Mit CD

MS SQL Server + + + Gimp + + + Extreme Programming

S&S

entwickler magazin

entwickler magazin

Software, Systems & Development

6.07 | November/Dezember | www.entwickler-magazin.de | Deutschland € 6,50 Österreich € 7,00 Schweiz sFr 13,40

Enterprise Mashups

Neue Generation agiler Business-Anwendungen

Oracle 11g ▸ Neue Features für Administratoren und Entwickler

▸▸DB2 on Rails

Code-Analyse & Fehlersuche

XML Trend: DITA Open Toolkit für Dokumentationen nutzen

CD-Inhalt

Vollversion: Intrexx 4.0
inkl. dreier Lizenzen und Workshop im Heft

Testversionen
Sybase iAnywhere SQL Anywhere 10
XSL Formatter V4.2
Videotraining Visual C#

Tools
FOP 0.94
JAMP
XQilla 1.1

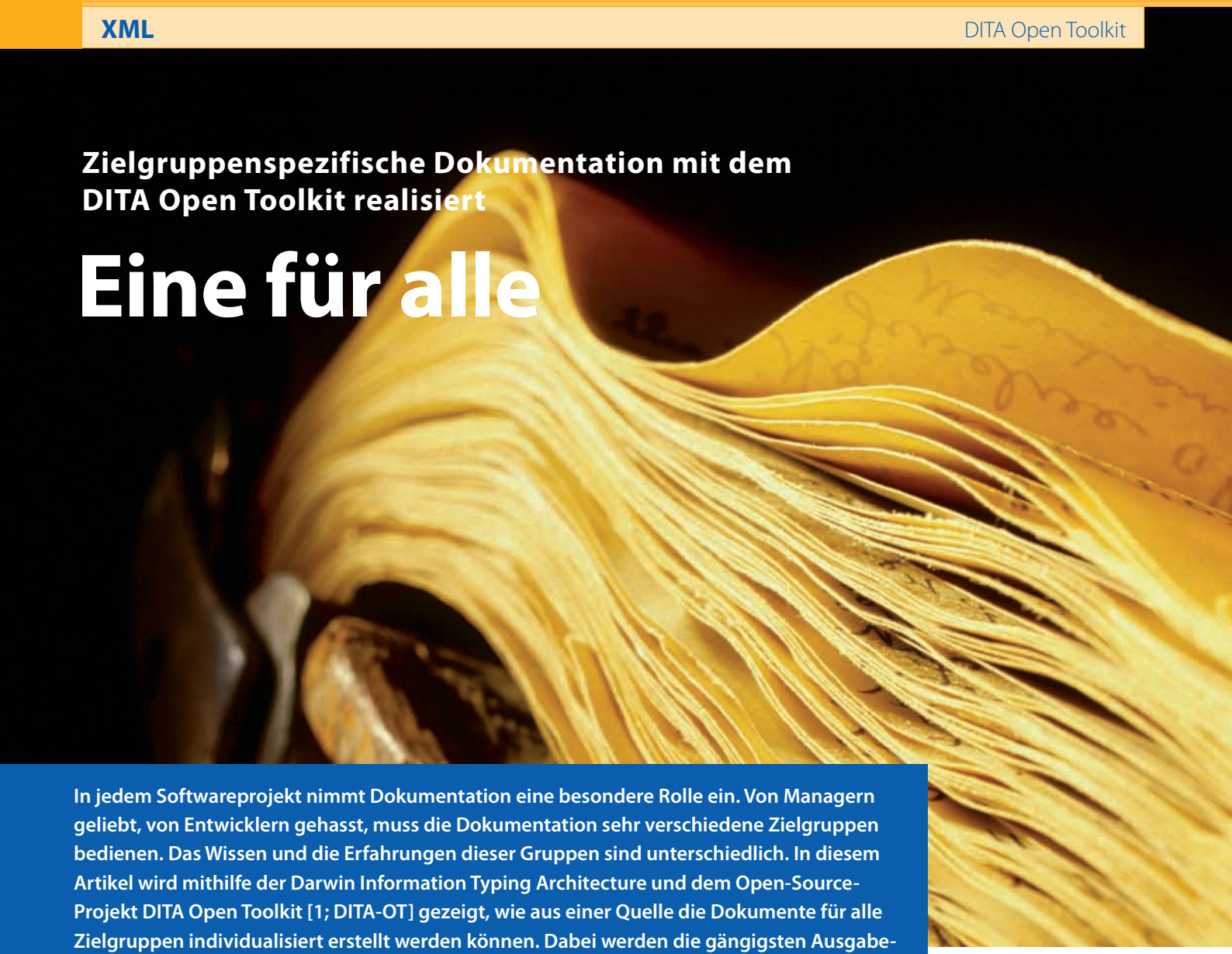
▸ Alle Infos auf Seite 3

Datenträger enthält nur Lehr- oder Infoprogramme

4 194156 606509 06

Zielgruppenspezifische Dokumentation mit dem DITA Open Toolkit realisiert

Eine für alle



In jedem Softwareprojekt nimmt Dokumentation eine besondere Rolle ein. Von Managern geliebt, von Entwicklern gehasst, muss die Dokumentation sehr verschiedene Zielgruppen bedienen. Das Wissen und die Erfahrungen dieser Gruppen sind unterschiedlich. In diesem Artikel wird mithilfe der Darwin Information Typing Architecture und dem Open-Source-Projekt DITA Open Toolkit [1; DITA-OT] gezeigt, wie aus einer Quelle die Dokumente für alle Zielgruppen individualisiert erstellt werden können. Dabei werden die gängigsten Ausgabemedien wie PDF, Windows-Hilfe, HTML und Eclipse Help bedient.

von Gernot Glawe

Zur Aufbereitung der Informationen wird meist der klassische Weg beschritten, für jede Zielgruppe wie Fachbereich, Support, Wartungsprogrammierer eigene Dokumente zu erstellen. Das Prinzip der Wiederverwendung, auch DRY (Don't repeat yourself) genannt, kommt hier nicht zur Anwendung. So wird unnötiger Aufwand betrieben, um die verschiedenen Zielgruppendokumentationen synchron zu halten. Doch es geht auch anders.

Zielgruppenakzeptanz

