FH Karlsruhe
UNIVERSITY OF APPLIED SCIENCES
HOCHSCHULE FÜR TECHNIK

Fachbereich Informatik

Diplomarbeit

Name:	Alexander Fiedler
Thema:	XML-basierte Modellierung und Übernahme von ERP-Preisinformationen in via MEDICI EPIM unter Berücksichtigung der Marktplatzanbindung mit Hilfe von via MEDICI Mediator
Arbeitsplatz:	viaMEDICI Software GmbH, Ettlingen
Referent:	Prof. Dr. Schaefer-Lorinser
Korreferent:	Prof. Dr. Fuchß
Abgabetermin:	15.02.04

Karlsruhe, den 15.10.03

Der Vorsitzende des
Prüfungsausschusses

Prof. Dr. Gmeiner

Inhaltsverzeichnis

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde zur Erlangung eines akademischen Grades vorgelegt.

Karlsruhe, den 13. Februar 2004

Alexander Fiedler

1 Einführung

1.1 Motivation

Vermehrt wird in den großen Industriezweigen, wie z.B. der Automobil- oder der chemischen Industrie, besonders im B2B-Bereich auf die Prinzipien des eBusiness gesetzt. Insbesondere der Austausch von Preiskatalogen im Rahmen des sogenannten eProcurement hat an Wichtigkeit für die Unternehmen und speziell für die Zulieferer der großen Industriezweige zugenommen.
Die Vorteile des elektronischen Datenaustausches für die einkaufenden Firmen liegen ganz klar auf der Hand. Durch eBusiness und speziell eProcurement ist es Firmen möglich, durch elektronisch vorliegende Produktdaten den Bestellvorgang zu automatisieren und damit implizit die dadurch entstehenden Prozesskosten zu senken. Als Nebeneffekt werden dabei durch den direkten Draht zu den Zulieferern über das Internet die Bestellzeiten stark verkürzt.
Damit dieser Austausch von elektronischen Produktdaten auch funktionieren kann, wurden verschiedene XML-basierte Katalogstandards, wie z.B. BMEcat und OAGIS, definiert. Durch eine Klassifikation z.B. via eCl@ss ist dann sogar ein Vergleich der Produktdaten von miteinander konkurrierenden Anbietern möglich, was für den Einkaufenden wieder neue Möglichkeiten bietet.

Zulieferer der großen Industriezweige können sich aber nicht nur darauf verlassen, dass ausschließlich Großunternehmen zu ihren Kunden gehören. Sie müssen auch sicherstellen, das kleinere Unternehmen ebenfalls Zugang zu Produkt- und Preisinformationen bekommen. Hier bieten sich sogenannte Crossmedia-Systeme an, die Daten medienneutral aus verschiedenen Systemen importieren und für die weitere Verwendung vorhalten.
Diese weitere Verwendung kann sehr vielseitig sein. So können diese Daten sowohl im Publishing-Bereich eingesetzt werden, als auch von einem Web-CMS für die Darstellung im Internet genutzt werden. Es ist je nach System auch möglich, diese vorgehaltenen Daten direkt für die Anforderungen des eBusiness aufzubereiten und z.B. in einer einem Katalogstandard entsprechenden Struktur zu exportieren.

Die Firma viaMEDICI Software GmbH in Ettlingen ist ein Hersteller solch eines Crossmedia-Systems, das Inhalte wie z.B. Produktinformationen meist aus ERP-Systemen importiert und für die weitere Verarbeitung medienneutral vorhält.
Dieses Crossmedia-System nennt sich Enterprise Product Information Manage-

ment oder kurz viaMEDICI EPIM und gliedert sich in mehrere Teile mit unterschiedlichen Funktionalitäten:

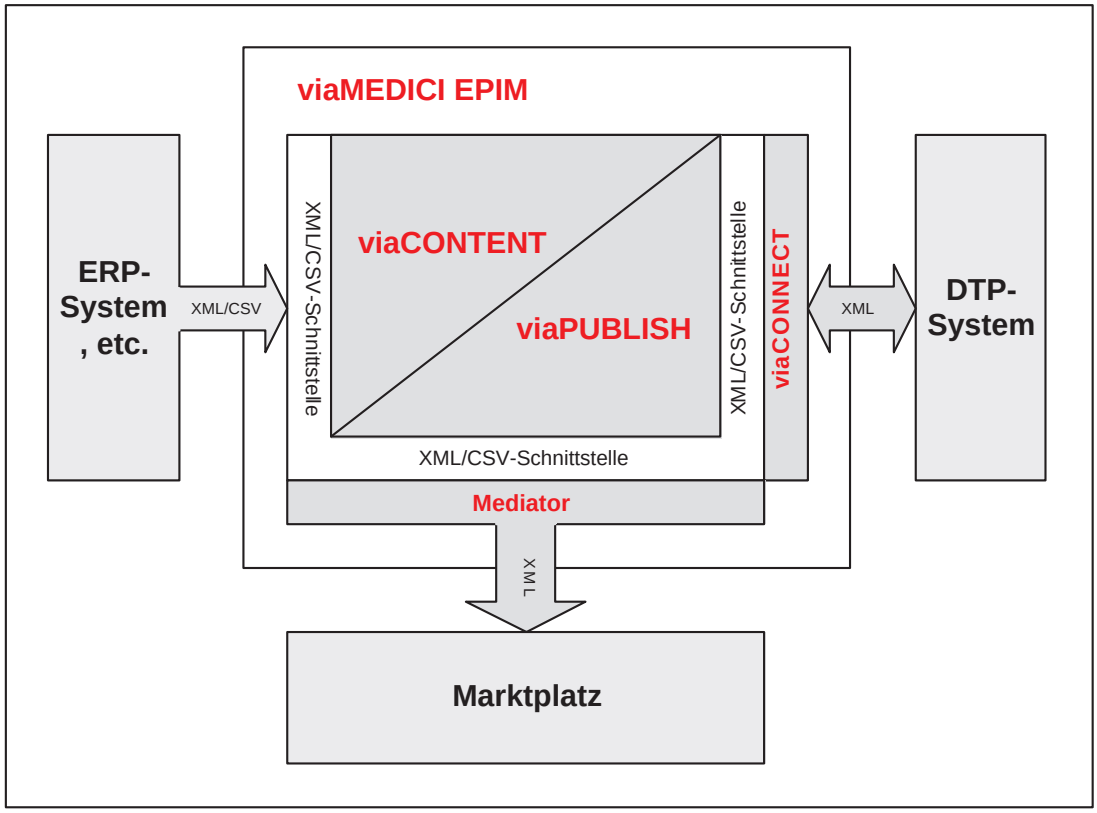


Abbildung 1.1: *Für die Diplomarbeit relevante Struktur von viaMEDICI EPIM*

- **viaCONTENT**
  Hier werden vielfältige Arten von Daten gespeichert. viaCONTENT ermöglicht ebenfalls die Verwaltung und Ergänzung von Daten, durch z.B. Produktmerkmale oder Metadaten.

- **viaPUBLISH**
  Dieser Teil übernimmt das Publikationsmanagement. viaPUBLISH baut dabei auf den XML-Standard als Format für den Datenaustausch und unterstützt so die Erstellung von Print- und Onlinemedien.

- **XML-/CSV-Schnittstelle**
  Damit Daten sowohl in das System importiert als auch aus dem System exportiert werden können, umgibt die beiden Teile viaCONTENT und viaPUBLISH eine Schnittstelle, die Daten im CSV- und XML-Format im- und exportieren kann.

- **viaCONNECT**
  Bei diesem Modul handelt es sich um eine bidirektonale Schnittstelle für Desktop-Publishing-Systeme. In gängigen DTP-Formaten (Quark Express, etc.) vorliegende Daten können insbesondere zur Datenerstübernahme importiert werden. Ebenso besitzt diese Schnittstelle die Fähigkeit, die in medienneutralen XML-Formaten vorliegenden Daten in DTP-Formate zu wandeln und so mit geringem Aufwand druckbare Kataloge zu erzeugen.

- **Mediator**
  Der Mediator übernimmt die Umwandlung der medienneutralen XML-Daten in gängige Katalogstandards (z.B. BMEcat), damit diese Daten auch für elektronische Marktplätze verwendet werden können. Bei der Umwandlung können die Daten auch nach Klassifikationssystemen (z.B. eCl@ss) klassifiziert werden.

Gerade für den Export in eBusiness-Plattformen hat dieses System noch einen Schwachpunkt, der die Motivation für diese Diplomarbeit darstellt: Preisinformationen lassen sich derzeit nur im CSV-Format über den Mediator in Katalogstrukturen einfügen, sie liegen aber nicht in viaMEDICI EPIM selbst vor. Zukünftig sollen Preisinformationen im XML-Format in viaMEDICI EPIM zur Ansicht und zur Weiterverarbeitung vorliegen und so das System vervollständigen.

1.2 Aufgabenbeschreibung

Gegenstand dieser Diplomarbeit ist die Evaluation von Preisfindungsstrukturen in ERP-Systemen und Preisstrukturen in Katalogstandards. Durch die Analyse dieser dem Crossmedia-System vorgelagerten bzw. nachfolgenden Datenstrukturen soll herausgefunden werden, wie die oftmals komplexen Preisinformationen in viaMEDICI EPIM abgebildet werden können.
Aus diesen Erkenntnissen soll dann eine XML-Struktur eintwickelt werden, die eine Weiterverarbeitung der Preisinformationen durch die Software von viaMEDICI ermöglicht. Insbesondere der Mediator von viaMEDICI, der medienneutrale XML-Datenstrukturen auch in den gängigen eBusines-Formaten exportieren kann, soll davon profitieren.
Weiter soll über eine Transformation der im XML-Standard vorliegenden Preisinformationen mittels XSLT eine Möglichkeit für den User geschaffen werden, diese Daten anzusehen. Wichtig ist hierbei, dass an dieser Stelle keine Pflege der Daten möglich sein soll.
Auch soll ein Verfahren entwickelt werden, dass es ermöglicht, die im XML-Format vorliegenden Daten mit dem Mediator in einen Katalogstandard zu transformieren. Als Beispiel könnte hier der in Deutschland weit verbreitete Katalogstandard BMEcat dienen.

2 Darstellung und Verwendung von Preisinformationen

2.1 Betriebswirtschaftliche Grundlagen - Preisdifferenzierung

Oftmals werden für die selben Produkte unterschiedliche Preise angesetzt. Als Beispiel sei an dieser Stelle die Deutsche Bahn AG (www.bahn.de) genannt, die für die Beförderung von einem Ort zu einem anderen je nach Produktvariante und angesprochener Personengruppe unterschiedliche Preise von den Konsumenten verlangt.
Das dieser Erscheinung zugrundeliegende betriebswirtschafliche Prinzip wird **Preisdifferenzierung** genannt. Nach [Skiera1] kann dabei in den Ein-Produkt-Fall und den Mehr-Produkt-Fall unterschieden werden.

2.1.1 Ein-Produkt-Fall

Die Preisdifferenzierung im Ein-Produkt-Fall bedeutet, dass es sich beim angebotenen Produkt um ein einzelnes, unabhängiges Produkt handelt. Bei dieser Variante der Preisdifferenzierung kann nochmals darin unterschieden werden, ob der Anbieter die Konsumenten in verschiedene Gruppen aufteilt und diesen Gruppen unterschiedliche Preise für das selbe Produkt anbietet oder ob sich die Konsumenten anhand verschiedener Produktvariationen selbst den Preis aussuchen können. In [Skiera1] wird der erste Fall **Preisdifferenzierung ohne Selbstselektion** und der zweite Fall **Preisdifferenzierung mit Selbstselektion** genannt.

Preisdifferenzierung ohne Selbstselektion

Diese Art der Preisdifferenzierung unterscheidet nochmals zwischen der individuellen und der gruppenbezogenen Festlegung der Preise.
Bei der **individuellen Festlegung** der Preise wird davon ausgegangen, dass dem Konsumenten ein Preis angeboten wird, der im Idealfall für den Anbieter der maximalen Zahlungsbereitschaft des Konsumenten entspricht. Diese Form der Preisgestaltung wird aber nur sehr selten anzutreffen sein, weil sie vom Anbieter verlangt, dass er die Zahlungsbereitschaft jedes einzelnen Kunden kennt, und weil sie meist aus juristischen Gründen oder Gründen der Fairness so nicht umgesetzt

werden kann.
Bei der **gruppenbezogenen Preisdifferenzierung** kann weiter zwischen **regionenbezogenen und personenbezogenen Preisen** unterschieden werden. Damit diese Art der Preisdifferenzierung Sinn macht, müssen sich die Konsumenten gegenüber dem Anbieter als Zugehörige einer bestimmten Gruppe (z.B. Studenten, Rentner, etc.) oder oder Region (Länder, Kontinente, etc.) identifizieren können.

Preisdifferenzierung mit Selbstselektion

Die Preisdifferenzierung mit Selbstselektion erfordert kein Wissen über die Zahlungsbereitschaft der Konsumenten oder deren Gruppenzugehörigkeit. Durch die Bildung von verschiedenen Variationen eines Produktes liegt es beim Konsumenten selbst, für welche Variation und damit implizit für welchen Preis er sich entscheidet. Bei der Bildung dieser Produktvarianten wird dabei nochmals unterschieden, auf welches Kriterium sich die Preisdifferenzierung genau bezieht.
Bei der **zeitbezogenen Preisdifferenzierung** unterscheidet sich der Preis eines Produktes entweder im Zeitpunkt der Nutzung (z.B. Telefongebühren) oder nach dem Maß der Zeitverzögerung, mit der z.B. Informationen an den Konsumenten weitergegeben werden. Als konkretes Beispiel kann hier die Karlsruher Firma web.de genannt werden, die das Abrufen der Freemail-Accounts im Vergleich zu den Bezahl-Accounts nur alle 15 Minuten erlaubt.
Die **mengenbezogene oder quantitative Preisdifferenzierung** bezieht sich auf die Menge der Produkteinheiten und deren durchschnittlichen Preis pro Mengeneinheit. So wird oftmals beim Kauf von größeren Mengen eines Produktes ein geringerer Preis pro Mengeneinheit bezahlt, als bei kleineren Mengen.
Bei der **leistungsbezogenen oder qualitativen Preisdifferenzierung** werden ähnliche Produktvarianten zu unterschiedlichen Preisen angeboten. Diese Produktvarianten können sich durch die Nutzung des Produktes (z.B. Diesel und Heizöl) oder die an ein Produkt gebundenen Leistungen (z.B. Webspace mit und ohne Werbebanner, Software-Lizenzen für den Einzelplatz oder für mehrere Rechner) unterscheiden.
Die **suchkostenbezogene Preisdifferenzierung** bezieht sich auf die Vertriebskanäle der Produkte oder unter welchem Markennamen bzw. nach welchen Verkaufsstrategien die Produkte angeboten werden. Es wird bei diesem Prinzip davon ausgegangen, dass Konsumenten die höhere Suchkosten investieren, auch eine höhere Zahlungsbereitschaft haben. Bei [Skiera1] wird dabei als Beispiel AOL aufgeführt, die Promotions-CDs mit einer unterschiedlichen Anzahl an Freistunden streuen. Nur durch die Aufwendung hoher Suchkosten, wird es möglich sein, die CDs zu finden, die die maximale Anzahl an Freistunden bieten.
Die vorausgegangenen Betrachtungen bezogen sich lediglich auf eine Dimension der Preisdifferenzierung. In der Praxis sind aber durchaus auch **mehrdimensionale Preisdifferenzierungen** zu beobachten, die die unterschiedlichen Ar-

ten der Preisdifferenzierung kombinieren. Besonders gut zu beobachten ist diese mehrdimensionale Preisdifferenzierung bei Mobilfunk-Anbietern. Zu verschiedenen Zeiträumen und Taktungen kommen meist noch mengenbezogene und personenbezogene Aspekte zur Preisgestaltung hinzu. Laut [Skiera1] soll mit der mehrdimensionalen Preisdifferenzierung eine feinere Segmentierung der Konsumenten angestrebt werden, um so die vorhandene Zahlungsbereitschaft besser auszunutzen.

2.1.2 Mehr-Produkt-Fall

Im Mehr-Produkt-Fall kann zuallererst unterschieden werden, ob die Preise der Produkte voneinander abhängig oder unabhängig sind. Sind die Preise der verschiedenen Produkte unabhängig voneinander, kann man wieder von einem Ein-Produkt-Fall sprechen, weil für jedes Produkt die Prinzipien der Preisdifferenzierung gesondert angewendet werden können.
Sind die Preise der verschiedenen Produkte voneinander abhängig, kann man zwischen einer **isolierten** und einer **kumulierten Kaufbetrachtung** unterscheiden. Im ersten Fall wird auch von Preisbündelung und im zweiten Fall von Prämien und Boni gesprochen.

Preisbündelung

Der Grundsatz der isolierten Kaufbetrachtung oder Preisbündelung liegt in der Zusammenstellung von Produktpaketen, die insgesamt meist günstiger angeboten werden, als die Summe der Preise der enthaltenen Produkte (z.B. Softwarepakete).

Prämien und Boni

Bei der kumulierten Kaufbetrachtung werden auf bestimmte Abnahmemengen verschiedener Produkte oder für langjährige Treue Rabatte und Boni gewährt. Als Beispiele hierzu soll die Vielzahl an Rabattkarten (z.B. Payback) genannt sein, die dem Konsumenten in Supermärkten und anderen Geschäften angeboten werden.

