln schrittweise vorgehen und jeden Schritt detailliert beschreiben.

Für alle folgenden Kapitel können die hier verwendeten Beispieldateien von der Homepage des Buches http://www.data2type.de/xml/word2003.html heruntergeladen werden.

7.1 Die Struktur der Ausgangsdaten

Ausgangspunkt für eine Transformation sind stets die Ausgangsdaten und deren Struktur. Sie benötigen folglich eine DTD oder ein Schema, welche die Struktur definieren.

XMLSpy

Da Word 2003 jedoch nicht in der Lage ist, DTDs zu lesen, muss ein Schema angegeben werden. Entweder man schreibt solch ein Schema selbst oder man lässt es sich einfacherweise aus einer DTD automatisch erzeugen (u.a. ist hierzu der XMLSpy in der Lage).

Vgl. Kapitel 6, S. 125

Die in den folgenden Kapiteln benutzten XML-Daten beruhen auf der DocBook-Struktur.

Schema

Um eine schemagesteuerte XML-Datei in Word 2003 laden und bearbeiten zu können, muss in einem ersten Schritt das zugrunde liegende

Schema Word 2003 bekannt gegeben und in der Schemabibliothek abgespeichert werden.

Wählen Sie im Menü *Extras -> Vorlagen und Add-Ins -> XML-Schema* und klicken Sie dort auf *Schema hinzufügen*. Es öffnet sich anschließend ein Auswahlfenster der verfügbaren Schemadateien. Wählen Sie hier das Schema `docbook.xsd` aus. Nun erscheint ein weiteres Fenster, in dem die Eingabe einer URI und eines Aliasnamens erwartet wird. Sowohl die URI als auch der Alias sind frei wählbar.

Vgl. Kapitel 1.1, S. 3

In unserem Beispielworkflow wurde als URI `http://docbook.org/ns/docbook` und als Alias `docbook` gewählt.

Abb. 7-1
Schemaeinstellungen

7.2 Das Beispieldokument erzeugen

Für die Entwicklung eines WordML-Stylesheets benötigen Sie ein Beispieldokument, das alle Elemente und die wichtigsten bzw. typischen Strukturen enthalten sollte. Dieses Dokument sollte nicht allzu umfangreich sein, da sonst die Überprüfung der Resultate während der Entwicklung unnötig erschwert wird. Die Datei `docbook.xml` wird uns bei den folgenden Kapiteln als Beispieldokument dienen.

Für die Entwicklung von WordML-Stylesheets können Sie Word 2003 als Codegenerator nutzen. Dazu brauchen Sie lediglich eine formatierte Datei, die als WordML-Datei abgespeichert wird. Wenn nun die Beispieldatei formatiert und als WordML abgespeichert wird, können Sie immer Ihr mit XSLT-Mitteln erzeugtes Dokument mit dem Ideal, das von Word 2003 erzeugt wurde, vergleichen.

Sie sollten folglich das XML-Dokument in Word 2003 laden und dort entsprechend Ihren Vorstellungen formatieren. Diese Arbeit wurde im Beispieldokument `docbook-wordml.xml` bereits getätigt. Sie sollten insbesondere darauf achten, dass alle Formatvorlagen, die Sie verwenden möchten, in diesem Dokument Anwendung finden.[1]

[1] Formatvorlagen, die in einer *dot-Datei* abgespeichert wurden und nicht im Dokument verwendet wurden, werden im WordML nicht kodiert.

7. 3 Der Grundaufbau der Stylesheets

Wie bereits in den vorangegangenen Kapiteln erläutert, ließe sich ein XSLT-Stylesheet schreiben, das alle Templates mit den Formatvorlagen, der eigentlichen Verarbeitung und dem Grundaufbau eines WordML-Dokumentes enthielte. Es hätte allerdings den Nachteil, dass es äußerst umfangreich ausfallen würde und darüber hinaus nicht modular aufgebaut wäre.

Es ist daher empfehlenswert, mit mehreren Stylesheets zu arbeiten, die verschiedene Aufgaben übernehmen. Die folgende Abbildung zeigt eine solche Aufteilung in verschiedene Aufgabenbereiche.

- Das Stylesheet `Formatvorlagen.xsl` enthält alle Formatvorlagen, die verwendet werden dürfen.
- Die Datei `Grundaufbau.xsl` enthält den Rahmen eines WordML-Dokumentes. In diesem Rahmen sind u.a. Angaben über die Seitengröße, die Ränder etc. abgelegt.
- Das eigentliche Stylesheet, das die Templates enthält, welche die eigentliche Transformation des XML-Dokumentes in ein WordML-Dokument vornehmen, ist die Datei `Logik.xsl`.

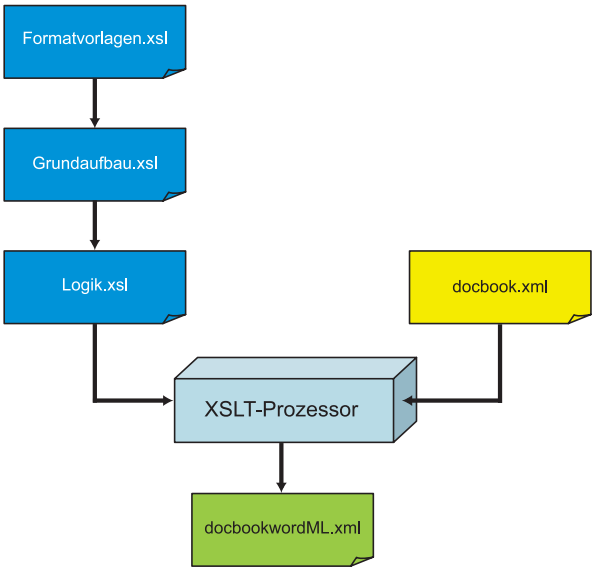


Abb. 7-2
XSLT-Ablaufschema

In den folgenden Abschnitten werden wir diesen beschriebenen Workflow anhand des Beispieldokuments `docbook-wordml.xml` aufzeigen. Dort befinden sich alle relevanten Informationen für die Stylesheets `Formatvorlagen.xsl` und `Grundaufbau.xsl`.

Das Formatvorlagen-Stylesheet

Das Stylesheet `Formatvorlagen.xsl` enthält alle Formatvorlagen für die zu erzeugenden WordML-Dokumente. Es hat nur ein einziges Template, das allerdings nicht von Elementen in unserer Ausgangsdatei `docbook.xml` abhängig ist.

Die Abtrennung der Formatvorlagen in ein eigenes Stylesheet `Formatvorlagen.xsl` bietet den Vorteil, dass diese Vorlagen ausgetauscht werden können, indem einfach ein anderes Stylesheet mit anderen Vorlagen aufgerufen wird und sich dadurch das Layout ändert, ohne dass hierfür Eingriffe in die eigentlichen Templates (`Logik.xsl`) nötig sind.