Andreas Rüping beschreibt in [2] ein Pattern mit der Problemstellung: „Wie kann ein Projektteam sicherstellen, dass die produzierte Dokumentation akzeptiert und angenommen wird?" Zuerst einmal muss dabei die Zielgruppe klar definiert sein! Wenn ich nicht weiß, für wen ein bestimmtes Dokument erstellt wird, kann ich auch nicht sagen, welches Vorwissen beim Leser besteht und welche Anforderungen der Leser an das Dokument stellt. Wenn sich herausstellt, dass das Dokument mehrere Zielgruppen zufrieden stellen soll, z.B. Manager und Programmierer, dann kommt es der Quadratur des Kreises nahe, alle Leser zufrieden zu stellen. Dem einen ist es zu ausführlich, dem anderen zu kurz. Man hat nun die Wahl, für jede Zielgruppe ein Extradokument zu erstellen oder keine Zielgruppe optimal zu bedienen. Rüping schlägt als Lösung vor: „Zuerst ist es am wichtigsten, dass jedes einzelne Dokument eine klar adressierte Leserschaft hat, und dass es diese Leser direkt adressieren muss, um sinnvoll zu sein". Die Folge ist, dass Informationen, die für alle Zielgruppen interessant sind, mehrfach erstellt werden. Bei der späteren Wartung der Dokumente muss dann für eine Änderung eines bestimmten Aspektes diese Änderung in allen Dokumenten einzeln aktualisiert werden. Wie wäre es, stattdessen ein Dokument zu erstellen, in dem alle Basisinformationen enthalten sind und zusätzlich die Informationen, die nur für bestimmte Zielgruppen relevant sind? Man könnte sich in einem manuel-

len Prozess vorstellen, dass der Schreiber der Dokumentation eines Programms bestimmte Abschnitte in seinem Word-Dokument grün färbt, damit diese dann nur in dem Dokument für Programmierer enthalten sind und in dem Dokument für die Zielgruppe „Manager" herausfliegen. Informationen nur für Programmierer wären also grün. Dann nehmen wir noch die Verkaufszahlen des Programms mit auf. Diese färben wir rot und vereinbaren dabei, dass rote Informationen nur für die Manager sind, also dann bei dem Dokument für die Programmierer rausgefiltert werden. Dann kann man die Abschnitte jeweils für die endgültigen Dokumente der Zielgruppe filtern. Es ergeben sich zwei Punkte:

- Generierung: Die endgültige Dokumentation wird (möglichst automatisch) aus einem Quelldokument generiert.
- Metadaten: Das sind Daten *über* den einzelnen Informationsabschnitt. Es gibt eine Möglichkeit, diese Metainformation „zielgruppenrelevant" an die eigentlichen Informationen anzuhängen.

Lösungssuche

Das bekannteste Werkzeug zum Erstellen von Dokumenten ist immer noch Word. Die Bedienung ist bekannt, es ist akzeptiert. Doch es gibt keine vernünftige Möglichkeit, Metadaten an die Absätze anzuhängen, außerdem ist die Automatisierung z.B. über COM/DCOM nicht problemlos. Daher wurde diese Möglichkeit verworfen. Ein allgemeiner Ansatz wäre es, die Informationen in eine XML-Datei zu packen und über Attribute Metainformationen dazuzumixen. Mit XSLT werden Ausgabedokumente erzeugt. Als Ablaufsteuerung wäre eine Batchdatei oder, etwas komfortabler, ein Build Script, z.B. mit Ant, denkbar. Mit forrest [3] oder DocBook [4] hätte man eine Basisfunktionalität. Die Filterung über Metainformationen müsste aber dazugebaut werden. Um das Rad nicht neu zu erfinden oder anders ausgedrückt, da wir uns im Zeitalter der postmodernen Programmierung befinden (siehe [5]), wurde weiter gesucht, ob auch Lösungen inklusive Filterfunktionalität bereits realisiert wurden. Die Recherche traf relativ schnell auf DITA, die *Darwin Information Typing Architecture*. Wie der Zufall es wollte, wurde kurze Zeit später im Newsletter von entwickler.press das Buch von Sissi Closs [6] vorgestellt. Mit den genauen Informationen aus dem Buch war schnell klar, dass DITA mit DITA-OT das Problem im Projekt lösen kann.

Dieter wer?

Die Darwin Information Typing Architecture wurde Ende der 90er Jahre von IBM für den Eigengebrauch entwickelt und 2004 Open Source an OASIS [7] übergeben. Die Haupttopicarten „Konzept" (*concept*), „Aufgaben" (*tasks*) und „Referenz" (*reference*) finden sich daher auch direkt in IBM-Produkten wie z.B. der Eclipse-Hilfe wieder (Abbildung 1).

Eine ausführliche Einführung und die Referenzdokumentation finden sich bei DITA-OT. Und DITA „eats its own dog food": Diese Dokumentation ist in DITA geschrieben und als PDF, als Windows-Hilfedatei *chm* und als HTML im Verzeichnis *doc* vorhanden.

Single Source Publishing SSP

Der eigentliche Fokus von DITA ist nicht die Zielgruppenorientierung, sondern das SSP (Single Source Publishing). Beim SSP wird Dokumentation in einer einzigen Quelle (Single Source) erfasst. Aus dieser Quelle werden verschiedene Ausgabemedien bedient. Klassische Beispiele für die Ausgabemedien sind HTML und PDF. Aber auch die Windows-Hilfe, JavaHelp oder die Eclipse-Hilfe sind denkbar.

Topics

Um Informationen in mehreren Dokumenten verwenden zu können, muss man Informationshäppchen festlegen und diese wieder verwendbar gestalten. Diese Häppchen sind in ihrer Struktur definiert und werden in XML abgelegt. So ein Häppchen wird dann *Topic* genannt. Ein einfaches Beispiel für den Topictyp *concept* zeigt Listing 1.