2.2 Allgemeine Anforderungen an Datenstrukturen

Durch die betriebwirtschaftlichen Grundlagen der Preisdifferenzierung ergeben sich nun die Anforderungen an Datenstrukturen zur Darstellung von Preisinformationen.

2.2.1 Strukturierungskonzepte

In [LeukelSchmitz1] wurden insbesondere die bekannten Katalogstandards näher beleuchtet. Dabei wurden drei Strukturierungskonzepte von Datenmodellen für Preisinformationen festgestellt:

1. **Ebenenkonzept**
   In [LeukelSchmitz1] wird angenommen, dass ein Preismodell aus mehreren Komponenten besteht, die zusammen den Produktpreis beschreiben. Sie unterteilen diese Komponenten nochmals in drei geschäftliche Ebenen: *Produkt*, *Transaktion* und *Vertrag*.
   Die in der Ebene *Produkt* enthaltenen Komponenten beziehen sich direkt auf dass zu verkaufende Produkt. Diese Ebene unterteilt sich in weitere Unterebenen, die noch näher beschrieben werden. Der Ebene *Transaktion* ordnen [LeukelSchmitz1] alle Komponenten zu, die wie z.B. Verpackungs- oder Transportkosten durch eine Bestelltransaktion ausgelöst werden. In der dritten Ebene *Vertrag* werden all jene Preiskomponenten gesammelt, die durch Vertragsverhandlungen zweier Geschäftspartner (Anbieter und Kunde) vereinbart wurden. Zu diesen Vertrags-Komponenten zählen z.B. Prämien und Boni. Sie werden oft auch nicht direkt in einem Preismodell angegeben, sondern lediglich durch eine Vertragsnummer oder ähnliches referenziert.

2. **Abhängigkeitskonzept**
   Hier gehen die beiden Autoren davon aus, dass ein Preis von verschiedenen Faktoren (z.B. Kunde, Region, etc.) abhängig ist. Dieses Konzept eignet sich dabei insbesondere zur Erzeugung flacher Datenstrukturen, weil zu jedem Preis nur die Ausprägungen der relevanten Faktoren anzugeben sind.

3. **Zu- und Abschlagskonzept**
   Das letzte Konzept definiert einen Endpreis über verschiedene Zu- und Abschläge, die einen angegebenen Basispreis entsprechend erhöhen oder mindern.

Anhand der Ebene Produkt aus dem Ebenenkonzept wird in [LeukelSchmitz1] nun weiter beschrieben, aus welchen Komponenten sich Produktpreise bilden lassen. Sie unterteilen diese Ebene nochmals in drei Teile: Den eigentlichen Produktpreis, Produktbündel und Zu- und Abschläge mit der Besonderheit Steuern.

2.2.2 Der eigentliche Produktpreis

Bei der Ermittlung des eigentlichen Produktpreises spielen viele Faktoren eine Rolle, die bereits in den betriebswirtschaftlichen Grundlagen der Preisdifferenzierung näher beschrieben wurden. Diese Faktoren können einen Produktpreis einzeln aber auch in unterschiedlichen Kombinationen beeinflussen.
Insgesamt werden in [LeukelSchmitz1] sieben sogenannte Bestimmungsfaktoren aufgezählt, die in Datenstrukturen abgebildet werden sollten:

- **Bestelleinheit**
  Einer der wichtigsten Faktoren für die Bestimmung des Produktpreises sind die beiden Faktoren Bestellmenge und Bestelleinheit. Anhand der Bestelleinheit können auch Produkte unterschieden werden.

- **Gebiet**
  Ein weiterer wichtiger Faktor für die Bestimmung eines Produktptreises ist das Gebiet (Land, Kontinent), in dem das Produkt verkauft werden soll. An diesen Faktor können auch weitere preisbestimmende Faktoren wie z.B. Transportkosten, Steuern oder Zölle gebunden sein.

- **Kunde**
  Gerade im Umfeld von elektronischen Marktplätzen haben Multi-Buyer-Kataloge an Relevanz gewonnen. Diese Multi-Buyer-Kataloge ermöglichen es neben dem Standardpreis von Produkten auch mehrere kundenspezifische Preise zu umfassen. Damit dies möglich ist, muss auch die Angabe eines Kunden in den Datenstrukturen ermöglicht werden.

- **Preistyp**
  Dieser Faktor dient dazu, auf kompakte Art Aussagen über die Preisgestaltung zu machen. Weiter kann zwischen allgemeinen Preistypen, wie z.B. Handelsstufe oder Preis mit Umsatzsteuer, und zeitlich bzw. räumlich begrenzten Preistypen, wie z.B. Schlussverkaufs- oder Aktionspreis, unterschieden werden.

- **Zeitraum**
  Preise sind im allgemeinen nur für einen bestimmten Zeitraum gültig. Dies ermöglicht zu unterschiedlichen Zeitpunkten verschiedene Preise für ein Produkt zu definieren.

- **Vertrag**
  Wie bereits schon erwähnt, hängen Preise insbesondere für Großabnehmer oftmals von zusätzlichen vertraglichen Vereinbarungen ab. In diesen Verträgen werden die genauen Konditionen detailiert aufgeführt. Meistens reicht deshalb in Datenstrukturen für Preismodelle ein Verweis auf den betreffenden Vertrag.

- **Währung**
  Zu jedem Preis zugehörig ist der Faktor Währung. Wird ein Produkt nun in mehreren Währungsräumen angeboten, so müssen Datenstrukturen auch mehrere Währungen aufnehmen können (z.B. in Europa).

Zuätzlich wird in [LeukelSchmitz1] noch zwischen unterschiedlich komplexen Preisangaben selbst unterschieden:

- Preise können durch einen Wert und einen Multiplikator angegeben werden (z.B. 20,00 € für 500 Stück). Meistens wird hier der Multiplikator 1 anzutreffen sein, der dann den Preis für genau ein Stück festschreibt.

- Preise können aber auch linear von Produktmerkmalen abhängig sein, die erst zum Zeitpunkt der Bestellung festgelegt werden. So findet man vorallem im Bereich der Kabel oder Schläuche oft Waren, die nach der Anzahl der benötigten Meter berechnet werden.

- Die komplexeste Art der Preisangaben werden zum Zeitpunkt der Bestellung durch Formeln bestimmt, wie es z.B. bei Wertpapieren oder Metallzuschlägen der Fall ist. In diesem Fall muss dann die Formel in der Datenstruktur erfasst werden.

2.2.3 Produktbündel

Unter Produktbündel versteht man ein Set von einzeln erwerbbaren Produkten (z.B. Büroset, bestehend aus Locher, Kugelschreiber Schreibtischunterlage, etc.), das oftmals zu einem günstigeren Preis angeboten wird, als die Summe der Preise aller einzelnen Bestandteile. Aus Sicht der Datenstruktur von Katalogen werden diese Produktündel in der Regel als eigenständige Produkte angesehen, weil diese Set-Preise auf die unterschiedlichsten Arten ermittelt werden können.

2.2.4 Zu- und Abschläge

Zur Umsetzung der Prinzipien der Preisdifferenzierung werden oftmals Zu- und Abschläge in unterschiedlichen Komplexitätsstufen auf Preise angewandt.
Dazu wird meist ein Basispreis definiert, der als Grundlage für die mindernden Abschläge bzw, erhöhenden Zuschläge dient. Diese Minderungen bzw. Erhöhungen des Basispreises können auf vier unterschiedliche Arten erfolgen:

- *Relativ, Prozent:* Der Zu- bzw. Abschlag wird über einen Prozentsatz angegeben, der zur Errechnung des Endpreises mit dem Basispreis multipliziert wird (z.B. 2% Skonto bei Barzahlung).

- *Relativ, Betrag:* Der Zu- bzw. Abschlag wird über einen festen Betrag definiert, der dann zur Ermittlung des Endpreises vom Basispreis abgezogen

oder zum Basispreis hinzuaddiert wird (z.B. 5,00 € Versicherungspauschale).

- *Absolut:* Der evtl. auf Grund von Regeln ermittelte Endpreis wird durch einen absolut definierten Zu- bzw. Abschlagspreis ersetzt. Dies funktioniert meist mit Hilfe sogenannter Min/Max-Regeln, die bewirken, dass ein niedriger oder höher festgelegter Wert als der erechnete Endbetrag diesen dann ersetzt.

- *Natural:* Die Anzahl der gelieferten Produkte unterscheidet sich von der Anzahl der bestellten und auch bezahlten Produkte (z.B. der Kunde erhält vier Produkte beim Kauf von drei).

Um die Anwendung der Zu- und Abschläge begründen zu können, müssen diese näher charakterisiert werden. Diese Charakterisierung kann auf zwei Arten erfolgen: **volumen- oder leistungsbezogen**. Die **volumenbezogene Art** liegt dann vor, wenn sich ein Preis anhand des Bestellvolumens verändert. Mengenstaffeln können z.B. als volumenbezogene Zu- bzw. Abschläge angesehen werden. **Leistungsbezogene Zu- bzw. Abschläge** entstehen durch zusätzlich erbrachte Leistungen, wie z.B. Skonto bei Barzahlung des Kunden oder Versandpauschalen.
Ein weiterer wichtiger Punkt ist die Abrechnungsart. Zu- und Abschläge können sowohl **produkt- oder transaktionsbezogen** als auch **vereinbarungsbezogenbezogen** abgerechnet werden. Bei der produkt- oder transaktionsbezogenen Abrechnung, werden die Zu- bzw. Abschläge direkt zum Zeitpunkt der Erstellung der Transaktion berechnet. Bei der vereinbarungsbezogenen Abrechnung werden die Zu- bzw. Abschläge meist erst nach Ablauf eines Abrechnungszeitraumes ermittelt und dann zusammen abgerechnet.
Ebenfalls wichtig ist die Reihenfolge der Abrechnung der einzelnen Zu- bzw. Abschläge. Dies ist insbesondere bei mehrstufigen Rabattmodellen unumgänglich, weil Zu- und Abschläge in verschiedenen Reihenfolgen angewendet, auch unterschiedliche Endpreise bewirken können.
Eine besondere Art der Zu- und Abschläge stellen **Steuern** dar. Sie werden in Datenstrukturen für Preisinformationen oftmals aus rechtlichen Gründen und aus Gründen der regionalen Unterschiede gesondert aufgeführt. In Deutschland z.B. wird normalerweise die enthaltene Umsatzsteuer mit ausgewiesen.

2.3 Preisinformationen in ERP-Systemen

Zuerst soll ein Blick auf die Struktur von Preisinformationen in ERP-Systemen geworfen werden, weil diese sozusagen die Quelle der in XML zu strukturierenden Preisinformationen sind.
Dabei war es nicht wirklich einfach, Informationen über die Entstehung und Speicherung von Preisinformationen in ERP-Systemen zu bekommen. Nach einer zeitaufwändigen Suche konnten jedoch im Internet und über Beziehungen Handbücher zum Thema Preisfindung in den gängigen ERP-Systemen von SAP [SAP1] und JDEdwards (zukünftig Peoplesoft) [JDE1] ausfindig gemacht werden, die beschreiben, wie Preise anhand verschiedener Voraussetzungen wie z.B. Kunde, Zeitraum, Land, etc. gebildet werden.

2.3.1 SAP

Die SAP AG mit Hauptsitz in Walldorf/Deutschland ist der Marktführer bei ERP-Systemen und darüber hinaus. Die weltweite Präsenz der Systeme von SAP und insbesondere die starke Verbreitung der SAP-Systeme in Deutschland ist die Motivation die Preisfindung in diesen Systemen im Rahmen dieser Arbeit genauer anzuschauen.
Eine große Hilfe war dabei die frei verfügbare Online-Hilfe [SAP1], die einen Einblick in die Funktionsweisen der einzelnen Systemmodule ermöglicht.

Funktionsweise

Im ERP-System von SAP ist das Modul SD (Sales and Distribution) ([SAP2]) für die Bildung von Preisen zuständig. Die Preise werden in diesem Modul mit Hilfe der sogenannten Konditionstechnik ermittelt. Dabei nehmen die eingegebenen Auftragsdaten direkt Einfluss auf die Preisfindung.
Über das im Kundenstammsatz hinterlegte Kundenschema wird das zugehörige Kalkulationsschema ermittelt, dass den weiteren Preisfindungsprozess steuert. Im Kalkulationsschema wird festgelegt,

- welche Konditionsarten zulässig sind und in welcher Reihenfolge sie abgearbeitet werden.
- welche Bedingungen erfüllt sein müssen, damit eine bestimmte Konditionsart berücksichtigt wird.
- welche Zwischensummen gebildet und angezeigt werden.
- inwieweit die Preisfindung durch manuelle Bearbeitung beeinflusst werden darf.
- auf welcher Basis prozentuale Zu- und Abschläge berechnet werden.

Es können in SAP sowohl vordefinierte Kalkulationsschemata ausgewählt als auch eigene Schemata angelegt werden.
In den im Kalkulationsschema gefundenen und auch wirklich zu berücksichtigenden Konditionsarten wird nun in der festgelegten Reihenfolge nach zutreffenden Konditionssätzen gesucht. Dabei bildet jede Konditionsart einen bestimmten Aspekt der Preisfindung ab. In den Konditionssätzen werden dabei die Rechenregel (z.B. prozentual) und die zugehörige Bezugsgröße hinterlegt.

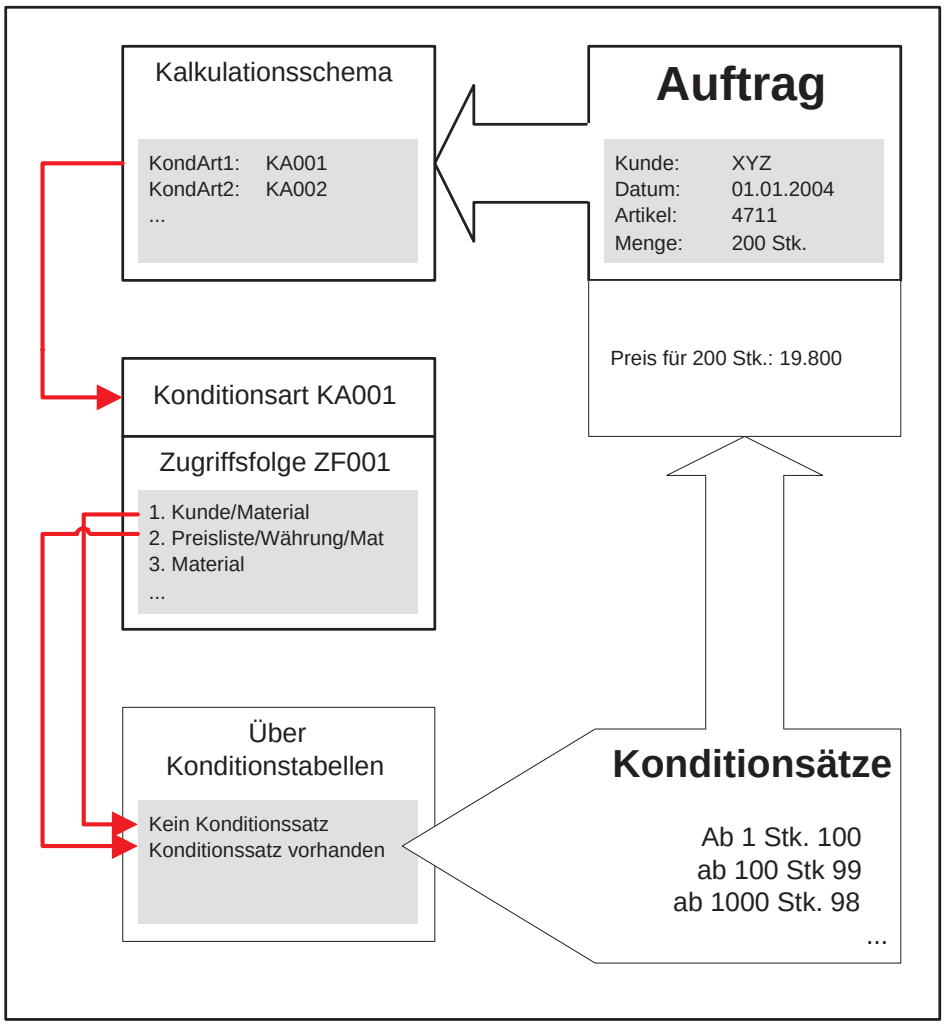


Abbildung 2.1: *Ablauf des Preisfindungsprozesses in SAP*

Jeder Konditionsart kann weiter eine Zugriffsfolge zugeordnet sein. Eine Zugriffsfolge ist dabei eine Suchstrategie, mit deren Hilfe gültige Daten für eine Konditionstabelle gesucht werden können. Dabei kann eine Zugriffsfolge aus einem oder mehreren Zugriffen bestehen. Durch die Vergabe einer Priorität wird festgelegt, in welcher Reihenfolge Zugriffe auf Konditionssätze erfolgen sollen. Dabei

ist bei der Einrichtung dieser zu beachten, dass die Suche von spezifischen Konditionssätzen in Richtung allgemeingültiger Konditionssätze erfolgen sollte. Zum Beispiel sollte erst ein kundenspezifischer Konditionssatz gefunden werden, bevor ein Konditionssatz für alle Großhändler gefunden wird.
Die Suche nach Konditionssätzen erfolgt dabei nicht direkt. Konditionssätze werden vielmehr über Konditionstabellen gesucht, die eine Kombination der Felder (Schlüssel) darstellt, die einen solchen Konditionssatz kennzeichnen. Wird nun ein passender Konditionssatz gefunden, wird aus den darin enthaltenen Informationen ein Preis berechnet.
Dieser Vorgang wird nun für jede im zugehörigen Kalkulationsschema vorgefundene Konditionsart wiederholt. Durch Verrechnung der einzelnen Preise wird am Ende des Vorganges ein Gesamtpreis ermittelt.