Die Grundstruktur dieses Stylesheets sieht wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/        ❶
    XSL/Transform" xmlns:w="http://schemas.microsoft.com/office/word/
    2003/wordml" xmlns:v="urn:schemas-microsoft-com:vml"
    xmlns:w10="urn:schemas-microsoft-com:office:word" xmlns:sl="http://
    schemas.microsoft.com/schemaLibrary/2003/core" xmlns:aml="http://
    schemas.microsoft.com/aml/2001/core" xmlns:wx="http://
    schemas.microsoft.com/office/word/2003/auxHint" xmlns:o="urn:schemas-
    microsoft-com:office:office" xmlns:dt="uuid:C2F41010-65B3-11d1-
    A29F-00AA00C14882">
   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="no"/>
   <xsl:template name="vorlagen">                                       ❷
   <!--VORLAGEN-->
   </xsl:template>
</xsl:stylesheet>
```

❶ Da in den Vorlagen verschiedene Namensräume vorkommen können und diese bei der Verwendung der entsprechenden Elemente alle bekannt sein müssen, werden für das Stylesheet einfach alle in WordML erlaubten Namensräume angegeben.

Vgl. Kapitel 3.4, S. 82

❷ Das Stylesheet besitzt lediglich ein Template, das den Namen `vorlagen` hat und alle Formatvorlagen enthält. Bei einem Aufruf dieses Templates mit `<xsl:call-template name="vorlagen">` wird der Inhalt des Templates an die entsprechende Stelle kopiert.

In einem weiteren Schritt werden nun die Formatvorlagen in das Template eingefügt. Hierfür wird das Beispieldokument `docbook-wordml.xml` mit einem XML-Editor geöffnet und alle Elemente beginnend mit dem `<w:fonts>`-Element bis zum `<w:body>`-Element kopiert und in das Template eingefügt.

Hier ein Ausschnitt mit Anfang und Ende des Templates:

```
...
   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="no"/>
   <xsl:template name="vorlagen">
```

8 Von WordML zu XML

Eine der interessantesten Anwendungsmöglichkeiten von WordML ist die Transformation der Layoutstruktur in eine Struktur, die unabhängig von der Ausgabe ist.

In diesem Kapitel werden wir die WordML-Daten in DocBook umwandeln und so einen der Grundgedanken von XML, nämlich die klassische Trennung von Struktur (DTD, Schema etc.) von Inhalt (XML-Instanz) und Layout (XSLT-Stylesheets etc.), wiederherstellen.

8.1 Praxisworkflow

Wie wichtig eine solche Extraktion der Inhalte sein kann, soll der folgende Beispielworkflow verdeutlichen, der in Verlagen und in der technischen Dokumentation immer mehr Einzug findet.

Verlage haben meist viele Autoren, denen nicht zuzumuten wäre, mit XML-Editoren ihre Dokumente zu verfassen. Word hingegen ist ein weit verbreitetes Werkzeug, bei dem die Autoren mit Hilfe von Dokument- und Formatvorlagen ihre Texte schreiben können.

Diese Dokumente werden nach Ablieferung an den Verlag mit einem Word-zu-XML-Stylesheet in eine XML-Struktur überführt, in der auch die anderen Publikationen erfasst sind.

Die erzeugten XML-Dokumente werden vom Verlag oder von dessen Dienstleister redaktionell bearbeitet, strukturelle Fehler werden beseitigt und beispielsweise mittels XSL-FO in ein druckfertiges PDF gebracht.

Das XML-Dokument wird mit einem WordML-Stylesheet wieder zu einem Word-Dokument umgewandelt und für die nächste Auflage zurück an den Autor gesendet.

Bei der nächsten Auflage lässt sich das nun veränderte Word-Dokument mittels eines Word-internen Vergleiches mit dem zugesendeten Word-Dokument vergleichen. Die Unterschiede können nun entweder direkt in einem XML-Editor erfasst werden oder bei umfangreichen Änderungen wieder mit einem Word-zu-XML-Stylesheet in eine XML-Struktur überführt werden.

Nun werden Sie sich vielleicht fragen, warum sollte man einen Vergleich fahren und die Unterschiede manuell in einem XML-Editor ein-

pflegen, wenn es doch ein Stylesheet gibt, das einem die Arbeit vollautomatisch abnimmt. Diese Frage beantworten wir im folgenden Abschnitt 8.2.

8.2 Problemstellungen

Ein generelles Problem bei der Umwandlung von WordML nach XML besteht darin, dass Word anders als ein XML-Editor keine Strukturüberprüfungen vornimmt und dem Autor sehr viele Freiheiten lässt. Hier zwei Beispiele zur Veranschaulichung:

1. Ein Autor benutzt nicht die Formatvorlagen, sondern formatiert einen Titel, indem er einem normalen Standardabsatz die Schriftgröße 36pt zuweist. Bei einer Transformation wiederum müssen in Templates Annahmen getroffen werden, d.h., eine Überschrift sollte mit einer Formatvorlage formatiert worden sein, was sich in WordML entsprechend wiederfinden lässt. In diesem Fall würde der Titel wohl als Absatz erkannt werden und daher falsch ausgezeichnet werden, der Kapitelcontainer (`<section>`) würden ebenfalls nicht zugeordnet werden, was u.U. zu invaliden Strukturen führen kann, in jedem Fall aber zu inhaltlich falschen Kapitelstrukturen.
2. Ein weiterer "Klassiker" sind falsche Zuordnungen von Formatvorlagen. Eine Überschrift wurde beispielsweise als `Überschrift 4` statt `Überschrift 2` formatiert. Bei der Umwandlung in XML wird dies erkannt und es werden Kapitelcontainer für die Ebene 2 und 3 gebildet, ohne dass sie einen Titel haben.

Wie aus den Beispielen gut ersichtlich wird, kann es keine technische Lösung geben, die diese Probleme immer zuverlässig löst. Eine fachkundige Überprüfung ist daher immer ratsam. Aus Erfahrung kann man jedoch sagen, dass Automatisierungsgrade von bis zu 99% realistisch sind.

8.3 Die Stylesheet-Entwicklung

Wie auch für die Visualisierung von XML-Dokumenten wird bei der Extraktion von XML-Inhalten aus WordML-Dokumenten XSLT eingesetzt. Im Folgenden werden einige Templates gezeigt, die uns helfen, aus einem WordML-Dokument eine DocBook-Instanz zu erzeugen. Die generierte Instanz muss aus den bereits oben genannten Gründen nicht zwangsläufig valide sein, sollte aber in jedem Fall wohlgeformt sein, so dass sie in einem XML-Editor geladen werden kann.

Als Grundlage soll uns die folgende Word-Datei dienen. Die Absätze und Überschriften wurden korrekt mit Formatvorlagen erstellt, die Listen

und inzeiligen Auszeichnungen mit den Bedienschaltflächen auf der Oberfläche.

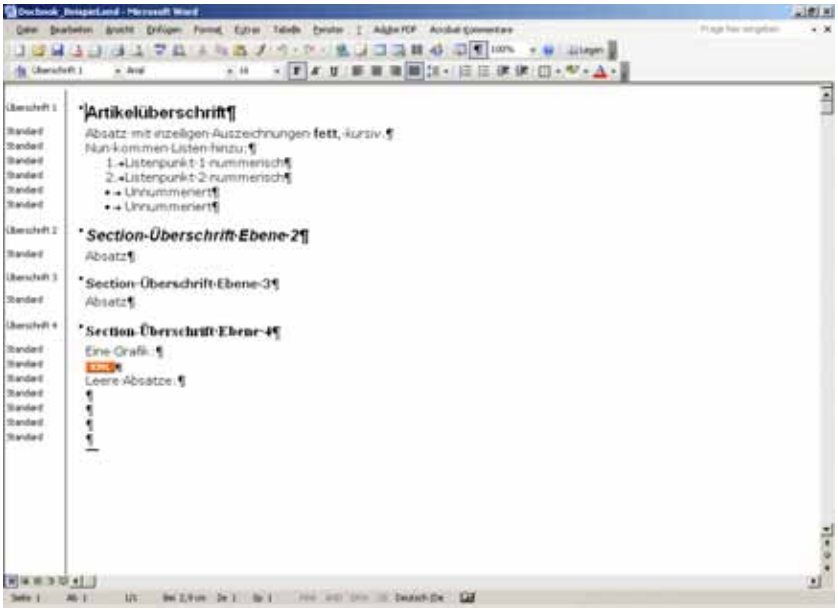


Abb. 8-1
Das Beispieldokument

Der Grundaufbau

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" w:macrosPresent="no"                    ❶
    w:embeddedObjPresent="no" w:ocxPresent="no" xmlns:xsl="http://
    www.w3.org/1999/XSL/Transform" xmlns:w="http://schemas.microsoft.com/
    office/word/2003/wordml" xmlns:v="urn:schemas-microsoft-com:vml"
    xmlns:w10="urn:schemas-microsoft-com:office:word" xmlns:sl="http://
    schemas.microsoft.com/schemaLibrary/2003/core" xmlns:aml="http://
    schemas.microsoft.com/aml/2001/core" xmlns:wx="http://
    schemas.microsoft.com/office/word/2003/auxHint" xmlns:o="urn:schemas-
    microsoft-com:office:office" xmlns:dt="uuid:C2F41010-65B3-11d1-
    A29F-00AA00C14882" exclude-result-prefixes="w v w10 sl aml o dt wx">
   <xsl:output version="1.0" method="xml" indent="no"/>
   <xsl:template match="/">                                           ❷
      <article>                                                       ❷
         <xsl:apply-templates select="//w:body"/>                     ❸
      </article>
   </xsl:template>
</xsl:stylesheet>
```