Das „Darwin" in DITA spielt auf die Möglichkeit der Vererbung an. Jeder Topictyp ist von dem Typ *topic* abgeleitet. Daher erbt *concept* die Eigenschaften von *topic*. In der DTD *concept.dtd* sind die gültigen Elemente des Typs beschrieben. Dadurch können XML-Editoren auch eine automatische Komplettierung

Abb. 1: Eclipse Help

der Elemente anbieten. Wie ist die Datei *beispiel.xml* aufgebaut? Jedes *topic* muss durch eine *id*, hier „beispiel" eindeutig gekennzeichnet werden. Die *id* wird zum Referenzieren des Topics verwendet. Über das *lang*-Tag wird die Sprache festgelegt. Dies ermöglicht es, in den verschiedenen Ausgaben die sprachabhängige Ausgabe um beispielsweise „Seite" statt „page" automatisch zu verwenden. Der *title* wird in XHtml als *title*-Tag, in PDF für das Inhaltsverzeichnis verwendet. Es folgen verschiedene strukturelle Abschnitte, um einleitende Worte zu schreiben, also *prolog* oder *abstract*. Im *conbody* geht es um die eigentliche Information. Innerhalb des *conbodys* kann man mit Paragraphen „<p>" oder *sections* weiter untergliedern. Eine detaillierte Beschreibung gibt es in der DITA-OT-Dokumentation. In dem Verzeichnis *doc* der DITA-OT-Installation findet sich die Datei *ditaref-book* als PDF und als Windows-Hilfe. Je nach Möglichkeiten des Ausgabemediums werden die verschiedenen Tags in der Ausgabe dargestellt. Das Layout ist getrennt vom Inhalt in *css*-Dateien definiert.

DITA-OT-Praxis

Wir nehmen für ein einfaches Beispiel an, dass die Installation eines Programms dokumentiert werden soll. Und weil wir uns

Listing 1

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE concept PUBLIC "-//OASIS//DTD DITA
                                    Concept//EN"
"../../dtd/concept.dtd">
<concept id="beispiel" xml:lang="de-de">
<title>Verwendung von DITA-ot</title>
<prolog></prolog>
<conbody><p> Das Grundprinzip ist es, die
    Informationen in kleine Häppchen zu verteilen.</p>
</conbody>
</concept>
```

vorher Gedanken über die Wartbarkeit der Dokumentation gemacht haben und DRY für uns nicht nur das Gegenteil von „wet“ ist, wählen wir DITA als Werkzeug. Unser Programm läuft nun auf mehreren Betriebssystemen. Ein realitätsnahes Beispiel: Windows und Unix. Die Installation des Programms sei für beide Systeme fast gleich, es gibt aber ein paar kleine Unterschiede. Unsere Zielgruppen sind also Windows-User und Unix-User. Soll man für die Installationsanleitung zwei verschiedene Texte pflegen? Copy-and-Paste? Natürlich nicht, denn das steht im Widerspruch zu „don't repeat yourself“! Die etwas konservativer formulierte Variante dieser Regel nennt man dann: Redundanzen vermeiden. Besser ist es, in einem einzigen Text bestimmte Absätze als Windows- bzw. als Unix-relevant zu kennzeichnen. Eben diese Möglichkeit bietet DITA mit der freien Implementierung DITA-OT. Doch zuerst müssen wir DITA-OT installieren.

Listing 2

```
<project name="beispiel_pdf" default="beispiel2pdf"
                                              basedir=".">
<property name="dita.dir" value="${basedir}${file.
                                          separator}..."/>
<import file="${dita.dir}${file.separator}integrator.
                                                  xml"/>
<target name="beispiel2pdf" depends="integrate">
 <ant antfile="${dita.dir}${file.separator}build.xml"
                                          target="init">
 <property name="args.input" value="beispiel.
                                              ditamap"/>
 <property name="output.dir" value="out"/>
 <property name="transtype" value="pdf"/>
 </ant>
</target>
```

Listing 3

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE concept PUBLIC "-//OASIS//DTD DITA
                                          Concept//EN"
"../../dtd/concept.dtd">
<concept id="beispiel" xml:lang="de-de">
<title>Installation des Programms</title>
        <prolog></prolog>
        <conbody>
        <p audience="Windows">
    Vor der Installation des Programms müssen Sie den
                                  Rechner anschalten.</p>
        <p>Sie installieren das Programm durch
                              den Aufruf von setup.exe</p>
        </conbody>
</concept>
```