Datenstrukturen

Trotz intensiver Suche im Internet ist es nicht gelungen, einen Einblick in das Datenmodell der Konditionstechnik im SAP-System zu bekommen. Über die Exportmöglichkeit über die iDoc-Schnittstelle (siehe Seite 21) lässt sich aber zumindest auszugsweise aufzeigen, wie die Daten innerhalb des Systemes strukturiert sein müssen.
Für alle Konditionen werden für jeden Artikel eine Vielzahl an Informationen im SAP-System hinterlegt. Diese Informationen lassen sich mit Hilfe der umfangreichen Datenstruktur E1KOMG im iDoc COND_A1 exportieren, die aus Gründen der Übersichtlichkeit und des Textflusses im Anhang auf Seite 108 detailliert dargestellt und erläutert wird. In welchen Strukturen diese und auch die im folgenden aufgezeigten Daten im System selbst gespeichert sind, konnte mit Hilfe von frei zugänglichen Quellen nicht ermittelt werden. Zu diesen Daten zählen jedenfalls Informationen zu Kunde, Artikel, Steuerberechnung, Vertriebsinformationen und vieles mehr. Einige der dort abgebildeten Daten werden auch für die Entwicklung der XML-Datenstruktur von Wichtigkeit sein.
Für jeden im System zu bepreisenden Artikel gibt es zusätzlich zu den Filterinfornmationen einen oder mehrere Konditionsköpfe, die einige allgemeine Informationen zu den Artikelkonditionen bereithalten. Welche das im Detail sind zeigt die folgende Tabelle:

Feld	**Bemerkung**
KNUMH	Nummer des Konditionssatzes
DATAB	Gültigkeitsdatum von
DATBI	Gültigkeitsdatum bis
KOSRT	Suchausdruck für Konditionen
KZUST	Verantwortlichkeit in SD für Kondition bzw. Material
KNUMA_PI	Nummer für die Promotion/Aktion
KNUMA_AG	Nummer für die Verkaufsaktion

Feld	Bemerkung
KNUMA_SQ	Nummer für die Kundenabsprache
KNUMA_SD	Nummer für die Standard-Vereinbarungen
AKTNR	eindeutige Nummer für die Aktion, für die der aktuelle Beschaffungsvorgang erfolgt
KNUMA_BO	eindeutige Nummer für die Rabatt-Vereinbarungen

Tabelle 2.1: SAP: Datenstruktur des Konditionskopfes E1KONH

Jedem Konditionskopf müssen nun eine oder mehrere Konditionspositionen zugeordnet sein, die den einzelnen Zu- bzw. Abschlägen entsprechen, die auf den Produktpreis wirken. Die detaillierte Struktur der einzelnen Konditionspositionen zeigt die folgende Tabelle:

Feld	Bemerkung
KSCHL	Konditionstyp
KNUMT	Anzahl der Texte
STFKZ	Staffeltyp
KZBZG	Indikator der Staffelbasis, legt fest, wie eine Staffel in einer Kondition interpretiert wird.
KSTBM	Menge der Konditionsstaffel
KONMS	Maßeinheitauf, die sich die Mengenstaffel bezieht
KSTBW	Staffelwert
KONWS	Staffelwährung
KRECH	Rechenart der Kondition
KBETR	Konditionsrate (als Betrag oder Prozentsatz)
KONWA	Einheit der Konditionsrate (Währung oder Prozent)
KPEIN	Preiseinheit der Kondition von einem Nicht-SAP-System
KMEIN	Mengeneinheit, die sich auf die Konditionsrate bezieht
PRSCH	Preislevels
KUMZA	Zähler für die Umrechnung der Mengeneinheit in die zugehörige Basiseinheit
KUMNE	Nenner für die Umrechnung der Mengeneinheit in die zugehörige Basiseinheit
MEINS	Basiseinheit
MXWRT	untere Grenze der Konditionsrate
GKWRT	obere Grenze der Konditionsrate
PKWRT	geplanter Konditionswert
KWAEH	Konditionswährung
UKBAS	geplante Konditionsbasis
KZNEP	Ausschlusskennzeichen
KUNNR	Kundennummer 1

Feld	Bemerkung
LIFNR	Nummer des Lieferanten
MWSK1	Umsatzsteuer
LOEVM_KO	Löschkennzeichen
ZAEHK_IND	Index der Konditionsposition
BOMAT	Artikel für Rabattabrechnung
KBRUE	Abgrenzungssumme
KSPAE	zeigt an, wenn der Rabat nachträglich gewährt wurde
BOSTA	Status der Rabatt-Vereinbarungen
KNUMA_PI	Nummer der Aktion/Promotion
KNUMA_AG	Nummer der Verkaufsaktion
KNUMA_SQ	Nummer der Kundenabsprache
VALTG	legt Tage zwischen Rechnungsdatum und Gültigkeit der Zahlungsbedingungen fest
VALDT	Datum an dem die Zahlungsbedingungen im Bezug auf den Verkaufsbeleg gültig werden
ZTERM	Schlüssel der Zahlungsbedingungen
ANZAUF	Maximale Anzahl an Kundenaufträge pro Konditionssatz
MIKBAS	Minimaler Wert der Konditionsbasis
MXKBAS	Maximaler Wert der Konditionsbasis
KLF_STG	Anzahl der Staffeln
KLF_KAL	Anzahl der Preisstaffeln
VKKAL	Verkaufspreisberechnung
AKTNR	Aktion/Promotion
KNUMA_BO	Vereinbarung (nachträgliche Abgrenzung)
BOMAT_GUID	externe GUID für Feld BOMAT

Tabelle 2.2: SAP: Datenstruktur der Konditionspositionen E1KONP

Jeder Konditionsposition können noch eine oder mehrere Mengen- oder Wertstaffeln zugeordnet sein. Die folgenden beiden Tabellen beschreiben die detaillierte Datenstruktur dieser Staffeln:

Feld	Bemerkung
KSTBM	Menge, auf die sich die Staffel bezieht
KBETR	Konditionsrate, kann ein Prozentsatz oder ein konkreter Wert sein

Tabelle 2.3: SAP: Datenstruktur der Mengenstaffel

Feld	Bemerkung
KSTBW	Menge, auf die sich die Staffel bezieht
KBETR	Konditionsrate, kann ein Prozentsatz oder ein konkreter Wert sein

Tabelle 2.4: SAP: Datenstruktur der Wertestaffel

Der Unterschied von Mengenstaffel und Wertstaffel liegt dabei nicht in der Datenstruktur selbst, sondern in den hinterlegten Werten. Bei der Mengenstaffel ist die bestellte Menge die Bezugsgröße für die hinterlegte Konditionsrate, bei der Wertestaffel bezieht sich die hinterlegte Rate auf den Geldwert der bestellten Ware.

Welche Teile dieser recht umfangreichen Datenstruktur letztendlich in die zu erstellende XML-Struktur einfließen werden, wird die Analyse der Anforderungen (siehe Seite 45) im Detail zeigen.

Export der Preisinformationen

Direkt in der Online-Hilfe zum Thema „Preisfindung und Konditionen“ ([SAP2]) wird der Export über die SAP-Schnittstelle iDocs beschrieben. Über diese iDocs-Schnittstelle lassen sich Daten insbesondere zwischen zwei SAP-Systemen sowohl im- als auch exportieren.
Die iDoc-Datenschnittstelle für den Austausch von Katalogdaten inklusive der Konditionen heißt bei SAP COND_A01 und wird im Interface Repository ([SAP4]) detailliert beschrieben.
Der grundsätzliche Aufbau der die Konditionstechnik betreffenden Datenelemente ist wie folgt:

- **für Filterfunktionen E1KOMG**
  Dieses Element muss genau einmal vorhanden sein und enthält Informationen für Filterfunktionen (Struktur auf Seite 108).
  - **Konditionskopf E1KONH**
    Für jeden Artikel gibt es von einem bis zu 9999 Konditionsköpfe mit allgemeinen Informationen zum Konditionssatz (Struktur auf Seite 19).
    * **Konditionspositionen E1KONP**
      Jedem Konditionskopf können mehrere Konditionspositionen zugeordnet sein (Struktur auf Seite 20).
      Dieses Element muss mindestens einmal und kann bis zu 99 mal vorhanden sein.

- **Mengenstaffel E1KONM**
  Dieses Element kann bis zu 9999 mal vorkommen (Struktur auf Seite 20).
- **Wertestaffel E1KONW**
  Dieses Element kann bis zu 9999 mal vorkommen (Struktur auf Seite 21).

2.3.2 JDEdwards

JDEdwards bzw. Peoplesoft bieten nur spärliche Informationen über ihre ERP-Systeme respektive der Preisfindung in diesen Systemen, die frei zugänglich sind. Über geschäftliche Beziehungen ist es jedoch gelungen, das Handbuch zur Preisfindung einer älteren Version (A7.3 vom Juni 1996) [JDE1] zu bekommen, dass nochmals eine etwas andere Art der Preisfindung aufzeigt.

In JDEdwards wird zwischen der Basisbepreisung und der erweiterten Bepreisung unterschieden. Die Basisbepreisung selbst ist Bestandteil des Systemes „Vertriebssystem“, während die erweiterte Bepreisung in der beschriebenen Version ein eigenständiges Softwaresystem ist, das in das Standardsystem integriert werden kann.

Basisbepreisung

Funktionsweise Die Grundlage der Basisbepreisung in JDEdwards bildet die Basispreishierarchie. In ihr wird festgelegt, in welcher Reihenfolge die Basispreis-Datensätze durchsucht werden sollen. In der Basispreishierarchie ist es sinnvoll die Suchreihenfolge so festzulegen, dass zuerst nach spezifischen Basispreisen gesucht wird und die zu suchenden Preise dann immer allgemeiner werden. Das heißt nun konkret, dass die Basispreishierarchie so festgelegt werden sollte, dass z.B. zuerst nach kundenspezifischen Preisen gesucht werden soll, bevor nach Preisen für eine Kundengruppe gesucht wird.
Zur Vereinfachung der Pflege der Basispreise können Artikel bzw. Kunden mit ähnlichen Eigenschaften zu Artikel- bzw. Kundengruppen zusammengefasst werden. Artikel können so z.B. nach Farben und Kunden z.B. nach Regionen zu Gruppen zusammengefasst werden, für die jeweils ein Basispreis hinterlegt werden kann. Ergänzend zu diesen Gruppen lassen sich Basispreise für jedes Werk bzw. jede Niederlassung bis hin zum Lagerort festlegen. Die Realisierung dieser Bepreisungsmöglichkeit wird über Bepreisungsebenen im System festgelegt.
Die Bepreisung kann auch über einen im System festzulegenden Gültigkeitszeitraum variieren. Eine letzte Möglichkeit, unterschiedliche Preise festzulegen, ist der Währungscode. Dieser Währungscode ist Bestandteil des Schlüssels jedes Basispreises und ermöglicht so die Definition von Preisen für verschiedene Länder und Währungen.

Datenstrukturen Interessant für die Darstellung von Preisinformationen in einem XML-Format ist, wie die Datenstruktur der Basispreise in JDEdwards aufgebaut ist. Dem Handbuch [JDE1] kann man entnehmen, das zu jedem Set von Basispreisen Kopfdaten gehören. Diese Kopfdaten enthalten je nach eingerichteter Hierarchie Daten zu Artikel, Kunde oder den jeweils vorhandenen Gruppen. Zu diesen Daten gehören z.B. die Artikel- und Kundennummer, eine Bezeichnung für die Niederlassung bzw. das Werk, dessen Standort und das Los bzw. die Seriennummer eines Artikels.
Die Datenstruktur der Basispreise selbst baut sich wie folgt auf:

- **Los/Lagerort**
- **Währungscode**
- **Mengeneinheit**
- **Preis**
- **Gültigkeitsdauer**

Wie die Datenstruktur im Detail aussieht, lies sich aus den vorhandenen Quellen leider nicht rekonstruieren. Die vorhandenen Informationen sind aber als Anhaltspunkt für die Modellierung von Preisinformationen sicherlich hilfreich.

Export der Preisinformationen Über Exportmechanismen und -datenstrukturen für Daten der Basisbepreisung des ERP-Systemes von JDEdwards konnten leider weder über freie Quellen noch durch Beziehungen Informationen gefunden werden.

Erweiterte Bepreisung

Funktionsweise Im Gegensatz zu den nicht so umfangreichen Konfigurationsmöglichkeiten von Preisen in der Basisbepreisung von JDEdwards bietet die erweiterte Bepreisung eine Vielzahl an Möglichkeiten die Preisgestaltung zu beeinflussen.
Die Datensätze, die die Preisfindung beeinflussen, heißen bei JDEdwards Preisanpassungen. Preisüberschreibungen, Aufschläge oder Rabatte können mit diesen Preisanpassungen über absolute oder prozentuale Angaben, variable Preistabellen oder Formeln definiert werden. Auf jede Auftragsposition können mehrere Preisanpassungen angewandt werden. Die Zuordnung der einzelnen Preisanpassungen zu einem Kunden und die Reihenfolge der Anwendung erfolgt über Anpassungspläne.
Zur Vereinfachung der Preisanpassung insgesamt können auch bei der erweiterten Preisanpassung mehrstufige Artikel- oder Kundengruppen definiert werden. Das heißt, dass Preisanpassungen sowohl für einzelne Kunden oder Artikel als auch

für ganze Gruppen von Artikeln bzw. Kunden definiert werden können. Bei den Kunden kann hier sogar nochmals zwischen Lieferadresse, Kaufadresse und einer übergeordneten Adresse unterschieden werden.
Insgesamt gibt es in der erweiterten Bepreisung im ERP-System von JDEdwards mehrere Preisanpassungsarten. So können Preisanpassungen bestimmte Beträge sein, die den ursprünglichen Preis überschreiben können. Auch ist es möglich, Aufschläge bzw. Rabatte über einen Prozentwert des gegenwärtigen Preises festzulegen. Die komplexeste Art der Preisanpassung lässt sich bei JDEdwards aber über mehr oder minder komplexe Formeln bzw. Bezüge auf Preistabellen (z.B. bei schwankenden Materialpreisen) definieren.
Mittels der erweiterten Bepreisung lassen sich auch bei JDEdwards Gratisartikel definieren, die Kunden z.B. zur effektiveren Bewerbung eines Artikels zu ihrer Lieferung dazu erhalten sollen. Mit Gratisartikeln lassen sich auch Naturalrabatte definieren. Auch lassen sich im Erweiterungsmodul sogenannte Auftragsschwellen festlegen. Diese Auftragsschwellen beeinflussen den Preis von Artikeln entweder direkt bei der Bestellung oder kumuliert in einem bestimmten Zeitraum. In anderen Systemen ist diese Methode auch als Mengenstaffel bekannt.
Neben den genannten Möglichkeiten, bietet auch die erweiterte Bepreisung die Möglichkeit, über einen Gültigkeitszeitraum und verschiedene Währungen auf die Preisbildung Einfluss zu nehmen. Auch lässt sich im System festlegen, welche Details beim Rechnungsdruck beachtet werden sollen und welche nicht.
Alle Preisanpassungen werden in einer Bepreisungshistorie hinterlegt. Das System sorgt so dafür, dass die Entstehung von Preisen auch nach der Berechnung noch nachvollziehbar bleibt.
Das Flussdiagramm (siehe Seite 25) zeigt den Ablauf der Preisfindung in einem ERP-System von JDEdwards mit der erweiterten Bepreisung.