❶ Um alle Namensräume, die im Stylesheet vorkommen können, verwenden zu dürfen, werden alle deklariert. Das Attribut `exclude-result-prefixes` verhindert, dass diese Namensräume in unserem Zieldokument ausgegeben werden. Alle Namensräume, die nicht übernommen werden sollen, müssen hier aufgezählt werden.

❷ Das Template verweist mit `match="/"` auf den Wurzelknoten des XML-Dokumentes und kann daher nur einmal zu Beginn der Transformation aufgerufen werden. Daher enthält es das Wurzelelement `<article>` des Zieldokumentes.
❸ Ziel unserer Transformation ist die Überführung unserer Inhalte ins DocBook-Format. Daher ignorieren wir alle Formatvorlagen und Metainformationen, die in WordML enthalten sind, und selektieren den Inhalt von `<w:body>`, das unsere Texte enthält.

Die Container und Titel

```
<xsl:template match="//wx:sub-section" >                                    ❶
   <section>                                                                ❶
      <xsl:apply-templates/>
   </section>
</xsl:template>
<xsl:template match="w:p[descendant::*/@w:val='Heading1'] | w:p[des-         ❷
    cendant::*/@w:val='Heading2'] | w:p[descendant::*/@w:val='Heading3']
    | w:p[descendant::*/@w:val='Heading4']" priority="2">
   <title><xsl:apply-templates select=".//w:t"/></title>
</xsl:template>
```

❶ Die Containerelemente `<section>` sind dank der `<wx:sub-section>`-Elemente in WordML relativ leicht zu finden. Diese Elemente mit dem Namensraum `wx` werden von Word nicht zur Formatierung benutzt, sondern sind lediglich eine Hilfe für weiterführende Prozesse. Bei jedem Absatz, der eine Überschriftformatierung enthält, werden diese Elemente in Containerform um das gesamte Kapitel eingefügt. Daher muss hier nur danach gesucht werden und die Elemente in `<section>` überführt werden.
❷ Da die Hierarchieebene in DocBook von den Containerelementen `<article>` und `<section>` festgelegt werden und alle Überschriften mit dem Element `<title>` ausgezeichnet werden, ist es hier nicht notwendig, zu unterscheiden, mit welcher Überschriftsformatvorlage gearbeitet wurde. Alle Absätze, die einer Formatvorlage mit Überschriftscharakter zugeordnet wurden, werden mit dem Element `<title>` ausgezeichnet.

Die Absätze und inzeiligen Elemente

```
<xsl:template match="w:r[descendant::w:b]"><emphasis                        ❶
    role="bold"><xsl:apply-templates select=".//w:t"/></emphasis></
    xsl:template>
<xsl:template match="w:r[descendant::w:i]"><emphasis><xsl:apply-            ❶
    templates select=".//w:t"/></emphasis></xsl:template>
<xsl:template match="w:p[w:r]" priority="1">                                ❷
   <para><xsl:apply-templates/></para>
</xsl:template>
```

❶ Alle <w:r>-Elemente, die als Formatierung <w:b> oder <w:i> enthalten, werden als <emphasis>-Elemente umgesetzt. Im Falle von Fettformatierung wird das Attribut role="bold" hinzugefügt.
❷ Die Absatzelemente <w:p>, die auch ein <w:r>-Element enthalten, werden in <para>-Elemente umgewandelt. Durch die Bedingung wird sichergestellt, dass leere Absätze nicht in das Zieldokument übernommen werden.

Die Grafiken

```
<xsl:template match="w:p[descendant::v:imagedata]" priority="2">          ❶
   <mediaobject>
      <imageobject>
         <imagedata fileref="{.//v:imagedata/@src}" ></imagedata>          ❷
      </imageobject>
   </mediaobject>
</xsl:template>
```

❶ Wenn ein Absatz ein <v:imagedata>-Element enthält, muss es sich um eine Grafik handeln. Das Attribut priority="2" wird wie auch schon bei den Überschriften dazu verwendet, eine Rangordnung für Templates zu definieren. Bei zwei Templates, die beide zutreffend sind, entscheidet sich der XSLT-Prozessor für dasjenige, das den ausgewählten Knoten genauer beschreibt. In unserem speziellen Fall ist es jedoch so, dass das Template für die <para>-Elemente präziser ist. Es stellt als Bedingung, dass das <w:p>-Element ein <w:r>-Kindelement haben muss. Daher würde dieses Template nie benutzt werden. Durch den höheren Wert des Attributes priority im Abbildungstemplate im Vergleich zu dem für die Absätze wird immer das Abbildungstemplate eingesetzt, wenn die Bedingung erfüllt wurde.
❷ Das Attribut fileref des Elementes <imagedata> enthält den Pfad zur Grafikdatei. Die geschweiften Klammern erlauben es in XSLT, Attributwerte direkt einzufügen, ohne den Umweg über das XSLT-Element <xsl:attribut> zu gehen.

Mit diesem Pfad .//v:imagedata/@src wird das Attribut src, das in WordML den Pfad zur Grafikdatei enthält, direkt übernommen.

Die Listen

Wie wir in den vorherigen Kapiteln, die sich mit Listen befassten, gesehen haben, ist das Konstrukt der Listen eher problematisch, insbesondere wegen der fehlenden Containerelemente in WordML.

Vgl. Kapitel 2.6, S. 51

Unter Umständen führt dies zu nicht lösbaren Aufgabenstellungen. Wenn z.B. ein Autor einen leeren Absatz zwischen zwei Listenpunkte einschiebt, lässt sich nicht mehr automatisiert entscheiden, ob es sich um zwei

9 SmartDocuments – Grundlagen

Die bislang gezeigten Möglichkeiten der Integration von XML in Microsoft Word bezogen sich auf Im- und Exportfunktionalitäten. Obwohl dies bereits ein wichtiger Aspekt für die Wiederverwend- und Austauschbarkeit von Word-Dateien ist, geht Microsoft mit den SmartDocuments noch einen Schritt weiter. Sie stellen eine neue Qualität in der Anwendungsprogramierung in Microsoft Word dar, ermöglichen eine intuitive Benutzerführung und erweitern das Leistungsspektrum von Word-Dokumenten um ein Vielfaches.