Listing 4

```
The <keyword platform="Linux">chmod</keyword>
                                              command...
<ph product="WhiteknuckleHandsoap">Amalgamated
                              Cleansers get the grime!</ph>
<msgph audience="programmer administrator">
                              Divide by -1 error.</msgph>
<ph otherprops="java">When using Java, use the
 <apiname>com.ibm.obscureclass</apiname>
                              to calculate the value.</ph>
<p importance="recommended" rev="3.2">
                      Update anti-virus software often.</p>
```

Hello Topic World

Die Open-Source-Implementierung setzt die XSLT-Verarbeitung mit Ant und verschiedenen Java-Bibliotheken (jar-Dateien) um. Für den Start mit DITA sollte die Komplettinstallation *fullpackage* von [1] verwendet werden. Dies erspart einigen Installationsfrust, da in dem knapp 18 MB großen Packet alle jar-Bibliotheken in der passenden Version enthalten sind. Die Dateien werden nun in einem eigenen Verzeichnis *C:\ditaot* entpackt. Dazu benennen wir das extrahierte Verzeichnis nach *ditaot* um und verschieben es nach *C:*. Mit installierter Java-Runtime und gesetztem JAVA_HOME ist man nun startbereit.

Enthalten ist eine *startcmd.bat* für Windows bzw. *startcmd.sh* für Unix oder cygwin. Man erhält nach Aufruf der *startcmd* eine Shell, um den Generierungsprozess starten zu können. Um zu prüfen, ob die Installation erfolgreich war, gibt man in dieser Shell *ant -f ant/sample_pdf.xml* ein. Steht am Ende der Bildschirmausgabe *build successful*, so ist alles ok. Gibt es Probleme, so kann dies eventuell durch zusätzliche Installation des XSLT-Prozessors Saxon behoben werden. Dies wird im DITA-OT-Forum beschrieben. Für unser erstes Topic verwenden wir die Datei *beispiel.xml* aus Listing 1. Diese und alle weiteren Dateien editieren wir im Verzeichnis *beispiel*, das wir in *C:\ditaot* erstellen. Zur Steuerung des Generierungsprozesses benötigen wir eine Ant-Steuerdatei (buildfile).

Als Vorlage für *build*-Dateien gibt es im Verzeichnis *ant* die Datei *template_pdf.xml*. Auf die Ant-Syntax wollen wir nicht weiter eingehen, sie ist unter [8] beschrieben. Die Bedeutung der Parameter ist in der Tabelle 1 zusammengefasst. Die wichtigsten Properties beschreiben die Eingabe, die Ausgabe und den Typ der Ausgabe. Die Eingabe referenziert eine *ditamap*-Datei, die wir nun etwas näher betrachten.

Topics zu einem Ganzen bündeln

Bevor wir die Ausgabedateien generieren können, müssen wir noch definieren, welche Topics verarbeitet werden sollen. In der Ditamap-Datei wird nun beschrieben, welche Topics zu einem Informationsgebinde zusammengefügt werden. Schließlich ist die Wiederverwendung der Topics in anderen Zusammenhängen bzw. Dokumenten ein Vorteil, der genutzt werden soll. Die Ditamap-Datei ist also der Bauplan des zu erstellenden Dokumentes. Dieses Konzept der Bündelung erlaubt es, die Topics in verschiedenen Kontexten dadurch wiederzuverwenden, dass sie in einer anderen Ditamap-Datei referenziert werden.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN"
"../dtd/map.dtd">
<!-- (C) Copyright IBM Corporation 2001, 2005. All Rights
                                              Reserved.
-->
<map title="Map fuer die Dita-Benutzung mit ant">
            <topicref href="beispiel.xml"
type="concept"/>
</map>
```

Normalerweise werden mehrere Topics gebündelt. Diese werden unter dem

args.input	Ditamap-Eingabedatei
output.dir	Das Ausgabeverzeichnis relativ zum Basisverzeichnis. Der Dateiname ergibt sich aus dem Namen der Eingabedatei
transtype	Wahl des Ausgabemediums [pdf|xhtml|chm|docbook|...]
dita.input.valfile	Steuerungsdatei für die Zielgruppeninformationen

Tabelle 1: DITA-OT ant Properties

ersten Topic in die Ditamap eingefügt. Durch Verschachtelung der Topics erzeugt man eine Kapitelstruktur, die dann z.B. in der PDF-Ausgabe ein Inhaltsverzeichnis erzeugt. Mit *ant -f beispiel/beispiel-ant.xml* wird die *build*-Datei verarbeitet. Unter *beispiel/out/* befindet sich nun die erzeugte Beispieldatei *beispiel.pdf*.

Zielgruppenmetainformation einmischen

Wir haben nun für die Installationsanleitung einen kleinen Text vorbereitet: „Vor der Installation des Programms müssen Sie den Rechner anschalten." Wir nehmen zwei verschiedene Zielgruppen an: Unerfahrene Windows-Benutzer und erfahrene Unix-Anwender. Der Hinweis ist dann nur für die Windows-User relevant. Der Abschnitt wird mit dem *audience*-Attribut der Zielgruppe zugeordnet. Die vollständige Topicdatei zeigt Listing 3.

Damit wir die Ditamap-Datei unverändert verwenden können, bleibt der Name *beispiel.xml* bestehen. In einer weiteren Datei, der *ditaval*-Datei, wird nun beschrieben, für welches Betriebssystem die Installationsanleitung erstellt wird:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<val>
<prop att="audience" val="Unix" action="include" />
<prop att="audience" val="Windows" action="exclude" />
</val>
```