Datenstrukturen Das Handbuch [JDE1] legt anhand der einzugebenden Daten einen Teil der notwendigen Datenstrukturen für die erweiterte Bepreisung offen. Diese Datenstrukturen sollen nun im Hinblick auf die zu entwerfende XML-Datenstruktur näher betrachtet werden.
Bevor die eigentlichen Preisanpassungen eingegeben werden können, müssen zuerst die Basispreishierarchie und die Anpassungsdefinitionen erstellt werden.
Über die Basispreishierarchie lässt sich festlegen, mit welcher Priorität über bestimmte Kriterien nach Preisanpassungen gesucht wird. Dabei werden die Kriterien als Matrix von verschiedenen Adressen (Lieferadresse bis alle Adressen) und Artikeln (Artikelnummer, Artikelgruppe und alle Artikel) dargestellt, in die die jeweilige Hierarchieebene von 1 bis maximal 21 eingetragen werden kann. Die Suche nach Anpassungen lässt sich so von spezifischen Kriterien, wie z.B. der Adresse des Kunden und einer Artikelnummer, bis hin zu allgemeinen Kriterien, wie z.B. alle Kunden und alle Artikel, sortieren.
Die Anpassungsdefinitionen dienen der Spezifikation der Eigenschaften einer An-

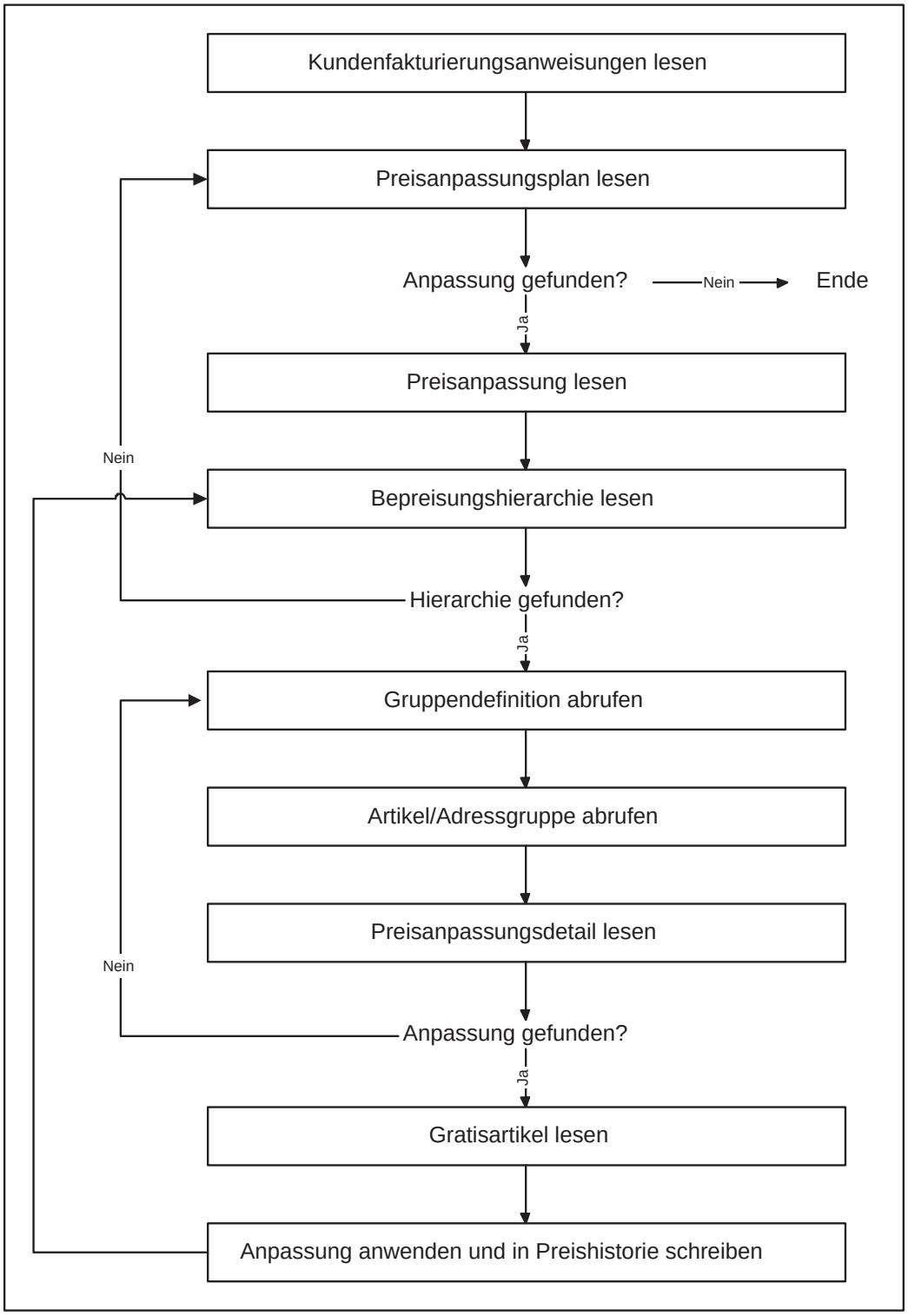


Abbildung 2.2: *Flussdiagramm der erweiterten Bepreisung in JDEdwards (aus [JDE1])*

passung. In diesen Definitionen können der Name der Anpassung, die Artikel- und Kundengruppe, die Preisschwellenart (Menge, Gewicht oder Betrag), ob der Basispreis überschrieben werden soll, ob manuelle Änderungen möglich sind und vieles mehr festgelegt werden.
Wie bereits beschrieben müssen nun Anpassungspläne eingerichtet werden. Diese Anpassungspläne bestehen aus einem Namen und den einzelnen Anpassungen, deren Anwendungsreihenfolge über eine Sequenznummer festgelegt wird.
Die eigentlichen Preisanpassungen werden in den Anpassungsdetails festgelegt. Festlegen lassen sich folgende Eigenschaften:

- **Währungscode**
- **Menge - Von**
  Hier kann die minimale Menge angegeben werden, ab der eine bestimmte Anpassung in einer Mengenstaffel gültig wird.
- **Mengeneinheit**
- **Faktor**
  Dieser Faktor gibt in Zusammenhang mit dem Basiscode an, wie die Auftragsposition angepasst werden soll. Ist der Basiscode 1 bis 5 wird die Veränderung der Auftragsposition als positive oder negative Zahl angegeben. Ist der Basiscode 6, wird der neue Preis über eine variable Datei ermittelt. Bei 7 wird eine Formel für die Berechnung des Preises verwendet. Mit Basiscode 8 wird die Anpassung mittels eines anwenderspezifischen Programmes berechnet.
- **Basiscode**
  Über den Basiscode, der eine Zahl von 1 bis 8 sein kann, wird festgelegt, wie sich der eingegebene Faktor auf die Preisanpassung auswirkt. Hier können von verschiedenen Rechenarten wie Multiplikation mit verschiedenen Basiswerten oder eine Addition zu den Artikelkosten bis zu Aufrufen von Formeln und Dateien definiert werden.
- **Gültigkeitsbereich**

Jeder beliebigen Anpassung können weiter Gratisartikel zugeordnet werden. Diese Gratisartikel werden mit einem Gültigkeitszeitraum, einer Mindestmengenangabe und einem Währungscode angelegt. Die einzelnen Artikel haben dann noch folgende kennzeichnende Datenfelder:

- **Artikel**
  Eindeutige Artikelnummer, die den Gratisartikel kennzeichnet.
- **Menge**
  Menge der abzugebenden Gratisartikel.

- **Preis**
  Hier kann ein Preis in der Maßeinheit des Ausgangsartikels angegeben werden, der aber in der Regel den Wert 0 hat.

- **Verarbeitungsart**
  Dieser Code gibt an, wie das System den Gratisartikel verarbeiten soll (gesonderter Posten, Netto-Reduktionspreis oder Netto-Reduktionsmenge).

- **Menge überbestellt**
  Der Wert in diesem Feld gibt an, ab welcher Menge über der in der Anpassung definierten Mindestmenge ein Gratisartikel der Bestellung zugefügt werden soll. Ein Gratisartikel wird auch automatisch bei jedem Vielfachen des Wertes zugefügt.

Export der Preisinformationen Über Exportmechanismen und -datenstrukturen für Daten der erweiterten Bepreisung des ERP-Systemes von JDEdwards konnten leider weder über freie Quellen noch durch Beziehungen Informationen gefunden werden.

2.3.3 Fazit

Abschließend kann zur Preisfindung in ERP-Systemen gesagt werden, dass sowohl die Software von SAP als auch die Software von JDEdwards im Hinblick auf die allgemeinen Anforderungen an Datenstrukturen dem Konzept der Zu- und Abschläge (siehe Seite 14) folgen. In beiden Systemen werden Preise adhoc durch festgelegte Zu- und Abschläge ermittelt. Ausgangspunkt für die Berechnungen sind dabei die definierten Basispreise und nähere Informationen zur aktuellen Bestellung (Kunde, Datum, Menge, etc.).
Die bei der Evaluation der ERP-Systeme gewonnenen Erkenntnisse werden sich in einer statischen XML-Datenstruktur nur schwer darstellen lassen. Vorstellbar ist aber, dass die in den ERP-Systemen definierten Zu- und Abschläge in der XML-Struktur der zusätzlichen Preistransparenz dienen können. Das heißt in diesem Fall, dass die Berechnung der Preise nachvollziehbar und in anderen Softwaresystemen auch reproduzierbar wird.

2.4 Darstellung von Preisinformationen in Katalogstandards

Interessanter als die Datenquellen der Preisinformationen sind die möglichen Zielformate. Diese Zielformate nennen sich Katalogstandards oder in der erweiterten Form Standards für den Austausch von Geschäftsdaten und ermöglichen den Austausch von Produkt- und Preisdaten zwischen verschiedenen und unterschiedlich strukturierten Unternehmen.
Diese Zielformate sind für diese Arbeit interessanter, weil die Preisinformationen aus den ERP-Systemen letztlich in der zu erstellenden Datenstruktur nur zur Ansicht zwischengespeichert werden und das Ziel ein Export dieser Daten in Richtung elektronischer Marktplätze ist. Dieser Export soll mittels einem der gängigen Katalogstandards realisiert werden. Da sich der angesprochene Kundenkreis im Moment auf Deutschland beschränkt, wird im folgenden der deutsche Katalogstandard BMEcat [BMEcat1] näher betrachtet. In der anschließenden Zusammenfassung werden auch einige der anderen Katalogstandards kurz beleuchtet und deren Besonderheiten dargestellt.

2.4.1 BMEcat

Historie

Initiator der Entwicklung eines gemeinsamen Katalogstandards war der Bundesverband Materialwirtschaft, Einkauf und Logistik e. V. (BME) mit Sitz in Frankfurt a.M.. Den Bestrebungen des BME, einen einheitlichen Katalogstandard zu entwickeln, schlossen sich bald namhafte Firmen wie DaimlerChrysler, BMW, Siemens, VISA und viele mehr an, weil die Vielzahl der existierenden Katalogstandards auch den elektronischen Handel dieser großen Firmen nicht einfacher machte. Der neue Katalogstandard hatte also das Ziel, den Austausch von Produktkatalogen zwischen Lieferanten und einkaufenden Organisationen zu standardisieren und damit implizit zu vereinfachen. Im November 1999 konnte als Ergebnis der Bemühungen die Version 1.0 des neugeschaffenen Katalogstandards BMEcat vorgestellt werden. Die Mitarbeit vieler großer Firmen sicherte dem neuen Standard eine rasche Verbreitung in Deutschland.
Am 02.11.2000 wurde dann nach Beseitigung einiger Inkonsistenzen und einigen Überarbeitungen die Version 1.01 veröffentlicht. Die aktuelle Version 1.2, die nun auch näher betrachtet werden soll, wurde am 27.03.2001 veröffentlicht.
Seit dem Jahr 2002 wird nun an der Version 2.0 des Standards gearbeitet, dessen Veröffentlichung aktuell für Ende des 1. Quartals 2004 angekündigt ist. Diese Version wird vorallem geprägt sein durch Erweiterungen des Produkt- und Preismodells und durch eine Vielzahl an Verbesserungen:

- Multi-Kunden-Kataloge, d.h. Informationen für mehrere Kunden in einem

Katalogdokument

- Preise auf Grundlage verschiedener Rahmenverträge in einem Katalogdokument
- Preisformeln, die eine Berechnung des Preises nach der Erstellung des Kataloges mittels Benutzereingaben oder Informationen aus dem Internet ermöglicht
- Unterstützung preisrelevanter Konfigurationen

Grundsätzlicher Aufbau

Die eigentlichen Informationen werden bei BMEcat in sogenannten Katalogdokumenten abgelegt. Das Wurzelelement jedes Katalogdokumentes ist das Element BMECAT und besteht aus einem Kopfteil (HEADER) und einem Transaktionsteil.

Der HEADER steht grundsätzlich am Anfang des Katalogdokumentes und enthält gobale Informationen, die für alle Variationen des Datenaustausches gültig sind. Um diese Informationen aufzunehmen, gliedert sich der Kopfteil nochmals in Unterelemente, wie das folgende Bild zeigt.

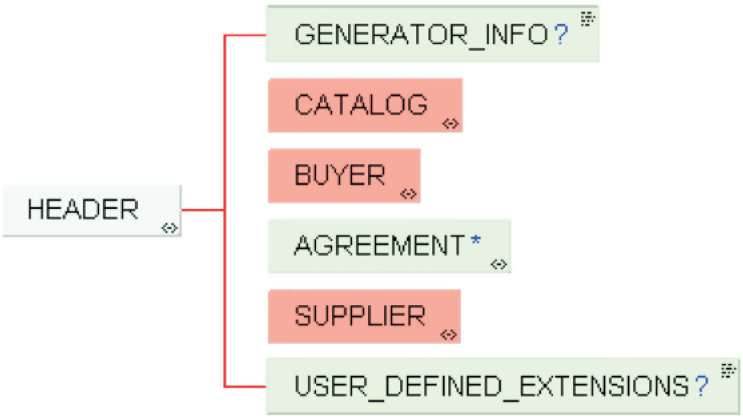


Abbildung 2.3: Struktur des BMEcat-Kopfteils

Im Unterelement GENERATOR_INFO kann das Werkzeug, dass das Katalogdokument erzeugt, Informationen zu sich selbst und der Erstellung ablegen. Das Element CATALOG speichert Informationen zur Identifikation und Beschreibung des Produktkataloges. Es enthält weiter eine Reihe von Defaultwerten, auf die zurückgegriffen wird, wenn in den spezifischeren Elementen keine Informationen

hinterlegt sind. Informationen zum katalogempfangenden Kunden werden im Element BUYER gespeichert. Im Element AGREEMENT werden Informationen zu den dem Katalogdokument zugrundeliegenden Rahmenverträgen hinterlegt. Im Element SUPPLIER werden Informationen zum katalogerzeugenden Unternehmen gespeichert und mit dem Element USER_DEFINED_EXTENSIONS wird ein Bereich festgelegt, in dem eigene Elemente überliefert werden können.

Der Transaktionsteil legt fest, welche Teile des Produktkataloges ausgetauscht werden. Festgelegt wird dies durch die drei möglichen Transaktionen, die in der folgenden Tabelle aufgezählt werden.

Transaktionselement	**Anwendung**
T_NEW_CATALOG	beim Anlegen eines neuen Kataloges
T_UPDATE_PRODUCTS	beim Verändern von Artikeldaten
T_UPDATE_PRICES	beim Verändern von Preisdaten

Tabelle 2.5: Transaktionsarten in BMEcat

Stellvertretend für alle drei Transaktionsarten soll hier ein Blick auf die wohl umfassendste Transaktionsart T_NEW_CATALOG deren Aufbau zeigen:

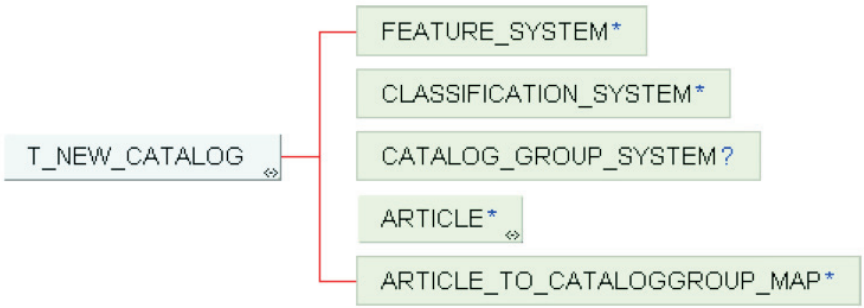


Abbildung 2.4: Struktur der BMEcat-Transaktionsart T_NEW_CATALOG

Das erste Element der Transaktionsart T_NEW_CATALOG ist FEATURE_SYSTEM, das es in den älteren Versionen des Standards ermöglichte, Merkmalsgruppensysteme abzubilden. Es ist in dieser Version nur noch aus Kompatibilitätsgründen vorhanden und wird in der kommenden Version 2.0 durch das zweite Element CLASSIFICATION_SYSTEM ersetzt. Das neue Element ermöglicht es dem Standard Klassifikationssysteme, wie z.B. eCl@ss, vollständig abzubilden. Mit dem Element CATALOG_GROUP_SYSTEM kann eine hierarchische Gruppenstruktur abgebildet werden, der einzelne Artikel zugeordnet werden können. Im Element ARTICLE werden dann die eigentlichen Informationen zu den einzelnen Produkten abgelegt. Das Element ARTICLE ist je nach ausgewähltem

Transaktionstyp unterschiedlich aufgebaut und somit kontextabhängig. Mit dem Element ARTICLE_TO_CATALOGGROUP_MAP kann dann jedes im Katalog enthaltene Produkt einem Kataloggruppensystem zugeordnet werden.
Die weitere Struktur wird Preisinformationen betreffend im nächsten Unterkapitel bzw. in der Spezifikation von BMEcat 1.2 [BMEcat1] ausführlich beschrieben.

Struktur der Preisinformationen

Ein Katalogdokument nach BMEcat umfasst in der aktuellen Version 1.2 neben den allgemeinen Informationen im Katalogkopf auch Informationen zu den Produkten und den zugeordneten Preisen, die für das einzelne Produkt gültig sind. Für die zu erstellende XML-Datenstruktur sind insbesondere Bestell- und Preisinformationen wichtig, weil diese noch nicht im Software-System viaMEDICI EPIM vorliegen. Sie sollen auf den folgenden Seiten als Vorarbeit zur Analyse der Anforderungen an die Datenstrukur näher betrachtet werden.
Bei genauerer Betrachtung der Spezifikation können sowohl im Katalogkopf als auch in den Transaktionselementen in Bezug auf Bestellinformationen und Preise wichtige Informationen gefunden werden.