9.1 Was sind SmartDocuments?

WordML ist für viele Anwendungsbereiche von Word ein wesentlicher Fortschritt. Die Offenheit des Datenformats, die einfache programmatische Erstellung neuer Word-Dateien aus XML-Instanzen mit Hilfe eines einfachen Stylesheets etc. sind hier zu nennen. Allerdings verfügt WordML über keine Möglichkeit, eine Datei semantisch zu beschreiben. Damit ist gemeint, dass WordML, ähnlich wie HTML auch, z.B. eine Adresse nicht von einer Unterschrift unterscheiden kann. Das ist auch nur natürlich, denn es ist selbstverständlich unmöglich, in einer Spezifikation alle jemals zu erwartenden semantischen Sinneinheiten mit vorgefertigten Elementen zu beschreiben. Aus diesem Grund ist HTML zu XML erweitert worden: XML ermöglicht ja gerade, ein eigenes Vokabular zu entwerfen und für eine gegebene Situation einzusetzen, ohne die Standardtechnologie zu verlassen.

Ebenso gestattet es Word dem Anwender, zusätzlich zu den XML-Elementen von WordML auch noch eigene XML-Elemente einzufügen. Auf diese Weise ist es erstmals möglich, Word zur Erzeugung strukturierter Dokumente zu nutzen. Denn: Word konnte zwar bislang schon strukturierte Dokumente erzeugen, deren Durchsetzung allerdings nicht erzwingen. Damit ist gemeint, dass Word zwar mit Hilfe der Absatzformate gut strukturierte Dokumente erzeugen kann, allerdings keine Funktionalität mitbringt, um zu erzwingen, dass ein Autor diese Absatzformate auch tatsächlich benutzt, oder gar die Reihenfolge des Einsatzes zu überwachen. So ist es also ohne Weiteres möglich, mit dem Absatzformat

Überschriftsebene 4 zu beginnen oder dieses Absatzformat einzusetzen, um ein Zitat zu formatieren, ohne dass Word dagegen etwas tun kann.

Eine solche semantische Auszeichnung ist nun mit Hilfe von eigenen XML-Elementen möglich. Diese Elemente müssen gegen ein mit der Vorlage in Word verbundenes Schema validieren und zeigen so eine möglicherweise fehlerhafte Struktur des Inhalts an. Außerdem kann Word so parametriert werden, dass die Speicherung von Dateien, die nicht gegen dieses Schema validieren, unmöglich ist. Mit diesen beiden Mitteln kann Word strukturierte Dokumente erzwingen. Word unterstützt die Verwendung der eigenen XML-Elemente hierbei durch einen eigenen Arbeitsbereich *XML-Struktur*, mit deren Hilfe sich die Elemente anzeigen lassen, die gemäß der verbundenen Schemabeschreibung benutzt werden können.

Dennoch bleiben einige Wünsche für den Anwender offen:

- Der Arbeitsbereich *XML-Struktur* ist sehr technisch geraten und erfordert vom Anwender das Wissen darüber, was XML ist und wie es eingesetzt wird. Außerdem ist der Zugriff auf die XML-Attribute nur sehr mühsam möglich, da kein eigenes, ständig verfügbares Eingabefenster für die Attribute angeboten wird. Stattdessen muss für ein XML-Element jedes Mal über die rechte Maustaste ein zusätzlicher Dialog mit dieser Funktionalität aufgerufen werden.
- Der Arbeitsbereich *XML-Struktur* gibt keine Information darüber, in welcher Reihenfolge eines von mehreren Kindelementen in das Dokument eingefügt werden muss. Dies hat zur Folge, dass Dokumente trotz gut gemeinter Bedienung Validierungsfehler aufweisen können.
- Auch wenn die Eingabe der XML-Elemente über einen Dialog schon einen Fortschritt darstellt, so ist der Alltag der Autoren von strukturierten Dokumenten doch oft auch davon geprägt, dass schon bereits digital vorliegende Informationen, etwa aus Datenbanken oder anderen Dokumenten, wiederverwendet werden sollen. Diese Daten müssen separat zu den Strukturierungsinformationen des Arbeitsbereichs *XML-Struktur* erfasst werden und komplizieren damit die Bedienung weiter.

SmartDocuments

Hier kommen nun die SmartDocuments ins Spiel. Der Grundgedanke ist: Wenn eine Word-Datei mit XML-Elementen semantisch ausgezeichnet ist, kann Word mit Hilfe der aktuellen Position des Cursors die umgebenden Elemente erkennen und eine Benutzeroberfläche anbieten, auf der alle benötigten Mittel verfügbar gemacht werden können, um leicht mit XML-Elementen, externen Daten etc. zu arbeiten und diese einfach in die Word-Datei einfügen zu können.

Interessante Szenarien werden dadurch denkbar: So aktiviert ein Klick in den Bereich der Adressaten-Angaben ein ActiveX-Steuerelement, das

Adressen aus einer Datenbank einlesen und formatiert in das Dokument einfügen kann. Textbausteine für die Erstellung intelligenter Antwortschreiben werden in Einblendmenüs vorgehalten, Rechnungsformulare schlagen den Umrechnungskurs von Fremdwährungen interaktiv in einem Web-Service nach und übergeben ihn dem Dokument zur Berechnung.

Allerdings fordern SmartDocuments einen Preis: Die zusätzliche Funktionalität muss der Vorlage hinzuprogrammiert werden. Dafür ist es erforderlich, in einer der unterstützten Programmiersprachen (VB6, VB.NET oder C#.NET) gegen das Word-Objektmodell zu programmieren. Der Aufwand ist zwar der Programmierung eines VBA-gestützten Makros vergleichbar, allerdings nicht in VBA selbst möglich. Außerdem sind eine Reihe weiterer Schritte notwendig, um aus einer einfachen Dokumentvorlage ein SmartDocument zu machen. Zudem ist für die Nutzung dieser Funktionalität das Office-Paket in der Professional-Version erforderlich: Nur diese Version gestattet die Integration von eigenen XML-Elementen in ein Word-Dokument, was die Voraussetzung für SmartDocuments darstellt.

Auf der anderen Seite steht ein Funktionsreichtum bereit, der in dieser Form mit Makroprogrammierung nicht erreicht werden kann: SmartDocuments machen es möglich, gänzlich neue Konzepte der Benutzerführung umzusetzen. Anstatt modale Dialoge anzuzeigen, wird dem Benutzer immer nur die Information angezeigt, die er im Moment braucht. Nicht benötigte Funktionen werden ausgeblendet. Dies alles erlaubt erstmals die Erzeugung und Validierung strukturierter Dokumente in Word.

SmartDocuments machen aus Word jedoch keinen XML-Editor. Die Möglichkeiten sind eher auf die Unterstützung zur Erzeugung schwach bis mäßig komplex strukturierter Dokumente zugeschnitten. Für datenzentrierte Dokumente ist das neue Office-Mitglied InfoPath, ein Programm zur Erzeugung und Bearbeitung von elektronischen Formularen, die bessere Alternative. Auf der anderen Seite bleibt die Erzeugung komplex strukturierter Dokumente, zum Beispiel aus dem Bereich der technischen Dokumentation, nach wie vor eine Domäne von Spezialprogrammen. SmartDocuments eignen sich also vor allem dort, wo mit zusätzlichem Aufwand bisherige Dokumentvorlagen in ihrer Funktionalität und Bedienbarkeit erweitert werden sollen. Sie stellen eine neue Möglichkeit als Ergänzung zur Makroprogrammierung in VBA dar, nicht mehr, aber auch nicht weniger.