Den Aufbau dieser Value-Datei findet man leider nicht in der sonst guten englischen Anleitung von DITA-OT. Jetzt müssen wir in der *build*-Datei noch konfigurieren, dass wir eine Value-(val-)Datei verwenden wollen. Dazu wird eine weitere Property-Zeile in die *build*-Datei eingefügt:

```
<property name="dita.input.valfile" value="unix-user.
                                          ditaval"/>
```

Jetzt führen wir erneut *ant -f beispiel/beispiel-ant.xml* aus, um die Ausgabedatei für die Unix-Anwender zu erstellen. Der mit *audience="Windows"* markierte Teil erscheint nicht in der Ausgabedatei, da als Aktion für Windows *exclude* angegeben ist.

Erweiterung

Da es sich bei *audience* um ein *Select*-Attribut (*select-atts*) handelt und dieses in Topic definiert ist, ist dieses Attribut universell anwendbar. Einzelne Paragraphen oder ganze Topics können mit diesem Attribut versehen werden. Aber die Differenzierung geht noch weiter. Es gibt neben der Zielgruppe (*audience*) noch das Produkt (*product*) und sechs weitere Attribute. Die „typischen Beispiele" aus der Anleitung zeigen weitere Möglichkeiten auf (Listing 4).

Fazit

Auch wenn man für produktionsfähige deutsche Ausgaben noch an mehreren Stellen per Hand eingreifen muss, so ist DITA-OT schon sehr stabil und auf jeden Fall eine Alternative, die man betrachten sollte. In die Strukturbeschreibung der einzelnen Topicarten wie *concept* oder *task* ist bereits viel Denkarbeit eingeflossen, das hilft dabei, nichts unberücksichtigt zu lassen. Bei den angesprochenen Alternativen wie *forrest* und *docbook* ist man etwas freier in der Strukturierung seiner Informationen. Daher ist für jedes Projekt eine genauere Betrachtung der Möglichkeiten anzuraten.

***Gernot Glawe**, 1967 in Hildesheim geboren, arbeitet als IT-Consultant bei der CS Consulting AG. Neben seinem Schwerpunktthema Vorgehensmodelle und agile Projekt-Methodologien beschäftigt er sich insbesondere mit der Dokumentation von Softwareprojekten. Kontakt können Sie unter gernot.glawe@cs-consulting.de aufnehmen.*

Links & Literatur

[1] DITA Open Toolkit: dita-ot.wiki.sourceforge.net

[2] Rüping, Andreas: Agile Documentation; John Wiley & Sons Ltd; 2003

[3] forrest.apache.org

[4] www.docbook.org

[5] Noble, James: Notes on Postmodern Programming: www.postmodernprogramming.org

[6] Closs, Sissi: Single Source Publishing; entwickler.press 2007

[7] www.oasis-open.org/specs/index.php#ditav1.0

[8] ant.apache.org