Im Katalogkopf (siehe Bild Seite 29) sind insbesondere die Elemente CATALOG, BUYER, AGREEMENT und SUPPLIER von Bedeutung.
Im Element CATALOG werden allgemeine Daten zum Katalogdokument selbst gespeichert. Dazu gehören folgende Elemente:

Feld	**Kard.**	**Beschreibung**
LANGUAGE	1:1	Sprache des Katalogdokumentes
CATLOG_ID	1:1	eindeutige ID des Katalogdokumentes, wird vom Ersteller vergeben
CATALOG_VERSION	1:1	Version des Katalogdokumentes
CATALOG_NAME	0:1	Name des Katalogdokumentes
DATETIME	0:1	das Datum der Erstellung des Katalogdokumentes, enthält Unterelemente für das Datum, die Uhrzeit und die Zeitzone
TERRITORY	1:N	territoriale Verfügbarkeit des Dokumentes
CURRENCY	0:1	Default-Währung des Dokumentes
MIME_ROOT	0:1	Basisverzeichnis für multimediale Daten
PRICE_FLAG	0:N	Preiskennzeichen, das über ein Attribut kennzeichnet, ob die Preise z.B. alle inkl. Frachtkosten sind

Tabelle 2.6: BMEcat: Aufbau des Elementes HEADER

Die im CATALOG-Element vorhandenen Datenstrukturen bilden zwar nicht

direkt Preisinformationen ab, sie dienen aber der Identifizierung des Katalogdokumentes, das die Preise beinhaltet und sind deshalb auch für die zu erstellende Struktur nicht uninteressant. Darüberhinaus sind in diesem Element auch einige Defaultwerte definierbar, auf die zurück gegriffen werden kann, wenn direkt bei den Preisinformationen zu den einzelnen Artikeln nichts weiter definiert ist.
In den HEADER-Elementen BUYER und SUPPLIER sind überwiegend Daten enthalten, die die Katalog liefernde Partei (SUPPLIER) und die Katalog empfangende Partei (BUYER) kennzeichnen. Die beiden genannten Elemente sind nochmals in eine ID, den jeweiligen Namen und ein Element für Adressdaten, das nochmals verschiedene Unterelemente enthält, aufgeteilt. Auch hier kann kein direkter Bezug zu Preisinformationen gefunden werden. Vielmehr helfen diese Daten dabei, die Preisinformationen einem Handelspartner-Paar zuzuordnen.
Ein im Bezug auf Preisinformationen wichtiges Element, ist das Element AGREEMENT, das einen Verweis auf weitere Verträge mit zwischen beiden Handelspartnern getroffenen Zusatzvereinbarungen beinhaltet. Es ist nochmals aufgeteilt in eine ID und zwei Zeit-Elemente, die den Gültigkeitszeitraum für den jeweiligen Vertrag festlegen.

In den Transaktionselementen (siehe Bild Seite 30) ist vor allem das Element ARTICLE interessant, das selbst wieder kontextabhängig Elemente enthält. Interessant im Hinblick auf Preisinformationen sind die Elemente SUPPLIER_AID, ARTICLE_ORDER_DETAILS und ARTICLE_PRICE_DETAILS.
Das Element SUPPLIER_AID ist als einziges der drei näher zu betrachtenden Elemente in allen drei Transaktionsarten vorhanden und enthält eine eindeutige Artikelnummer des liefernden Unternehmens.
Das Element ARTICLE_ORDER_DETAILS enthält mehrere Elemente, die nähere Informationen zu den Bestellkonditionen und Verpackungsmodalitäten enthalten. Das obere Bild auf Seite 32 zeigt den Aufbau des Elementes.

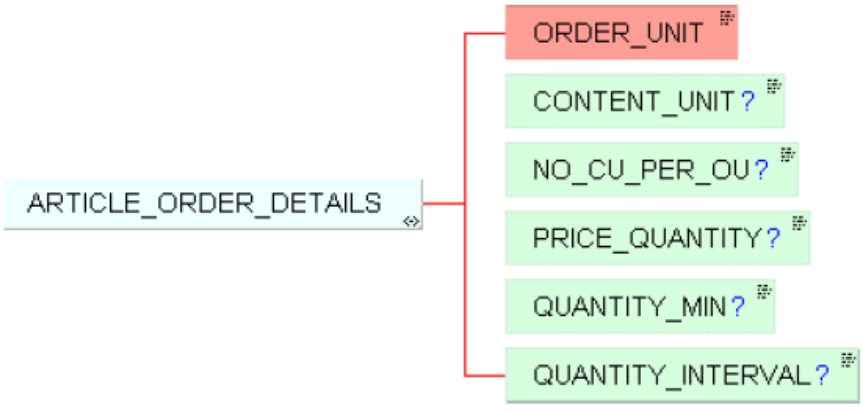


Abbildung 2.5: Aufbau des Elementes ARTICLE_ORDER_DETAILS

Die in ARTICLE_ORDER_DETAILS gespeicherten Daten haben keinen direkten Zusammenhang mit Preisinformationen. Sie umfassen aber Informationen, die mit Bestellvorgängen zusammenhängen und nicht in viaMEDICI EPIM vorhanden sind. Aus diesem Grund sind sie für die weiteren Betrachtungen in Hinblick auf die zu erstellende Datenstruktur sehr interessant.
Die nun folgende Tabelle erklärt die einzelnen enthaltenen Elemente näher:

Feld	**Kard.**	**Beschreibung**
ORDER_UNIT	1:1	Einheit, in der der Artikel bestellt werden kann
CONTENT_UNIT	0:1	Einheit des Atikels innerhalb der Bestelleinheit
NO_CU_PER_OU	0:1	Anzahl der Inhaltseinheiten pro Bestelleinheit des Artikels
PRICE_QUANTITY	0:1	Anzahl der Bestelleinheiten, auf die sich der angegebene Preis bezieht
QUANTITY_MIN	0:1	Mindestbestellmenge
QUANTITY_INTERVAL	1:1	Zahl, die angibt, in welcher Staffelung der Artikel bestellt werden kann

Tabelle 2.7: BMEcat: Aufbau des Elementes ARTICLE_ORDER_DETAILS

Im Element ARTICLE_PRICE_DETAILS werden nun die Preisinformationen zu einem Artikel abgelegt. Das untere Bild auf Seite 33 zeigt den Aufbau des Elementes.

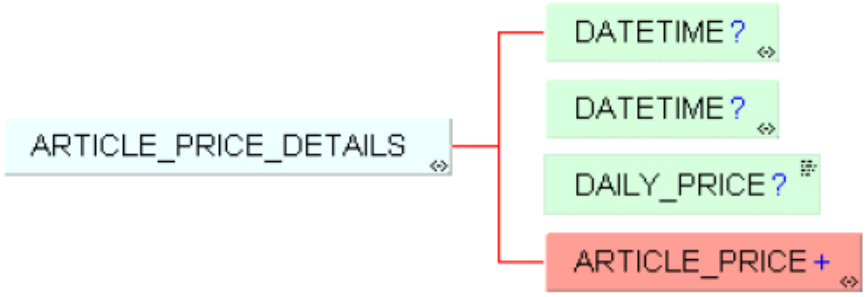


Abbildung 2.6: Aufbau des Elementes ARTICLE_PRICE_DETAILS

Die beiden im Element ARTICLE_PRICE_DETAILS enthaltenen DATETIME-Elemente legen den Gültigkeitszeitraum fest und sind wie alle Elemente diesen Typs nochmals in Elemente für das Datum, die Uhrzeit und die Zeitzone unterteilt. Beim Element DAILY_PRICE handelt es sich um einen Boolschen Wert, der mit dem Wert true anzeigt, dass sich der Preis dieses Artikels z.B. durch

Materialpreisschwankungen häufig ändert.
Die Preise selbst sind im Element ARTICLE_PRICE definiert, das mindestens einmal vorhanden sein muss und aus mehreren Elementen besteht, die in der folgenden Tabelle näher beschrieben werden:

Feld	**Kard.**	**Beschreibung**
PRICE_AMOUNT	1:1	Höhe des Preises
PRICE_CURRENCY	0:1	Währung des Preises, überschreibt den Defaultwert im HEADER
TAX	0:1	Faktor für die Umsatzsteuer
PRICE_FACTOR	0:1	der (Rabatt-)Faktor, muss immer mit der Höhe des Preises multipliziert werden, um den Gesamtpreis zu erhalten
LOWER_BOUND	0:1	untere Staffelgrenze, die obere Staffelgrenze wird durch das Element LOWER_BOUND des nächsten Preises definiert
TERRITORY	0:N	Gebiet, in dem der Preis gültig ist; überschreibt das Element im Feld TERRITORY im HEADER

Tabelle 2.8: BMEcat: Aufbau des Elementes ARTICLE_PRICE

Das Element ARTICLE_PRICE hat desweiteren noch ein Attribut namens *price_type*, mit dessen Hilfe eine nähere Angabe zum Preistyp hinterlegt werden kann. Unter Preistyp versteht sich an dieser Stelle, um was für einen Preis es sich handelt. Beispiele könnten ein Netto-Listenpreis oder ein kundenspezifischer Preis sein.
Alle weiteren Elemente in der BMEcat-Katalogstruktur sind für die Speicherung von Bestell- und Preisinformationen nicht interessant und werden deshalb hier nicht näher beschrieben. Eine vollständige Elementreferenz findet sich in der Spezifikation von BMEcat 1.2 [BMEcat1].

2.4.2 Weitere Katalogstandards

Nach der genauen Betrachtung des deutschen Katalogstandards BMEcat sollen nun mit cXML (commerce eXtensible Markup Language) und OAGIS (Open Applications Group Integration Specification) zwei weitere Austauschstandards kurz betrachtet werden.

cXML

cXML ([cXML1]) wurde unter der Leitung von Ariba ([Ariba1]), einem führenden Hersteller von u.a. eProcurement-Systemen, in Zusammenarbeit mit mehreren Firmen wie z.B. AMD, Philips, Nestle, etc. entwickelt und Anfang Februar 1999 veröffentlicht. Aktuell liegt die Version 1.2.009 unter [cXML1] zum Download bereit.
cXML zählt derzeit zu einem der weltweit am weitesten verbreiteten B2B-Austauschstandards und verwendet das XML-Format für den Datenaustausch. Die Beschreibung der XML-Datenstruktur erfolgt mittels DTDs.
cXML ist im Vergleich zu BMEcat um einiges umfangreicher, da sich neben den Katalogdaten auch Geschäftsprozesse wie Bestellungen von Produkten und das Erstellen zugehöriger Rechnungen darstellen lassen. Der Austausch der dafür notwendigen Daten kann mittels zweier Transaktionsarten erfolgen:

- **Request-Response Modell**
  Dieses Transaktionsmodell ist an das HTTP-Protokoll gebunden. Ein Client A baut dabei eine HTTP-Verbindung zu einem Server B auf und schickt eine Anfrage (Request) an den Server B. Dieser verarbeitet die Anfrage mittels einem Hilfsprogramm und schickt die Antwort (Response) an den Client A zurück.

- **One-Way (asynchrones) Modell**
  Ein Client A verschlüsselt eine Anfrage in einem zuvor vereinbarten Übertragungsprotokoll und sendet diese an einen Server B. Da dieser das Übertragungsprotokoll kennt, kann er nun die Anfrage entschlüsseln, bearbeiten und entsprechend beantworten.

Dabei sind cXML-Dokumente immer nach dem selben Muster aufgebaut. Sie bestehen aus einem HEADER mit allgemeinen Informationen zu den beiden am Austausch beteiligten Parteien. Diesem HEADER folgt dann je nach gewähltem Transaktionsmodell ein Element mit den Daten für die jeweilige Transaktion. Beim Request-Response Modell sind das die beiden Elemente REQUEST und RESPONSE und beim One-Way Modell das Element MESSAGE.
Es ist derzeit auch des einizge Protokoll, mit dem sich ein sogenannter PunchOut realisieren lässt. Ein PunchOut ist praktisch eine Auslagerung der Produkt- und Preisinformationen in ein über einen Link erreichbares System des Anbieters, das über das Internet für den Kunden erreichbar ist und so die aktuellen Informationen zur Verfügung stellt. Damit wird auf ein Übertragen der eigentlichen Produkt- und Preisdaten im XML-Dokument verzichtet. In der Spezifikation ([cXML1]) konnten weiter keine Informationen zur Speicherung von Produkt- und Preisinformationen ausfindig gemacht werden, was diesen B2B-Austauschstandard für die weitere Betrachtung im Rahmen dieser Arbeit unrelevant werden lässt.

OAGIS

Die Open Applications Group ([OAGIS1]) als Schöpfer des Austauschstandards OAGIS ist eine gemeinützige Vereinigung verschiedener Firmen, deren Schwerpunkt es ist, sich mit der Entwicklung und Abbildung von Geschäftspraktiken und -prozessen in XML im Bereich des eBusiness und der Anwendungsintegration zu befassen. Mitglieder der Open Applications Group sind viele namhafte Firmen, zu denen z.B. DaimlerChrysler, IBM, Oracle, Peoplesoft und viele mehr gehören.
Mit OAGIS (Open Applications Group Integration Specification), die derzeit in der aktuelen Version 8.0 vorliegt, hat diese Gruppe einen XML-basierten Austauschstandard spezifiziert, dessen zentrale Dokumente sogenannte „Business Object Documents“ oder kurz BODs sind. BODs sind nach folgendem Schema aufgebaut:

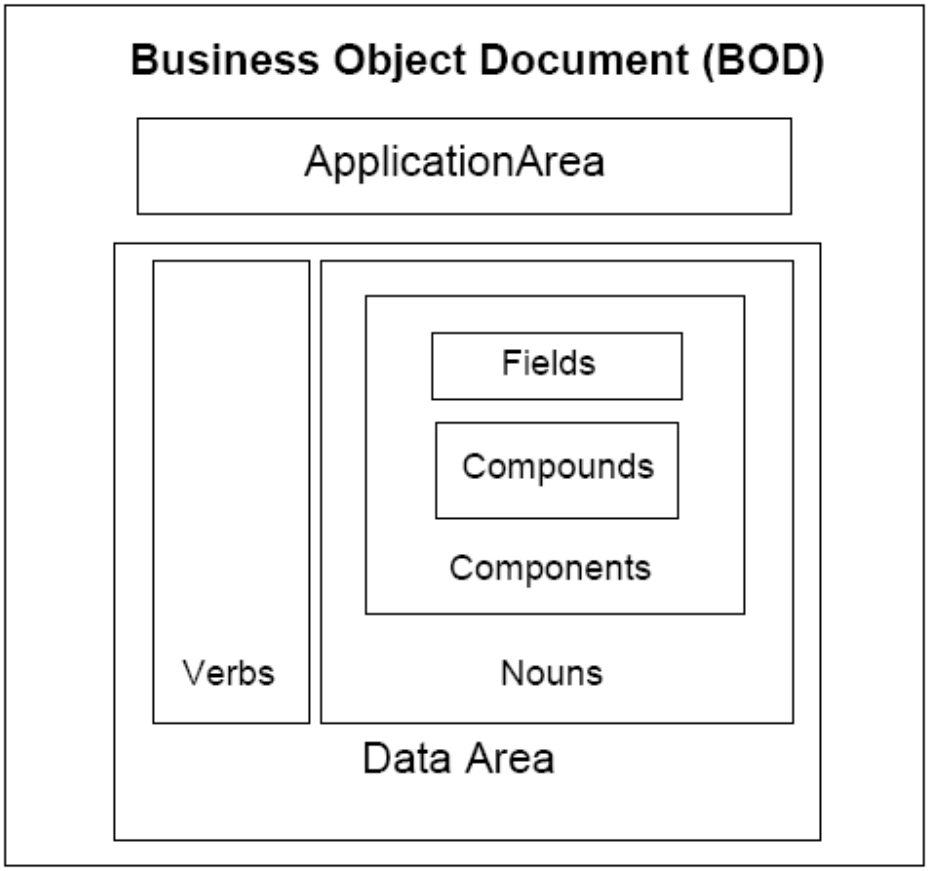


Abbildung 2.7: OAGIS - Aufbau eines Business Object Documents

BODs werden dabei grundsätzlich in zwei Bereiche, die *ApplicationArea* und die

DataArea aufgeteilt.
Die *ApplicationArea* enthält allgemeine Informationen zum BOD und zu den daran beteiligten Parteien und kann somit als eine Art Dokumentenkopf angesehen werden.
In der *DataArea* werden dann die auszutauschenden Daten abgebildet. Eine *DataArea* splittet sich dabei nochmals in *Verbs* und *Nouns*. Dabei definiert ein *Verb* die Aktion, die auf einem Transaktionsobjekt (*Noun*) ausgeführt werden soll. *Nouns* werden nochmals in *Fields* und *Components* aufgeteilt. Dabei entsprechen die *Fields* den einfachen Datentypen, die in *Components*, also einer Art Container, zu Blöcken zusammengefasst werden können.
Eine weitere Besonderheit von OAGIS ist die Tatsache, dass alle bereits vordefinierten *Components* mittels erweiternden XML Schemata um firmenspezifische Elemente ergänzt werden können. Dazu enthält ein *Component* ein Element namens *UserArea*.