9.2 Einführung in die verwendeten Technologien

Für diejenigen Leser, die sich einen intensiveren Überblick über SmartDocuments verschaffen möchten, folgt eine kurze Einführung in einige Schlüsseltechnologien, soweit sie für das Verständnis der Arbeitsweise

von SmartDocuments von Relevanz sind. Ausgenommen bei dieser Übersicht ist die Programmiersprache Visual Basic selbst, da dies den Rahmen des Buches bei weitem sprengen würde. Leser, die mit dem Objektmodell von Word und den Grundlagen von Visual Basic bzw. der damit verbundenen Techniken vertraut sind, können diese Einführung getrost überspringen.

Das Word-Objektmodell

Word ist ein Programm, das die Möglichkeit bietet, jede Aktion, die der Benutzer mit Hilfe von Menübefehlen ausführen kann, auch über ein im Hintergrund laufendes Programm auszuführen. Der bekannteste Weg, diese Art des Zugriffs zu benutzen, stellen sogenannte Makros dar. Word bietet für einfache Aufgaben einen Makro-Rekorder an, der eine Folge von Anwendungsbefehlen aufzeichnen, mit einem Makronamen versehen und beliebig oft ausführen kann. Doch die Funktionalität der Makros ist nicht auf das Aufzeichnen und Wiederholen von Benutzeraktionen beschränkt. Word liefert eine Programmiersprache mit, die den Funktionsreichtum von Makros beinahe beliebig erweitern kann: Visual Basic for Applications, kurz VBA.

Visual Basic for Applications (VBA)

Mit VBA können zusätzlich zu den eingebauten Funktionen von Word eigene Funktionen, Dialoge und Entscheidungsbäume definiert werden, die Word als vollständig programmierbare Anwendung erscheinen lassen und vielfältige Spezialanwendungen möglich machen. Dem erfahrenen Makroprogrammierer reichen hierzu die Möglichkeiten des Makrorekorders nicht aus, er programmiert VBA direkt, indem er den VBA-Editor aus Word benutzt. Dieser Editor kann über die Tastenkombination `Alt-F11` aufgerufen werden und steht in jeder Version von Word zur Verfügung.

Dieser Editor gestattet es, Programme zu erstellen, die eigene Logik mit den Funktionen aus dem Word-Objektmodell verbindet, in dem alle Funktionen von Word in einer hierarchischen Struktur verfügbar gemacht werden. Die gesamte Struktur ist bei weitem zu komplex, um sie hier darzustellen, es sollen nur die wesentlichen Einstiegspunkte benannt werden.

Application- und Document-Objekt

Grundlage der Programmierung sind zwei Objekte, von denen das eine das andere enthält: das `Application`-Objekt (die Word-Applikation selbst) und das `Document`-Objekt (das eigentliche Word-Dokument). Die `Application` enthält Funktionen, die für Word als Ganzes gelten (z.B. die Menüleisten, Erweiterungen und viele weitere Dinge). Von besonderem Interesse für dieses Kapitel ist die Auflistung aller Dokumente, die derzeit in Word geöffnet sind. Diese Dokumentenliste enthält immer genau ein aktives Dokument, das `ActiveDocument`. Das `ActiveDocument` ist das Dokument, in dem sich derzeit der Cursor befindet und dessen Fensterleiste

10 Erstellung einer SmartDocument-Solution

Dieses Kapitel schildert die Details der Erzeugung von SmartDocument-Solution. Leider müssen ab hier gewisse Voraussetzungen an den Kenntnisstand des Lesers gemacht werden. So sollte der Leser Kenntnisse in der Programmiersprache Visual Basic und grundlegenden Kenntnisse des Word-Objektmodells besitzen. Nach Lektüre der einleitenden Bemerkungen zu den verwendeten Technologien sollten Sie zwar das Prinzip der Umsetzung verfolgen können, im Detail werden aber Fragen offen bleiben. Auf der anderen Seite ist das detaillierte Verständnis dieses Kapitels auch nur für die ernsthaft an einer Erzeugung von SmartDocuments Interessierten relevant. Dieser Leserkreis muss die angesprochenen Kompetenzen allerdings ohnehin erwerben.

10.1 Voraussetzungen

Word erfordert einige Voraussetzungen, um mit SmartDocument-Solutions arbeiten zu können:

- Zunächst einmal ist Word 2003 in der Professional Edition erforderlich, da erst ab Version 2003 die Unterstützung von XML eingeführt wurde. Die Erstellung eigener XML-Elemente, basierend auf einem Schema, ist eine Option der Professional Edition.
- Soll die SmartDocument-Solution mit Hilfe von VB.NET oder C#.NET realisiert werden, sind zusätzlich folgende Komponenten notwendig:
  - .NET-Framework Version 1.1 oder höher
  - .NET-PIA (Primary Interop Assembly) für Office. Dieser Code wird automatisch mitinstalliert, wenn das .NET-Framework *vor* der Installation von Office bereits installiert war. Die PIAs können allerdings auch jederzeit nachinstalliert werden.

    Sollte das .NET-Framework in Version 2.0 eingesetzt werden, kann es erforderlich sein, ein Service-Pack von Microsoft für

Word einzuspielen, das den Aufruf von SmartDocuments in Word und Excel ermöglicht[1].

Es hat sich gezeigt, dass die Reihenfolge der Installation wichtig sein kann. Installieren Sie am besten zunächst das .NET-Framework in der Version, in der es später eingesetzt werden soll, und erst anschließend eine möglichst vollständige Version des Office-Pakets. Nicht benötigte Programmpakete nachträglich zu entfernen, hat sich als sinnvoller herausgestellt, als die Installation von vornherein nicht vorzunehmen.

10.2 Planung einer SmartDocument-Solution

SmartDocument-Solution planen

Eine SmartDocument-Solution erfordert eine genaue Planung. Dies betrifft insbesondere die Funktionalität der Lösung. Aufgrund des etwas umständlichen `ISmartDocumentInterface` gestalten sich spätere Änderungen sehr aufwändig. Daher sollte das Spektrum der benötigten Funktionen so genau wie möglich festgelegt werden. Aus diesem Spektrum von Funktionen ergibt sich, auf welche Bereiche des Dokuments die SmartDocument-Solution mit erweiterter Funktion reagieren muss. Diese Bereiche definieren die minimale Anzahl der in das Dokument einzubringenden XML-Elemente. Eventuell macht es Sinn, mehr Elemente vorzusehen als zunächst benötigt, vielleicht, um den Inhalt mittels Stylesheets für andere Umgebungen vorbereiten zu können (z.B. DocBook), oder aber, um Raum für zukünftige Erweiterungen der SmartDocument-Solution zu schaffen.