Zur Darstellung elektronischer Produktkataloge wurde ein *Noun* namens *ElectronicCatalog* definiert, das folgendermaßen aufgebaut ist:

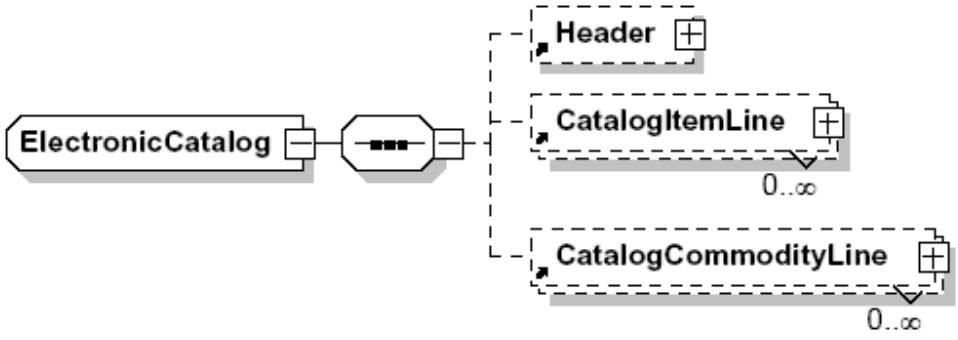


Abbildung 2.8: OAGIS - grober Aufbau des Nouns ElectronicCatalog

Im *Header* des Kataloges befinden sich allgemeine Informationen zum Dokument, den beteiligten Parteien und dem verwendeten Klassifikationsschema. In den Elementen vom Typ *CatalogItemLine* werden dann die eigentlichen Produktdaten, darunter auch die Preisinformationen, abgelegt. Handelt es sich bei den darzustellenden Produkten um Gebrauchsgegenstände, sollten die Informationen besser in Elementen vom Typ *CatalogCommoditiesLine* abgelegt werden.
Die für diese Diplomarbeit relevanten Preisinformationen werden innerhalb eines Elementes des Typs *CatalogItemLine* in einem oder mehreren Unterelementen des Typs *ItemPrice* gespeichert. Das Element ist aus folgenden *Components* und *Fields* aufgebaut:

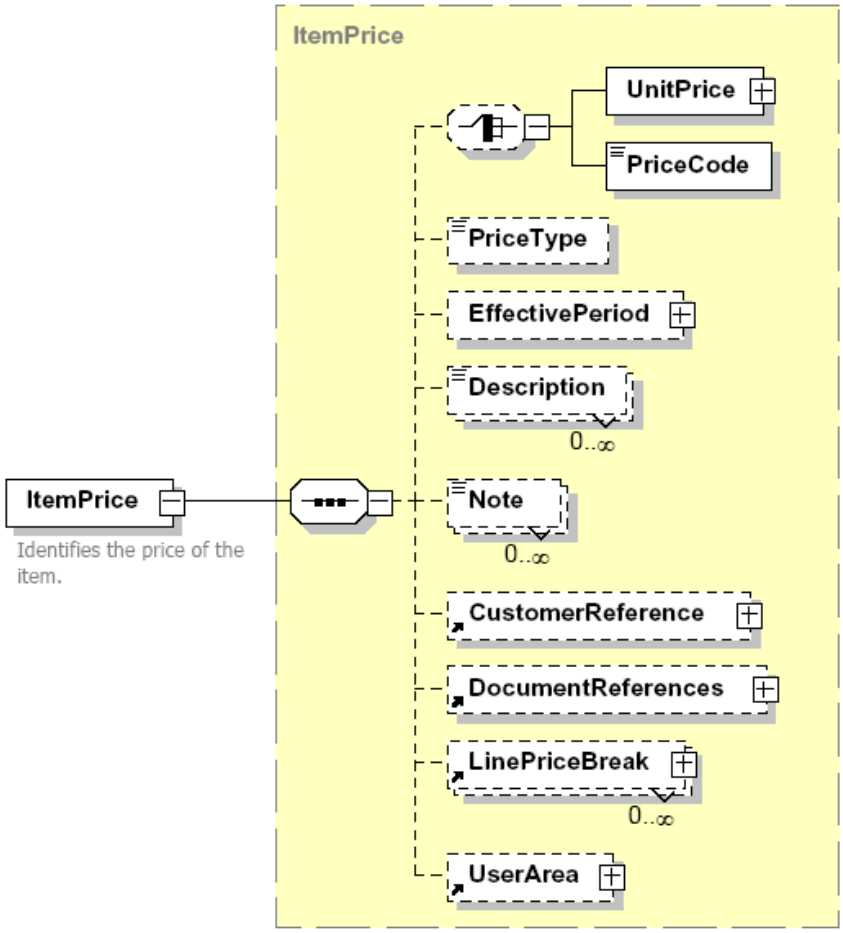


Abbildung 2.9: OAGIS - Aufbau des Elementes bzw. Component ItemPrice

Das Element *UnitPrice* legt fest, was eine Mengeneinheit des beschriebenen Produktes kostet. Alternativ dazu kann über das Element *PriceCode* auch ein Code angegeben werden. Das Element *PriceType* legt fest, um welchen Typ von Preis es sich handelt. Im *Field EffektivePeriod* kann der Gültigkeitszeitraum des Preises definiert werden. Mit den beiden Elementen *Description* und *Note* können der Preisinformation eine Beschreibung bzw. freie Notizen hinzugefügt werden. Das Element *CustomerReference* enthält eine Referenz auf den jeweiligen Kunden, für den die hinterlegten Preisinformationen gültig sind. Mit dem Element *DocumentReference* ermöglicht es OAGIS auf andere BODs zu verweisen, mit denen dieser Preis assoziiert werden kann. Über den *Component LinePriceBreak* lassen sich auf vielfältige Art und Weise Beeinflussungen des Preises durch bestimmte Mengen, Beträge, etc. abbilden. Als letztes Element befindet sich in *ItemPrice* das Element *UserArea*, dass die Ergänzung der vordefinierten Elemente um anbieterspezifische Elemente ermöglicht.

Genauere Informationen zu den einzelnen Elementen des Austauschstandards OAGIS bietet die umfangreiche Dokumentation, die bei [OAGIS1] heruntergeladen werden kann.

2.4.3 Fazit

Zum Schluss kann festgehalten werden, dass es zwei unterschiedliche Typen von Austauschstandards für elektronische Kataloge gibt. Die einen Standards, zu denen BMEcat und OAGIS als Vertreter betrachtet wurden, versuchen die Preisinformationen direkt abzubilden. Hier sind derzeit noch Mankos bei der Darstellung von Preisbeeinflussungen durch Produktvariationen und Produktkonfigurationen festzustellen, für die aber mit der Ankündigung der neuen Version 2.0 des Katalogstandards BMEcat eine Lösung in Aussicht steht.
cXML hingegen, das derzeit der einzige Vertreter der anderen Art von Austauschstandards ist, verweist über ein sogenanntes PunchOut-Verfahren auf ein die Produkt- und Preisinformationen näher spezifizierendes System, das meist über das Internet angesprochen werden kann. Hier gibt es allerdings keine Datenstrukturen, die Preisinformationen aufnehmen können.

3 Erstellung einer XML-Vorlage zur Darstellung von Preisinformationen

Nachdem im Kapitel 2 (Seite 9) nun detailiert die Grundlagen der Preisbildung und einge Beispiele für Quellen von Preisinformationen und Strukturen für die Weiterverarbeitung betrachtet wurden, gilt es nun in diesem Kapitel die XML-Struktur für die Preisinformationen in viaMEDICI EPIM festzulegen.
Bevor diese gesuchte Struktur nun festgelegt werden kann, muss zuerst einmal entschieden werden, welche Sprache zur Definition einer XML-Datenstruktur verwendet werden soll.
Danach müssen in einer detaillierten Analyse der genannten Beispiele für Datenquellen und Datenstrukturen zur Weiterverarbeitung aus Kapitel 2 (Seite 9), die XML-Elemente und -Attribute der Preis-Datenstruktur festgelegt werden.
Dieses Kapitel abschließend soll dann eine XML-Struktur entstehen, die den meisten Anforderungen der Preisinformationen entspricht und so der Datenhaltung in viaMEDICI EPIM und der Weiterverarbeitung der Daten gerecht wird.

3.1 DTD vs. XML Schema

Im ersten Schritt soll nun eine geeignete Sprache zur Strukturierung des XML-Dokumentes gesucht werden. Zur Strukturierung von XML-Dokumenten sind derzeit zwei Definitionssprachen vom W3-Konsortium empfohlen: Die bereits etwas ältere Document Type Definition (DTD), die erstmals 1998 mit XML 1.0 W3C-Empfehlung wurde, und die XML Schema Definitionssprache, die 2001 W3C-Empfehlung wurde. Beide sollen nun im folgenden näher betrachtet werden, bevor eine Entscheidung für eine der beiden Sprachen fällt.

3.1.1 DTD

Wie bereits erwähnt, wurde die Definitionssprache Document Type Definition (DTD) bereits mit XML 1.0 Empfehlung des W3C. Eine DTD selbst entspricht nicht dem XML-Format, was durchaus als Nachteil gesehen werden kann, da zur Erstellung und Validierung einer DTD unter Umständen andere Werkzeuge benötigt werden als zur Erstellung eines XML-Dokumentes. Sie ist von ihrem

Aufbau eher mit der Backus-Naur-Form vergleichbar, mit der in der Regel die Syntax einer Programmiersprache beschrieben werden kann.
Im Prinzip hat sie auch eine ähnliche Funktion: Mit ihr ist es möglich, die Datenstruktur von XML-Dokumenten festzulegen. Dies erfolgt in erster Linie durch die Definition von Elementen und Attributen. Elemente selbst können neben dem Inhalt auch ein oder mehrere Elemente beinhalten. Bei der Angabe der Sequenz der beinhalteten Elemente kann auch gleichzeitig die Reihenfolge der Elemente festgelegt werden.

```
<!ENTITY % STRING "(#PCDATA)">
...
<!ELEMENT ADDRESS (NAME?, NAME2?, NAME3?, CONTACT?, STREET?, ZIP
    ?, BOXNO?, ZIPBOX?, CITY?, STATE?, COUNTRY?, PHONE?, FAX?,
    EMAIL?, PUBLIC_KEY?, URL?, ADDRESS_REMARKS?)>
<!ATTLIST ADDRESS
  type %ADDRESS_QUALIFIERS; #REQUIRED
>
...
<!ELEMENT NAME %STRING;>
<!ELEMENT NAME2 %STRING;>
<!ELEMENT NAME3 %STRING;>
<!ELEMENT CONTACT %STRING;>
<!ELEMENT ZIP %STRING;>
<!ELEMENT BOXNO %STRING;>
<!ELEMENT ZIPBOX %STRING;>
<!ELEMENT STREET %STRING;>
<!ELEMENT CITY %STRING;>
<!ELEMENT STATE %STRING;>
<!ELEMENT COUNTRY %STRING;>
<!ELEMENT PHONE %STRING;>
<!ELEMENT FAX %STRING;>
<!ELEMENT EMAIL %STRING;>
<!-- new in version 1.2 -->
<!ELEMENT PUBLIC_KEY %STRING;>
<!ATTLIST PUBLIC_KEY
  type CDATA #REQUIRED
>
<!ELEMENT URL %STRING;>
<!ELEMENT ADDRESS_REMARKS %STRING;>
```

Listing 3.1: Beispiel einer DTD aus der BMEcat-DTD

Im Laufe der Zeit sind durch die weite Verbreitung von XML die Anforderungen an eine XML-Definitionssprache deutlich gestiegen. So reichen die Möglichkeiten der DTDs bei weitem nicht mehr aus.
Ein großer Schwachpunkt der DTD sind die weitgehenst fehlenden Datentypen. Der einzige Datentyp, der auch Inhalte und Werte aufnehmen kann, ist #PCDATA. Dieser Datentyp kann im allgemeinen nur Strings aufnehmen und ist somit

bei der oftmals geforderten Typisierung von Inhalten wenig hilfreich.
DTDs können im Gegensatz zu XML-Dokumenten auch keine Namespaces benutzen. Namespaces sind Namensräume, die es ermöglichen, dieselben Element- und Attributnamen mehmals zu benutzen. Allerdings immer im Kontext zu einem anderen Namensraum.
Ebenfalls von Nachteil ist, dass DTDs keine Vererbung bzw. Wiederverwendung kennen. Auch Einschränkungen bzw. Erweiterungen bereits bestehender Elemente sind so nicht möglich.

3.1.2 XML Schema

Die Definitionssprache XML Schema ist mit ihrem Empfehlungsjahr 2001 deutlich jünger als DTDs. Ein XML Schema ist selbst ein XML-Dokument, dass somit auch nach den Regeln von XML selbst validiert werden kann.

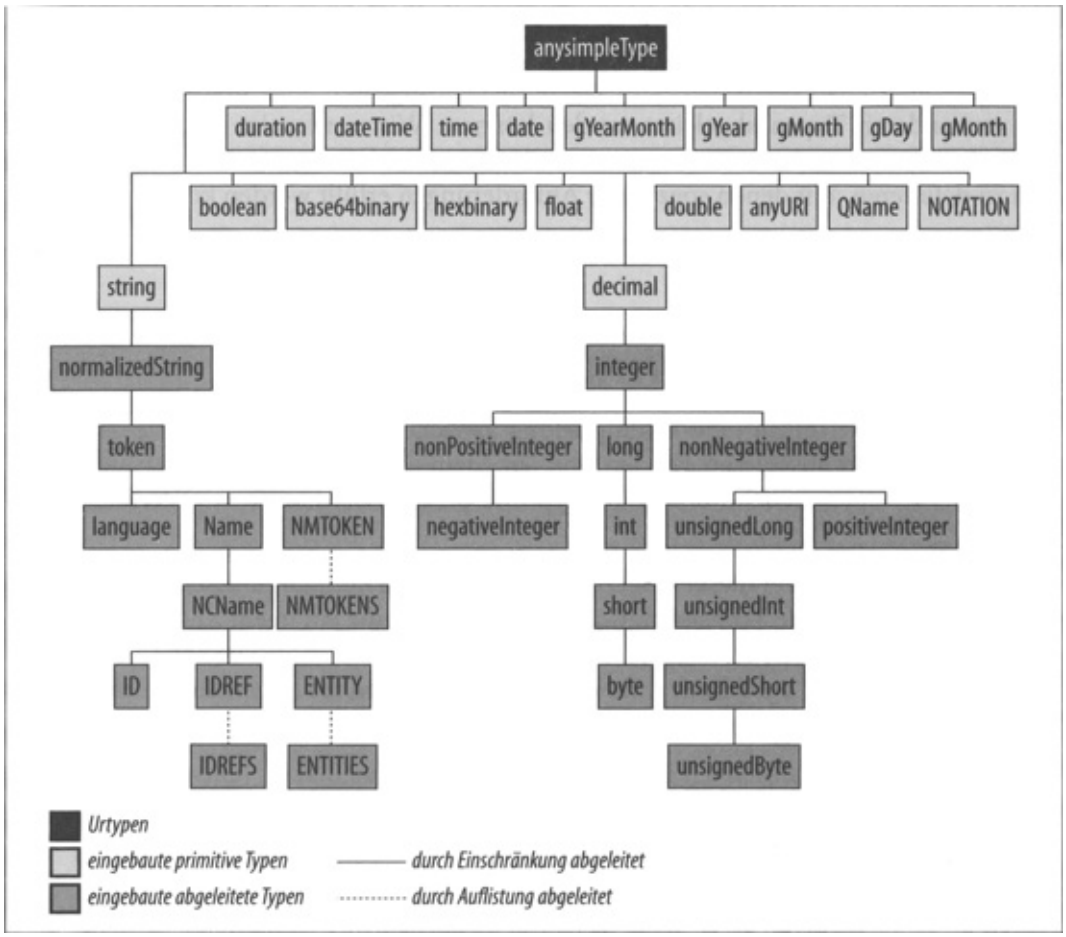


Abbildung 3.1: Hierarchie der Datentypen in XML Schema

Da ein XML Schema selbst ein XML-Dokument ist, kann es auch mit verschiedenen Namensräumen umgehen. Dadurch ist es möglich, auf Definitionen in externen Dokumenten zu verweisen. Der Standard-Namensraum, der in den meisten Dokumenten zu XML Schema zu finden ist, ist xsd oder auch xs.
Ein XML Schema deklariert wie auch DTDs Elemente und Attribute eines XML-Dokumentes. Da XML Schema aber eine strengere Typisierung von Inhalten hat, lassen sich die Datentypen der Elementinhalte bzw. Attributwerte bis hin zum regulären Ausdruck beliebig genau festlegen.
Neben vielen Standard-Datentypen ermöglicht es XML Schema auch, eigene Datentypen zu definieren. Dies geschieht in der Regel durch Einschränken oder Erweitern von bestehenden Datentypen. Diese Datentypen können komplex, das heißt aus mehreren Elementen bestehend, oder einfach, das heißt den direkten Inhalt aufnehmend, sein. Mit Hilfe dieser Datentypen lassen sich dann Elemente deklarieren, deren Inhalt genau der im Datentyp enthaltenen Definition entsprechen muss.
Dazwischen sind auch wie bei den DTDs gemischte Elemente möglich, die sowohl weitere Elemente, als auch Daten beinhalten können. Durch die Anordnung der einzelnen Elemente im Dokument oder einem komplexen Datentyp kann implizit auch gleich festgelegt werden, in welcher Reihenfolge die Datencontainer dann im XML-Dokument angeordnet werden müssen. Es lassen sich in einem XML Schema auch komplexe Elemente deklarieren, deren Reihenfolge der Kindelemente nicht festgelegt ist. Auch Auswahllisten können innerhalb von komplexen Elementen deklariert werden.
Besonders bei den komplexen Datentypen, aber auch bei den einfachen Typen haben XML Schemata den Vorteil, dass genau festgelegt werden kann, wie oft ein Element vorkommen kann bzw. darf. In XML Schemata lassen sich auch leere Elemente deklarieren.

```
1  <!-- BSP: Definition des komplexen Datentyps typeADDRESS -->
2  <xsd:complexType name="typeADDRESS">
3    <xsd:sequence>
4        <xsd:element ref="NAME" minOccurs="0"/>
5        <xsd:element ref="NAME2" minOccurs="0"/>
6        <xsd:element ref="NAME3" minOccurs="0"/>
7        <xsd:element ref="CONTACT" minOccurs="0"/>
8        <xsd:element ref="STREET" minOccurs="0"/>
9        <xsd:element ref="ZIP" minOccurs="0"/>
10       <xsd:element ref="BOXNO" minOccurs="0"/>
11       <xsd:element ref="ZIPBOX" minOccurs="0"/>
12       <xsd:element ref="CITY" minOccurs="0"/>
13       <xsd:element ref="STATE" minOccurs="0"/>
14       <xsd:element ref="COUNTRY" minOccurs="0"/>
15       <xsd:element ref="PHONE" minOccurs="0"/>
16       <xsd:element ref="FAX" minOccurs="0"/>
```

```
17        <xsd:element ref="EMAIL" minOccurs="0"/>
18        <xsd:element ref="PUBLIC_KEY" minOccurs="0" maxOccurs="
              unbounded"/>
19        <xsd:element ref="URL" minOccurs="0"/>
20        <xsd:element ref="ADDRESS_REMARKS" minOccurs="0"/>
21    </xsd:sequence>
22  </xsd:complexType>
23  ...
24  <!-- BSP: Anwendung des komplexen Datentyps typeADDRESS -->
25  <xsd:element name="BUYER">
26    <xsd:complexType>
27      <xsd:sequence>
28        <xsd:element ref="BUYER_ID" minOccurs="0"/>
29        <xsd:element ref="BUYER_NAME"/>
30        <xsd:element name="ADDRESS" minOccurs="0">
31          <xsd:complexType>
32            <xsd:complexContent>
33              <xsd:extension base="typeADDRESS">
34                <xsd:attribute name="type" fixed="buyer"/>
35              </xsd:extension>
36            </xsd:complexContent>
37          </xsd:complexType>
38        </xsd:element>
39      </xsd:sequence>
40    </xsd:complexType>
41  </xsd:element>
42  ...
43  <!-- BSP: Definition des einfachen Elementes NAME -->
44  <xsd:element name="NAME">
45    <xsd:simpleType>
46      <xsd:restriction base="dtSTRING">
47        <xsd:maxLength value="50"/>
48        <xsd:minLength value="1"/>
49      </xsd:restriction>
50    </xsd:simpleType>
51  </xsd:element>
```

Listing 3.2: Beispiel eines XML Schema aus der BMEcat-XSD

In XML Schemata können auch Ansätze der Objektorientierung verwendet werden. Durch Ersetzungsgruppen ist es z.B. möglich, verschiedene Elementdefinitionen unter einem Namen zu referenzieren. Dieses Verfahren entspricht in der objektorientierten Programmierung dem Vererbungsprinzip von Klassen an Unterklassen.
Durch die vielfältigen Möglichkeiten ist es für den menschlichen Leser eines Schemas relativ schwer, die Datenstruktur auch wirklich zu verstehen. Aus diesem Grunde wurde die XML Schema Definitionssprache um Dokumentationsmöglichkeiten ergänzt, die es ermöglichen, dass eine mit ihr erstellte Datenstruktur sowohl von Menschen als auch von Dokumentationswerkzeugen lesbar ist.

3.1.3 Fazit

Werden die beiden XML-Definitionssprachen miteinander verglichen, fällt eindeutig auf, wieviel weiter die XML Schema Definitionssprache im Vergleich zu den Dokumententyp-Definitionen (DTDs) entwickelt ist. Insbesondere das Vorhandensein von Datentypen und die Möglichkeit, eigene Datentypen zu definieren, bieten speziell im Bereich des Datenaustausches zwischen zwei Systemen viele Vorteile.
Weitere Vorteile bieten natürlich auch die objektorientierten Ansätze der XSD, die vor allem eine Wiederverwendung inklusive Anpassung mittels Erweiterung (z.B. durch Attribute) oder Einschränkung (z.B. der Inhaltsmuster) von bereits deklarierten Elementen ermöglicht.
Der größte Vorteil, den XML Schema gegenüber DTDs bietet, ist aber, dass eine XML Schema-Datei selbst ein XML-Dokument ist und mit bereits für die Erstellung von XML-Dokumenten benötigten Werkzeugen bearbeitet werden kann und somit keine weiteren Tools benötigt werden.
Wenn man die beiden Beispiele (Seite 41 und Seite 43), die das selbe Element ADDRESS deklarieren, vergleicht, fällt aber sofort auf, dass die vielen Vorteile eines XML Schemas auch zu einem Nachteil führen: Die Definitionsdatei ist um einiges aufgeblähter und durch Menschen nur schwer zu lesen. Doch hier bietet XML Schema viele Möglichkeiten, den verfassten Code zu dokumentieren und so sowohl für die Maschine als auch für den Menschen lesbar zu machen.
Diese überwiegenden Vorteile werden auch immer mehr im Bereich des eBusiness und dort insbesondere bei den Katalogstandards ausgenutzt. Immer mehr Standards, wie z.B. BMEcat [BMEcat1] bieten zu den schon längere Zeit verfügbaren DTDs seit Kurzem zusätzlich XML Schemata zur Definition der Datenstruktur an.
Die vielen Vorteile eines XML Schemas im Vergleich zu einer DTD und die immer größer werdende Verbreitung dieser Definitionssprache, insbesondere bei den für diese Arbeit wichtigen Zielformaten den Katalogstandards, machen es leicht, die Entscheidung für einen Einsatz der XML Schema Definitionssprache für den Entwurf der XML-Datenstruktur für Preisinformationen zu fällen.

3.2 Analyse der Anforderungen an die XML-Vorlage

Nach der Entscheidung für XML Schema als XML-Beschreibungssprache für das zu erstellende XML-Datenformat war es nun wichtig herauszufinden, welche Daten für eine Anzeige im System vaiMEDICI EPIM und die Ausgabe in einen Katalogstandard relevant sind. Dazu wurden die in der Recherche der ERP-Systeme und Katalogstandards gewonnenen Erkenntnisse nochmals genauer betrachtet und mit den von der viaMEDICI Software GmbH gestellten Anforderungen verglichen.

3.2.1 Zusammenfassung der Anforderungen

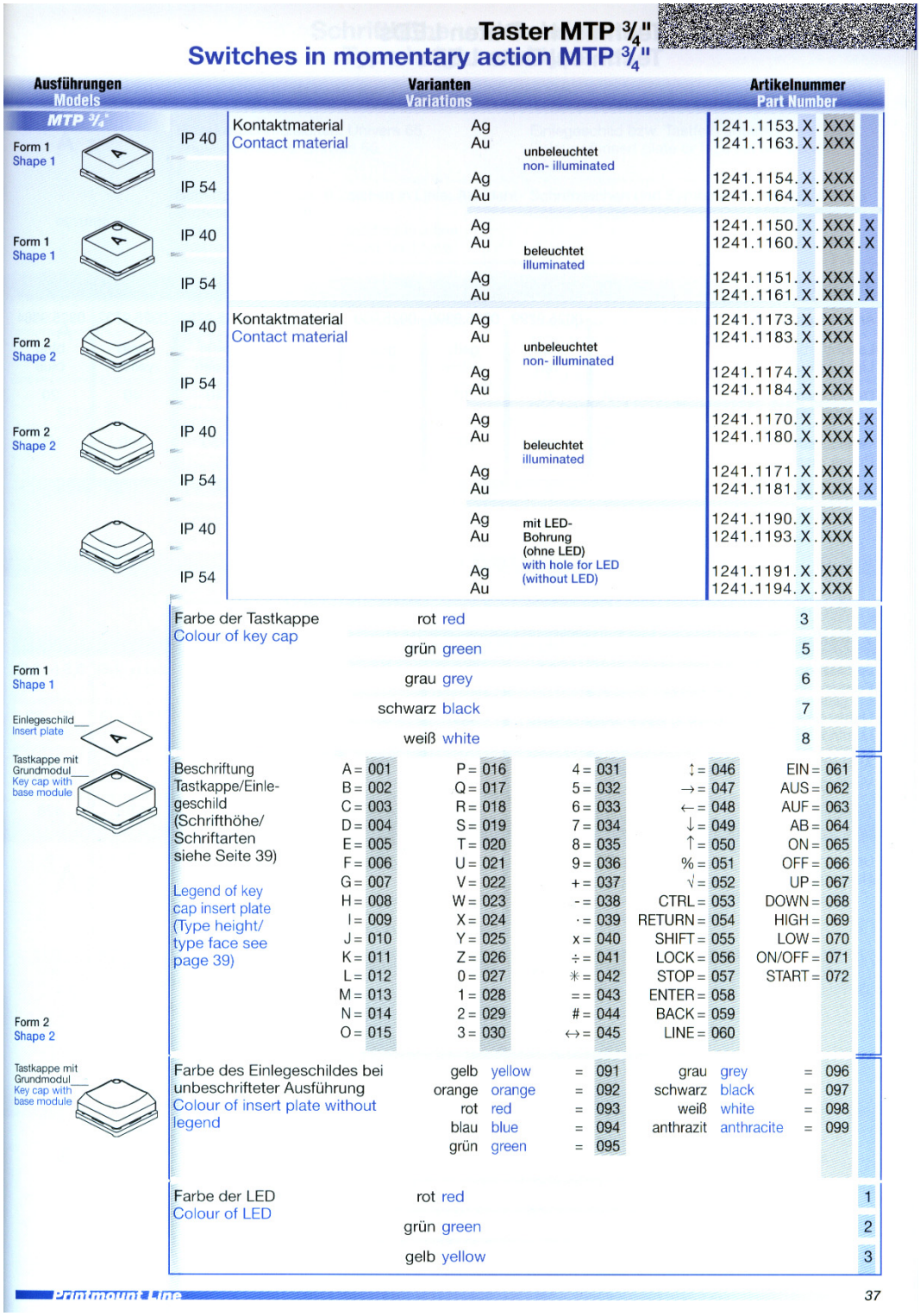
An dieser Stelle sollen zuerst die Anforderungen, die von Seiten der betreuenden Firma viaMEDICI Software GmbH an die XML-Vorlage gestellt wurden, kurz beschrieben werden.
Die XML-Vorlage soll in erster Linie Preisinformationen in der Form erfassen können, wie sie für den Export in Richtung elektronischer Marktplätze erforderlich sind. Aufgrund des derzeit überwiegend aus dem deutschsprachigen Raum stammenden Kundenstammes wird im folgenden der Hauptaugenmerk auf den deutschen Katalogstandard BMEcat in der Version 1.2 gelegt. Dabei soll im Gegensatz zu den Katalogdokumenten, die für jede Lieferanten-Einkäufer-Beziehung alle Produkte enthalten und somit ausschließlich kundenspezifisch sind, jeweils für jedes Produkt und jedes einkaufende Unternehmen ein eigenes XML-Dokument erstellt werden. Diese Anforderung liegt zum einen im Software-System viaMEDICI EPIM begründet, das die enthaltenen Daten artikelbezogen verwaltet und sich so die Preisdaten besser den restlichen Artikeldaten zuordnen lassen. Zum anderen wird durch den Einsatz eines kundenbezogenen Katalogstandards auch die Trennung nach Kunden sinnvoll, damit beim Export leichter ein kundenspezifisches Katalogdokument erzeugt werden kann. Das dritte Schlüsselkriterium, das neben dem Kunden und dem Artikel beachtet werden muss, kommt von Seiten des Katalogstandards BMEcat und ist die Sprache des Dokumentes.
Die Preisinformationen sollen weiter so ergänzt werden, dass eine Ansicht der Informationen im zu erstellenden Datenformat möglich ist, ohne Daten aus anderen Bereichen des Systemes hinzuzuziehen. Das heißt, dass auch einige Grundinformationen zum jeweiligen Produkt, wie z.B. Artikelbezeichnung und -beschreibung, etc., im XML-Dokument direkt gespeichert werden sollen. Darüberhinaus sollen natürlich auch Dokumentmerkmale wie die territoriale Verfügbarkeit, die Währung oder die Sprache des Dokumentes im Dokument selbst abgelegt werden können.
Weiter soll es möglich sein, unterschiedliche Produktvariationen, z.B. Stifte mit roter bzw. schwarzer Mine, und die dadurch evtl. mögliche Beeinflussung des Preises abzubilden. Dies ist insbesondere aus dem Gurnd notwendig, weil zum Kundenstamm der viaMEDICI Software GmbH auch einige Hersteller von z.B. Elektronikkomponenten und Elektromaschinen gehören, die verstärkt Produktvariationen einsetzen. Auch soll es möglich sein, eine als String verfasste Formel für die Preisberechnung im anzufertigenden Datenmodell abzulegen, die dann mit der entsprechenden Interpretation zur Berechnung eines Preises herangezogen werden kann.
Zum Schluss soll noch die Möglichkeit der Abbildung der Informationen geprüft werden, die in einem ERP-System, insbesondere in SAP, zur Berechnung von Preisinformationen hinterlegt sind, um besser belegen zu können, wie der berechnete Preis zustande kam und um die Möglichkeit einer Preisberechnung außerhalb des ERP-Systemes zu wahren.

3.2.2 Anforderungen durch den Katalogstandard BMEcat

Da eine Vielzahl an Produktinformationen inklusive Klassifikationsinformationen bereits direkt aus viaMEDICI EPIM ihren Weg über den Mediator in das Format des Katalogstandards BMEcat finden, müssen an dieser Stelle nur die Anforderungen der Preis- und Bestellinformationen, die bisher noch nicht in viaMEDICI EPIM vorliegen, näher betrachtet werden. Die Datenstrukturen der Preis- und Bestellinformationen innerhalb des BMEcat-Standards wurden bereits im Kapitel über den Katalogstandard BMEcat auf Seite 28 detailliert beschrieben. Da Preisinformationen auch in anderen Katalogstandards ähnlich den Möglichkeiten in BMEcat abgelegt werden können, wird für die folgenden Betrachtungen der BMEcat-Standard als ausreichend betrachtet.
Wichtig ist noch zu erwähnen, dass die von BMEcat verwendete Datenstruktur in erster Linie dafür ausgelegt wurde, dass dort für jeden Artikel nur komplett berechnete Preise bzw. auch Preisfaktoren abgelegt werden. Diese können sich noch je nach Bestellmenge in einer Mengenstaffel unterscheiden. Komplexe Berechnungsstrukturen, wie sie in der Regel in ERP-Systemen zu finden sind, können in Katalogstandards bisher noch nicht abgebildet werden.
Dadurch ergibt sich natürlich, dass die in einem Katalogstandard abzubildenden Preisinformationen entweder direkt vom ERP-System oder von einem der nachgelagerten Systeme, in diesem Fall entweder von viaMEDICI EPIM oder vom Mediator, berechnet werden müssen.

3.2.3 Anforderungen durch Abbildung von Variationen

Bereits in der Zusammenfassung wurde erwähnt, dass eine der Anforderungen der viaMEDICI Software GmbH ist, dass Produkte mit unterschiedlichen Variationen in der zu erstellenden Datenstruktur abgebildet werden können. Unterschiedliche Variationen eines Produktes werden dabei durch ein oder mehrere Produktmerkmale gebildet, die unterschiedliche Ausprägungen annehmen können. Wichtig ist dabei, dass auch Möglichkeiten der Preisbeeinflussung durch Faktoren oder auch Formeln und die zugehörige Rechenart, mit der ein solcher Faktor Einfluss auf den Preis nimmt, mit abgebildet werden können.
Durch die unterschiedlichen Variationen eines Produktes mit verschiedenen Merkmalsausprägungen ergeben sich oftmals auch unterschiedliche Artikelnummern, die von den jeweiligen Merkmalsausprägungen abhängig sind. Auch für die Darstellung solcher zusammengesetzter Artikelnummern, wie sie auch der Beispiel-Artikel im Katalogausschnitt auf Seite 48 zeigt, sollen Möglichkeiten in der Datenstruktur vorgesehen werden. Wichtig ist dabei, dass eine Reihenfolge der Variationsmöglichkeiten festgelegt werden kann. Ebenfalls erwünscht ist die Angabe eines Separator-Zeichens, das zwischen der bisherigen Artikelnummer und dem durch die Auswahl eines Merkmales zugehörigen Auswahlcodes eingefügt werden kann.