XML-Schema definieren

Ist die Anzahl und die Struktur der XML-Elemente bekannt, muss zunächst ein XML-Schema definiert werden, das die Elemente enthält, an die später die Funktionalität gekoppelt werden soll. Ohne ein XML-Schema lassen sich keine XML-Elemente in das Dokument einfügen und dementsprechend auch keine Bereiche festlegen, die mit einer Funktion verknüpft werden können. Dieses Schema sollte im Interesse einer erfolgreichen Implementierung der SmartDocument-Solution nicht zu komplex geraten. Verwenden Sie eine begrenzte Schachtelungstiefe, benennen Sie alle XML-Elemente eindeutig und machen Sie von den erweiterten Funktionen wie Ersetzungsgruppen, Vererbung etc. keinen Gebrauch.

Implementierung des `ISmartDocument Interface`

Ist dieses Schema durchdacht und erstellt, geht es daran, das `ISmartDocumentInterface` zu implementieren. Dieses Interface kann in allen unterstützten Programmiersprachen implementiert werden und stellt eine Liste von Methoden und Konstanten zur Verfügung, die für das Projekt benötigt werden. Leider ist die Implementierung dieses Interface eine langwierige und fehleranfällige Angelegenheit. Unverständlich ist, dass

[1] Leider ändern sich die Links häufiger, als Bücher gedruckt werden. Sie sollten in jedem Fall ein aktuelles Release von Office verwenden.

Informationen eingebracht werden müssen, deren Ermittlung normalerweise durch den Computer erreicht werden sollte, wie die Anzahl der eingesetzten Steuerelemente und andere Angaben. Hier zahlt sich ein schlankes XML-Schema besonders aus, da auf diese Weise die Verwaltung des Interface in erträglichen Grenzen gehalten werden kann.

Funktionalität programmieren

Durch die Implementierung des `ISmartDocumentInterface` ist die Topologie fertig gestellt und man kann daran gehen, die eigentliche Funktionalität in die beiden Event-Handler-Gruppen `OnPopulate` und `Action` einzufügen. Es empfiehlt sich, von vornherein eine weitere Codedatei mit den Methoden zu erstellen, in denen die eigentliche Arbeit geleistet wird. Diese Methoden werden dann aus dem `ISmartDocumentInterface` aufgerufen. Diese Methoden in das Interface zu integrieren, verschlechtert nur die Übersichtlichkeit und erschwert somit Entwicklung und Wartung der SmartDocument-Solution.

Lösung konfektionieren und kompilieren

Als letzter Schritt ist es erforderlich, Hilfsdateien, Grafiken, Filme oder sonstige externe Daten zur Verfügung zu stellen, damit die SmartDocument-Solution komplett funktionsfähig wird. All diese Dateien werden anschließend durch die Entwicklungsumgebung zu einem Ganzen verbunden, alle Pfade in das Manifest eingetragen, getestet und ausgeliefert. Sehen wir uns dazu die einzelnen Schritte etwas genauer an.

10. 3 Das XML-Schema

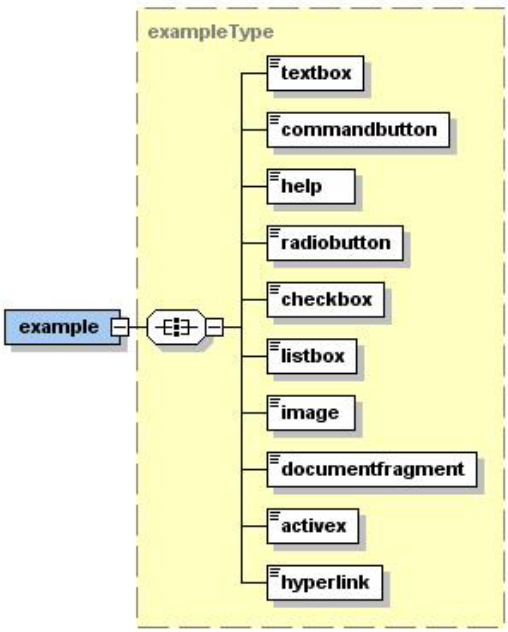


Abb. 10-1
Das XML-Schema des Beispieldokuments

SmartDocument-Solutions benötigen ein XML-Schema nach W3C-Spezifikation, um anwenderspezifische XML-Elemente in das Dokument einfügen zu können. Gleichzeitig werden die Dokumente standardmäßig gegen dieses Schema validiert. Es ist verlockend, für die Erstellung von SmartDocument-Solutions ein Schema zu verwenden, das bereits als Industriestandard vorliegt, wie etwa DocBook. Allerdings werden solche Versuche im Regelfall nicht zum gewünschten Erfolg führen. Es empfiehlt sich im Gegenteil, kleine, kompakte Schemata zu entwerfen und darauf zu achten, dass alle Elemente eindeutig benannt werden, unabhängig von ihrer Position in der Schemahierarchie.

In diesem Buch wird eine angepasste Version der Beispieldateien verwendet, die mit dem SmartDocument Development Kit geliefert werden (siehe Abbildung 10-1). Dies hat seinen Grund darin, dass dieses Beispiel, frei von einem aufgesetzten und meistens nicht sehr überzeugenden Einsatzfall, die Vielfalt der Möglichkeiten konzentriert zeigt. Dennoch ist es vielleicht sinnvoll, einige Anregungen zu geben, in welchen Umfeldern SmartDocument-Solutions eingesetzt werden können:

Editor für strukturierte Dokumente

Jeder Verantwotliche für eine größere Publikation kennt das Problem, eine gut strukturierte Datei aus einzelnen Word-Dateien verschiedener Autoren erzeugen zu müssen. Verschiedene Formatierungen mit allen möglichen (und unmöglichen) Mitteln von Word, inkonsistente Verwendung von Absatzformaten, Tabulatoren, Tabellen, Nummerierungen etc. können einem das Leben zur Qual machen. Hier wäre es denkbar, dem Anwender eine SmartDocument-Solution zur Verfügung zu stellen, die Formatierungen nach gewissen Regeln einfügt und alle anderen Formatierungsmöglichkeiten ausschließt.

Intelligente Word-Vorlagen

In beinahe allen Geschäftsbereichen existieren wiederkehrende, strukturierte Dokumente, die auf die jeweilige Situation angepasst werden müssen. Ob es sich um Support-Schreiben, Bestellungen, personalisierte Akquisitionsschreiben oder einfache Vertragstexte handelt, immer müssen bereits vorhandene Informationen aus Datenbanken in bestehende Dokumente eingepflegt werden. Hier können SmartDocument-Solutions Informationen über Web-Services, Datenbankanbindungen oder hinterlegte Textelemente zur Verfügung stellen und durch den Benutzer einfügen lassen. Hauptvorteil gegenüber einer Makroprogrammierung ist, dass im Regelfall keine modalen Dialoge erforderlich sind, die den Benutzer zu einer bestimmten Arbeitsweise zwingen. Im Gegenteil werden dem Benutzer lediglich dann Informationen angeboten, wenn sie aufgrund der Position des Cursors sinnvoll sind.

10.4 Die SmartDocument-Steuerelemente

Die SmartDocument-Solution bietet eine Auswahl verschiedener Steuerelemente an, die in den Dokumentaktionen angezeigt werden können. Es sind ausschließlich die im Folgenden genannten Steuerelemente verfügbar. Leider ist auch die Kontrolle über Anzeige und Anordnung der Steuerelemente auf den Dokumentaktionen begrenzt, so dass professionelle Applikationen wohl kaum um die Entwicklung von ActiveX-Steuerelements herumkommen, um die auferlegten Beschränkungen zu umgehen. Um die Steuerelemente intern eindeutig auswählen zu können, definiert das `ISmartDocumentInterface` Konstanten, die im Code referenziert werden müssen. Die folgende Liste gibt einen Überblick über die verfügbaren Steuerelemente und deren Konstanten, so wie sie das `ISmartDocument-Interface` definiert:

- ActiveX

  Verweis auf ein Steuerelemente, das als ActiveX-Steuerelement (*.ocx) hinterlegt wurde. Erweitert die Funktionalität über das von SmartDocument definierte Maß hinaus. ActiveX-Steuerelemente werden in Visual Basic 6 erzeugt und publiziert und stellen oftmals die einzige Möglichkeit dar, komplexere Formulare zu erzeugen. Smart-Document-Steuerelemente können z.B. nicht nebeneinander dargestellt werden und sind deshalb schon bei einfacheren Adressformularen nicht mehr zufriedenstellend einsetzbar. ActiveX-Steuerelemente sind lediglich in Bezug auf den zur Verfügung gestellten Platz begrenzt. Ansonsten kann das gesamte Funktionsspektrum von VB6 zum Tragen kommen. Es gilt natürlich die gleiche Anforderung an diese Steuerelemente, die generell an ActiveX-Steuerelemente in Word zu stellen ist: Saubere Programmierung ist Pflicht, da ein abstürzendes ActiveX-Steuerelement Word im Regelfall mit sich reißen wird.

  Konstante: C_TYPE.C_TYPE_ACTIVEX

- Button

  Eine Schaltfläche, z.B. für "OK" oder "Abrechen".

  Konstante: C_TYPE.C_TYPE_BUTTON

- Checkbox

  Ein Ankreuzfeld, das mehrere aktivierte Ankreuzfelder in einer Gruppe zulässt. Wird normalerweise für parallel verfügbare Optionen verwendet.

  Konstante: C_TYPE.C_TYPE_CHECKBOX

- Combo

  Ein Einblendmenü, das es gestattet, aus einer Liste von Optionen einen Wert auszuwählen.

  Konstante: C_TYPE.C_TYPE_COMBO

- DocumentFragment

  Ein Dokumentfragment in WordML. In diesem Fall ist eine gültige WordML-Instanz gemeint, die den benötigten Text enthält. Eine *gültige WordML-Instanz* ist eine WordML-Datei (kein Fragment!), die gegen das WordML-Schema validiert. Es benötigt insbesondere die Definitionen aller eingesetzten Absatz-, Schrift- oder Nummerierungsformate sowie ein Minimalset an WordML-Elementen, die die eigentliche Nutzinformation umgeben. Eignet sich nur für sehr kurze und statische Texte. Normalerweise wird das folgende Steuerelement DocumentFragmentURL verwendet.

  Konstante: C_TYPE.C_TYPE_DOCUMENTFRAGMENT

- DocumentFragmentURL

  Ein Verweis auf ein Dokumentfragment, das unter dem URL, der in diesem Steuerelement hinterlegt ist, erreichbar ist. Auf diese Weise können umfangreiche Textpassage in das Dokument eingefügt werden.

  Konstante: C_TYPE.C_TYPE_DOCUMENTFRAGMENTURL

- Help

  Dieses Steuerelement bietet eine auf XHTML basierende Hilfedatei an. Im Vergleich zu XHTML sind allerdings nicht alle Elemente erlaubt. Die Liste der unterstützten XHTML-Elemente finden Sie in der Onlinehilfe des SmartDocument Development Kit. Achten Sie bitte auch auf das *X* in XHTML, das eine strengere Schreibweise der Hilfedateien vorschreibt.

  Konstante: C_TYPE.C_TYPE_HELP

- HelpURL

  Analog zum Dokumentfragment können mit diesem Steuerelement umfangreiche Hilfedateien im XHTML-Format aufgerufen werden.

  Konstante: C_TYPE.C_TYPE_HELPURL

- Image

  Ein Steuerelement, das Bilder anzeigen kann. Es enthält einen URL, der auf eine Ressource in einem der unterstützten Bildformate zeigt.

  Konstante: C_TYPE.C_TYPE_IMAGE

- Label

  Dieses Steuerelement hat keine aktive Funktion, sondern stellt eine Möglichkeit dar, Bereiche der Dokumentaktionen zu beschriften.

  Konstante: C_TYPE.C_TYPE_LABEL

11 Visual Studio Tools for Office 2005

Im vorangegangenen Kapitel über *SmartDocuments* haben wir bereits häufiger auf die Visual Studio Tools for Office 2005 (VSTO 2005) verwiesen. Sehen wir uns daher in diesem Kapitel den aktuellen Stand der Entwicklungen zur Erstellung sogenannter *Rich-Client*-Applikationen etwas genauer an.

VSTO 2005 schlägt eine Brücke zwischen der älteren COM-basierten und der neueren .NET-basierten Technologie. Zielsetzung war es, das Word-Objektmodell, das ja mit VBA dem Programmierer zur Verfügung steht, auch dem .NET-Programmierer bereitzustellen und dabei die neueren Programmierparadigmen der Objektorientierung zu unterstützen. Außerdem wurde versucht, die Nützlichkeit von Word insofern zu steigern, als Word als Teil der Vision *Rich Client* die Schaffung neuer Anwendungstypen unterstützen sollte.

11.1 Übersicht über die Visual Studio Tools for Office 2005

Überblick über VSTO 2005

Die Visual Studio Tools for Office 2005 (VSTO 2005) sprengen den durch die SmartDocuments aufgezogenen Rahmen bei weitem. Man kann sich eher vorstellen, dass mit VSTO 2005 die Programmierrichtung umgedreht wird: Wurde früher innerhalb von Word ein Makro geschrieben, das Word um eine Funktion erweiterte, so ist es heute eher so, dass Word innerhalb eines VSTO-2005-Programms als ein intelligenter Editor benutzt wird, der dem Programm Fähigkeiten zum Erstellen, Layouten und Drucken von Text zur Verfügung stellt. Die XML-Unterstützung, die bei den SmartDocuments noch zentral für die Funktionalität war, ist hier zu einem von vielen Mechanismen zur Interaktion zwischen Word und der VSTO-2005-Applikation geworden.

Word ist eine COM-Applikation. Dies bedeutet, dass der Quellcode von Word nicht im relativ neuen .NET-Framework entstanden und somit kein sogenannter *Managed Code* ist, sondern mit älteren Technologien wie z.B. VB6, C++ oder ähnlichen Programmiersprachen erstellt wurde.

Die Programmierung der Word-Applikation aus einer .NET-Applikation heraus ist aus diesem Grund nicht ohne Weiteres möglich. Die Datentypen, aber auch die grundsätzliche Ausrichtung des .NET-Frameworks gegenüber der älteren COM-Technologie sind so unterschiedlich, dass die Benutzung älterer COM-Applikationen nur über spezielle Schnittstellen erreicht werden kann.

PIA

Soll Word also aus dem .NET-Framework heraus programmiert werden (oder anders ausgedrückt: Möchte man VBA durch .NET ersetzen), ist es erforderlich, ein Interface zwischen der älteren COM-basierten Applikation und dem Klassenmodell des .NET-Frameworks zu schaffen. Diese Interface kann man durchaus auch selbst programmieren. Bei einer Applikation wie Word dürfte dies jedoch keine gute Idee sein. Aus diesem Grund stellt Microsoft eine solche Schnittstelle zur Verfügung: die sogenannte *PIA (Primary Interoperability Assembly)*. Der Begriff *Primary* deutet bereits darauf hin, dass es durchaus auch weitere *Assemblies* (.NET-Programme) geben kann, die weitere Interface bereitstellen.

Um die PIA für Word nutzen zu können, muss sie zunächst allerdings installiert werden. Hier verhält sich Word relativ trickreich: Ist zum Zeitpunkt der Installation des Office-Pakets das .NET-Framework auf diesem Rechner installiert, bietet das Installationsprogramm die *Programmierunterstützung für .NET* für die einzelnen Office-Pakete an. Standardmäßig sind diese Pakete zur Installation bei der ersten Verwendung ausgewählt. Ist zum Zeitpunkt der Installation des Office-Pakets jedoch kein .NET-Framework installiert, wird die Installation dieser PIAs erst gar nicht angeboten. Nach der Installation des Frameworks muss diese Installation dann manuell nachgeholt werden[1]. Die PIAs werden bei erfolgreicher Installation im sogenannten *GAC (Global Assembly Cache)* des .NET-Frameworks verwaltet und stehen unter dem Namensraum `Microsoft.Office.Interop.Word` zur Integration in .NET-Projekte zur Verfügung. Dieser erste, wichtige Schritt macht das Word-Objektmodell für .NET verfügbar. So kann in .NET also z.B. eine Variable vom Typ `Microsoft.Office.Interop.Word.Application` definiert werden, die das gesamte Objektmodell von Word zur Verfügung stellt, wie im folgenden Codeabschnitt in C# gezeigt:

```
using Word = Microsoft.Office.Interop.Word;
Word.Application app = new Word.Application();
```

Mit diesem Schritt ist es grundsätzlich möglich, eine SmartDocument-Anwendung wie in Kapitel 9 *SmartDocuments — Grundlagen* beschrieben, auf Basis von .NET-Code aufzubauen. Dies ist ein möglicher, wenn

[1] Die PIAs stehen auch separat zum Herunterladen zur Verfügung. Zwar ändern sich die Adressen häufiger, doch können Sie Informationen über PIAs unter *http://msdn2.microsoft.com/en-us/library/aax7sdch.aspx* erhalten.

auch steiniger Weg, denn nach wie vor muss der gesamte Weg der Implementierung des `ISmartDocumentInterface` beschritten werden. Hier kommen nun die VSTO 2005 ins Spiel. Basierend auf der PIA bieten die VSTO 2005 eine Reihe eigener Möglichkeiten an, um eine leistungsfähige Applikation aufzubauen. Dabei kann man sich die VSTO 2005 wie eine Klarsichtfolie über dem Word-Dokument vorstellen. Sie bilden eine eigene Schicht in eigenen Namensräumen (`Microsoft.Office.Tools`, `Microsoft.Office.Tools.Word` bzw. `Microsoft.Office.Tools.Word.Controls`) und interagieren mit Hilfe der PIA mit dem Word-Objektmodell. Auf diese Weise werden viele Funktionen von Word erweitert und auf leistungsfähige Weise für das .NET-Framework verfügbar gemacht.

11.2 Relevante Technologien

Analog zum Kapitel 9 *SmartDocuments — Grundlagen* möchten wir einige zentrale Technologien erläutern, die zum Verständnis der folgenden Kapitel erforderlich sind. Wie auch im vorhergehenden Kapitel sollen dabei die Begriffe nur soweit in ihren Grundzügen erläutert werden, wie es zum Verständnis unbedingt notwendig ist. Selbstverständlich kann der Leser, der mit diesen Begriffen vertraut ist, diesen Abschnitt überspringen.

Rich Client

Dieses neue Modewort bezeichnet die Quintessenz aus den bislang einander widersprechenden Vorstellungen zur sinnvollen Programmierung von Anwendungen: Auf der einen (älteren) Seite die Client-Server-Architektur, bei der ein umfangreiches Programm auf dem Rechner des Endanwenders installiert wird und mit einer zentralen Applikation, meist einer Datenbank, kommuniziert, und auf der anderen Seite das Paradigma der weborientierten Anwendung, bei der auf dem Rechner des Endanwenders lediglich ein Browser installiert sein muss und die komplette Anwendung an wenigen Stellen im Netzwerk verfügbar gemacht wird.

Leider ist die Argumentation für den weborientierten Ansatz oftmals einseitig zu Gunsten des Netzwerkadministrators geführt worden, da insbesondere die Vorteile, keine Anwendungen mehr installieren zu müssen, herausgestellt wurden. Dem steht jedoch gegenüber, dass der Endanwender im Regelfall mit schlechteren Bedienoberflächen und einer mangelhaften Interaktionsfähigkeit mit seinen bekannten Werkzeugen (Excel, Word etc.) die Rechnung zahlen musste.

Hier setzt die Vision eines Rich Client an, bei der ein lokal vorhandenes Werkzeug wie Word so mit kundenspezifischer Funktion erweitert werden soll, dass sich insgesamt eine ideale Kombination aus Funktionalität und leichter Administration ergibt. Inwiefern diese Ideen den Köpfen der Marketingleiter bei Microsoft entsprungen oder tatsächlich eine rele-

vante Programmierstrategie darstellen, bleibt dahingestellt. Sehr wahrscheinlich ist allerdings, dass eine Generation von Software diesem Paradigma entsprechen muss, um verkäuflich zu sein ...

.NET-Programmierung bzw. Managed Code im Gegensatz zu COM-Anwendungen

Das .NET-Framework ist eine Technologie, mit deren Hilfe Programme auf .NET-fähigen Computern (derzeit nur Windows-Computer) ausgeführt werden. Es ist schwierig, in ein paar Sätzen den Unterschied des .NET-Frameworks gegenüber *normalen* Programmen zu erklären. Vielleicht ist es ausreichend, zu erläutern, dass das .NET-Framework von Grund auf neu konstruiert wurde, um neueste Programmierparadigmen (allen voran die Objektorientierung) dem Programmierer zur Verfügung zu stellen. Im .NET-Framework sind verschiedene Programmiersprachen verfügbar (VB.NET, C# etc.), die, vergleichbar zum sehr ähnlichen Java, in einen *Zwischencode* übersetzt werden, der dann vom Computer ausgeführt werden kann. Die Unterschiede zu herkömmlichen Programmierumgebungen auf Windows-Computern sind immens und fordern dem Entwickler eine längere Lernkurve ab. Leider ist Microsoft selbst bei der Programmierung seiner zentralen Anwendungen (insbesondere interessiert hier das Office-Paket) noch nicht dazu übergegangen, das eigene .NET-Framework zu nutzen. Vielleicht lässt sich dies aber für die Zukunft erwarten.

Im Gegensatz zu dieser recht neuen Technik wurde früher das sogenannte COM-Modell benutzt. Es ist im Vergleich zu .NET weniger modern und insbesondere durch seine mangelhaften Sicherheitsfunktionen in Verruf geraten.

C#

Basierend auf der Programmiersprache C sind viele der heute relevanten Programmiersprachen entstanden. Auf der einen Seite gibt es eine Evolution von C-Sprachen, die durch +-Zeichen kenntlich gemacht werden: C + bzw. C++. Auf der anderen Seite ist auch die Sprache Java (grob vereinfacht) eine 80/20-Evolution von C: 80% der Funktionalität, 20% der Komplexität von C. C# ist das Microsoft-Pendant zu Java (Kritiker haben *C# JAVA 3.0* genannt) mit dem kleinen, für Musiker offensichtlichen Witz, dass in der amerikanischen Notierung von Musik der Ton Cis (einen Halbton über C) als C# geschrieben wird ...