Taster MTP ¾"
Switches in momentary action MTP ¾"

Ausführungen / Models		Varianten / Variations			Artikelnummer / Part Number
MTP ¾"					
Form 1 Shape 1	IP 40	Kontaktmaterial / Contact material	Ag	unbeleuchtet / non- illuminated	1241.1153.X.XXX
			Au		1241.1163.X.XXX
	IP 54		Ag		1241.1154.X.XXX
			Au		1241.1164.X.XXX
Form 1 Shape 1	IP 40		Ag	beleuchtet / illuminated	1241.1150.X.XXX.X
			Au		1241.1160.X.XXX.X
	IP 54		Ag		1241.1151.X.XXX.X
			Au		1241.1161.X.XXX.X
Form 2 Shape 2	IP 40	Kontaktmaterial / Contact material	Ag	unbeleuchtet / non- illuminated	1241.1173.X.XXX
			Au		1241.1183.X.XXX
	IP 54		Ag		1241.1174.X.XXX
			Au		1241.1184.X.XXX
Form 2 Shape 2	IP 40		Ag	beleuchtet / illuminated	1241.1170.X.XXX.X
			Au		1241.1180.X.XXX.X
	IP 54		Ag		1241.1171.X.XXX.X
			Au		1241.1181.X.XXX.X
	IP 40		Ag	mit LED-Bohrung (ohne LED) / with hole for LED (without LED)	1241.1190.X.XXX
			Au		1241.1193.X.XXX
	IP 54		Ag		1241.1191.X.XXX
			Au		1241.1194.X.XXX

Form 1 Shape 1	Farbe der Tastkappe / Colour of key cap	rot red	3
Einlegeschild Insert plate		grün green	5
		grau grey	6
		schwarz black	7
		weiß white	8

Tastkappe mit Grundmodul Key cap with base module	Beschriftung Tastkappe/Einlegeschild (Schrifthöhe/Schriftarten siehe Seite 39) / Legend of key cap insert plate (Type height/type face see page 39)	A = 001	P = 016	4 = 031	↕ = 046	EIN = 061
		B = 002	Q = 017	5 = 032	→ = 047	AUS = 062
		C = 003	R = 018	6 = 033	← = 048	AUF = 063
		D = 004	S = 019	7 = 034	↓ = 049	AB = 064
		E = 005	T = 020	8 = 035	↑ = 050	ON = 065
		F = 006	U = 021	9 = 036	% = 051	OFF = 066
		G = 007	V = 022	+ = 037	√ = 052	UP = 067
		H = 008	W = 023	- = 038	CTRL = 053	DOWN = 068
		I = 009	X = 024	· = 039	RETURN = 054	HIGH = 069
		J = 010	Y = 025	x = 040	SHIFT = 055	LOW = 070
		K = 011	Z = 026	÷ = 041	LOCK = 056	ON/OFF = 071
		L = 012	0 = 027	✳ = 042	STOP = 057	UP = 072
		M = 013	1 = 028	= = 043	ENTER = 058	
		N = 014	2 = 029	# = 044	BACK = 059	
Form 2 Shape 2		O = 015	3 = 030	↔ = 045	LINE = 060	

Tastkappe mit Grundmodul Key cap with base module	Farbe des Einlegeschildes bei unbeschrifteter Ausführung / Colour of insert plate without legend	gelb yellow	= 091	grau grey	= 096
		orange orange	= 092	schwarz black	= 097
		rot red	= 093	weiß white	= 098
		blau blue	= 094	anthrazit anthracite	= 099
		grün green	= 095		

Farbe der LED / Colour of LED	rot red	1
	grün green	2
	gelb yellow	3

Abbildung 3.2: Beispiel eines Artikels mit verschiedenen Variationen aus einem gedruckten Katalog

Die Beachtung von Preisveränderungen bei Produktvariationen und auch die Generierung von variationsabhängigen Artikelnummern fehlt bis dato in allen betrachteten Katalogstandards und wird erst mit der Veröffentlichung von BMEcat 2.0 in Aussicht gestellt. Mit dieser Anforderung soll aber bereits eine erste Möglichkeit aufgezeigt werden, wie Produktvarianten und -konfigurationen inklusive ihrer Preisbeeinflussung und veränderliche Artikelnummern in einem XML-Datenformat dargestellt werden können.

3.2.4 Anforderungen durch SAP

Durch die Evaluation der in ERP-Systemen verwendetn Techniken zur Bildung von Preisinformationen angeregt, bestand Anfangs der Wunsch, die Daten in die XML-Struktur mit aufzunehmen, die zur Berechnung eines Preises verwendet werden. Dies sollte primär zu einer besseren Transparenz eines Preises dienen und sekundär auch eine erneute Berechnung eines Preises in einem anderen System ermöglichen.
Um die im Hintergrund des ERP-Systemes laufenden Preisberechnungsprozesse besser verstehen zu können, wurde von Seiten der Firma viaMEDICI Software GmbH ein SAP-Consultant hinzugezogen. In einem Gespräch mit Demonstration an einem SAP-System wurde aber sehr schnell klar, dass besonders SAP intern viel zu viele Konfigurationsmöglichkeiten zur Berechnung von Preisen bietet, als dass diese in einem überschaubaren Rahmen in einer XML-Struktur hätten dargestellt werden können. Erschwerend kommt hier noch hinzu, dass eine Vielzahl dieser Konfigurationen von jedem SAP-Kunden individuell vorgenommen werden können. Die in dieser Arbeit abgebildeten Tabellen (Seite 19 und 111) vermitteln einen kleinen Einblick in die komplexen Möglichkeiten der Preisberechnung in SAP-Systemen.
So blieb von Seiten eines ERP-Systemes nur noch die Anforderung übrig, den Typ des ERP-Systemes und die innerhalb des ERP-Systemes verwendete Artikelnummer in der zu erstellenden Datenstruktur abbilden zu können.

3.3 Design des XML Schema

Nachdem im vorherigen Unterkapitel die Anforderungen an die XML-Datenstruktur beschrieben wurden, geht es im folgenden nun darum, das Design oder besser ausgedrückt, den Aufbau des XML-Datenmodells aufzuzeigen und zu beschreiben. Dabei soll auch auf Daten- und Elementtypen und auf die Gründe für die einzelnen Entscheidungen eingegangen werden.
Zu Beginn ist gleich noch zu vermerken, dass sich das erstellte Datenmodell sehr eng an BMEcat 1.2 als primäres Zielformat orientiert. Hier stellt sich natürlich dem aufmerksamen Betrachter die Frage, warum an dieser Stelle nicht gleich ein BMEcat-Dokument als Container für die Preisinformationen gewählt wur-

de. Die Entscheidung fiel deshalb für ein eigenes, aber an BMEcat orientiertes XML-Datenformat, weil BMEcat neben den Preisinformationen noch eine Menge Informationen mehr zu Artikeln enthalten kann und die Verwendung des gesamten Formates an dieser Stelle etwas zu umfangreich gewesen wäre. Darüber hinaus sollte auch einige in der Praxis mit Katalogstandards entstandene Anforderungen, wie z.B. die Abbildung von Preisberechnungsformeln und Produktvarianten, in die Datenstruktur aufgenommen werden, was uns eine direkte Übernahme des BMEcat-Standards als Datenstruktur nicht ermöglicht hätte. Zum Schluss muss auch noch erwähnt werden, dass die Datenstruktur auch einige wenige Informationen aus dem ERP-System aufnehmen soll. Auch hier wären mit der direkten Übernahme des BMEcat-Standards zu viele Einschränkungen vorhanden gewesen.
Um die Übersichtlichkeit zu verbessern, wurde das XML Schema ebenfalls, ähnlich dem BMEcat-Schema, in mehrere Dateien aufgeteilt. Die Hauptdatei heißt priceinfo.xsd (siehe Listing Seite 133) und enthält die Grundstruktur des gesamten XML-Datenmodells. In der Datei priceinfo_base.xsd (siehe Listing Seite 113) sind dann die einzelnen verwendeten Datentypen, Elementtypen und Elemente selbst beschrieben. Zusätzlich zu diesen beiden Dateien gibt es noch insgesamt zwei weitere Dateien namens countries.xsd und currencies.xsd, die die Aufzählungsdatentypen, die nachfolgend näher beschrieben werden, definieren. Diese Aufzählungsdatentypen wurden aus Gründen der Kompatibilität direkt aus dem BMEcat-Stabdard übernommen. Aus Platzgründen und weil sie nur Aufzählungen von standardisierten Abkürzungen enthalten, wurden diese zwei Dateien nicht als Listing an diese Arbeit angehängt.

3.3.1 Allgemeine Datentypen

In BMEcat wurden auf Grundlage der nach XML Schema Spezifikation verfügbaren Datentypen eigene Datentypen definiert, die aus Gründen der Kompatibilität auch weitestgehend in die erstellte XML-Datenstruktur übernommen wurden. Neben mittels Pattern festgelegten Datentypen wie z.B. dtBOOLEAN, dtTIMETYPE wurden auch einige Datentypen durch Aufzählungen von gültigen Werten definiert. Insbesondere auf die beiden Datentypen, die mittels Aufzählungen definiert wurden, soll im Folgenden näher eingegangen werden.

Datentyp dtCOUNTRIES

Im Datentyp dtCOUNTRIES werden über eine Aufzählung mögliche Abkürzungen für Länder und Regionen nach den ISO-Standards ISO 3166-1:1997, ISO 3166-2:1998 und ISO 3166-3:1999 definiert. Dieser Datentyp wurde direkt aus der BMEcat-Spezifikation ([BMEcat1]) entnommen und der Inhalt darf maximal sechs Zeichen umfassen.

Datentyp dtCURRENCIES

Im Datentyp dtCURRENCIES werden über eine Aufzählung mögliche Abkürzungen für Währungen nach dem ISO-Standard 4217:1995 definiert. Dieser Datentyp wurde direkt aus der BMEcat-Spezifikation ([BMEcat1]) entnommen und der Inhalt darf maximal drei Zeichen umfassen.

3.3.2 Allgemeine Elementtypen

Für Datums-/Zeitangaben, Adressen, Preisberechnungsformeln und vieles mehr wurden im erstellten XML Schema jeweils Elementtypen deklariert, die so trotz nur einmaliger Beschreibung eine mehrfache Verwendung innerhalb der Datenstruktur ermöglichen. Ein Großteil dieser Datentypen wurde aus Gründen der Kompatibilität direkt aus dem BMEcat-Standard übernommen. Neben einigen einfachen Elementtypen, die hier nicht näher beschrieben werden sollen, wurden auch drei komplexe Elementtypen spezifiziert, die im Folgenden nun näher beschrieben werden sollen.

Elementtyp typeDATETIME

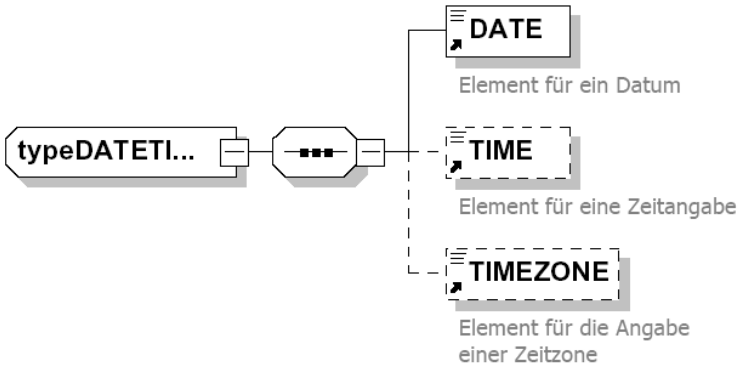


Abbildung 3.3: Aufbau des Elementtyps typeDATETIME

Der komplexe Elementtyp DATETIME dient zur Darstellung von Datums- und Zeitinformationen und ist in drei Unterelemente gegliedert.
Das erste Unterelement ist das Element DATE vom Datentyp dtDATETYPE, was dem Schema-Datentyp xsd:date („yyyy-mm-dd“) entspricht . Es muss innerhalb eines DATETIME-Elementes einmal vorhanden sein und nimmt das eigentliche Datum auf.
Ergänzt kann dieses Element noch durch die Elemente TIME und TIMEZONE werden. Das Element TIME entspricht dem aus der BMEcat-Spezifikation entnommenen Datentyp dtTIMETYPE, der über einen regulären Ausdruck definiert

wird und Uhrzeiten im Format „hh:mm:ss.sss“ aufnehmen kann. Das Element TIMEZONE entspricht dem ebenfalls der BMEcat-Spezifikation entnommenen Datentyp dtTIMEZONETYPE und kann Angaben zu Zeitzonen im Muster „[+—-]hh:mm“ oder den Buchstaben „Z“ fur die koordinierte Universalzeit (UTC) aufnehmen.
Dieser Elementtyp findet seinen Einsatz in den Elementen HEADER (Erstellungsdatum, Gültigkeit des Dokumentes und Gültigkeit der Vertragsinformationen) und PRICEDATA (Gültigkeit der Preisinformationen).
Obwohl in der Spezifikation von XML Schema ([XMLSchema1]) ein Datentyp *dateTime* definiert ist, wurde aus Gründen der einfacheren Kompatibilität zum primären Zielformat BMEcat das zuvor beschriebene Format für die Beschreibung von Datums- und Zeitinformationen gewählt. Darüberhinaus gewährt diese Art der Darstellung von Zeit- und Datumsinformationen ein Zusammenfügen der hier getrennt dargestellten Informationen zu einem den XML Schema-Typen entsprechenden String und somit eine höhere Flexibilität auch im Hinblick auf andere Katalogstandards.

Elementtyp typeADDRESS

Der Elementtyp ADDRESS wurde zur Aufnahme von Adressinformationen definiert. Da die einzelnen enthaltenen Elemente innerhalb des Elementtyps und die Reihenfolge aus Gründen der Kompatipilität weitgehendst dem ADDRESS-Element des BMEcat-Standard entsprechen, wird an dieser Stelle auf eine detaillierte Beschreibung der einzelnen Elemente verzichtet und auf die Spezifikation von BMEcat ([BMEcat1]) verwiesen. Die folgende Abbildung auf Seite 53 soll aber einen Überblick über den Aufbau des Elementtyps ermöglichen.
Der ADDRESS-Elementtyp findet im Element HEADER Verwendung in den Elementen zur Beschreibung der genaueren Adressinformationen der verkaufenden bzw. liefernden und der einkaufenden Partei.

Elementtyp typeFORMULA

Zur Abbildung von Formeln, insbesondere von Preisberechnungsformeln, wurde der Elementyp typeFORMULA geschaffen, der so bisher noch keine Beachtung im BMEcat-Standard findet. Er ist unterteilt in zwei Kindelemente.
Das erste Kindelement heißt FORMULASTRING und muss innerhalb eines Elementes vom Typ typeFORMULA genau einmal vorhanden sein. Damit diese Formel, die im typeFORMULA-Kindelement FORMULASTRING als String mit maximal 255 Zeichen abgelegt werden muss, auch interpretierbar ist, wurde dem Elementtyp noch ein Kindelement namens FORMULADESCRIPTION hinzugefügt.
Dieses Element erlaubt eine Beschreibung der Formel mit einem String von maxi-

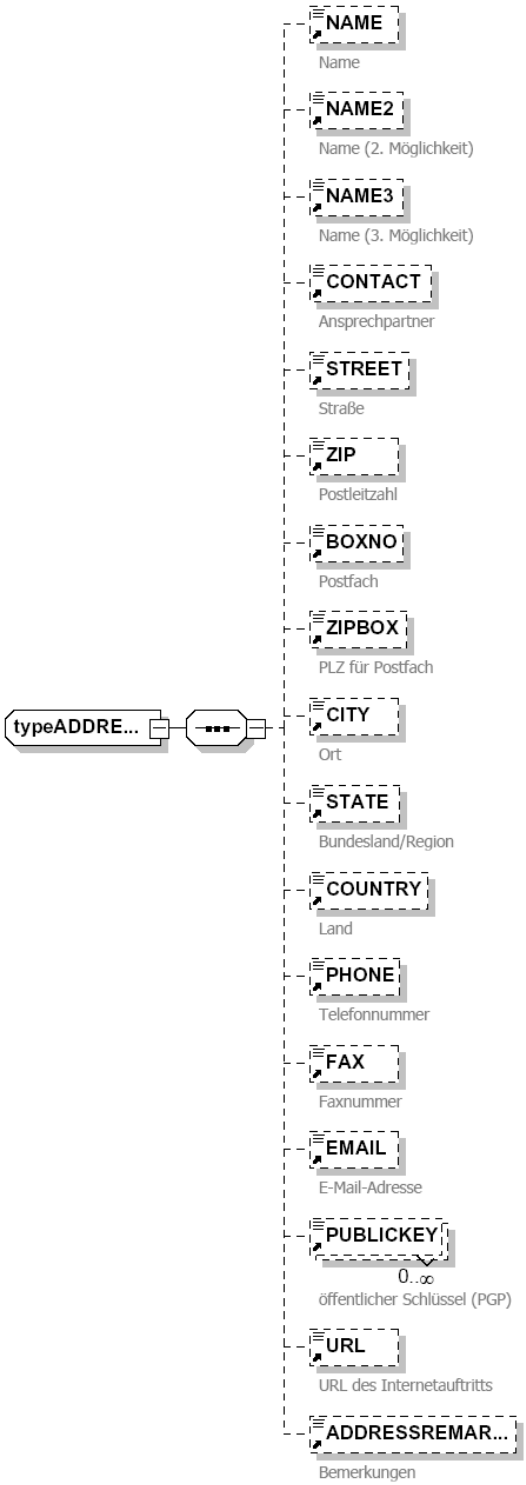


Abbildung 3.4: Aufbau des Elementtyps typeADDRESS

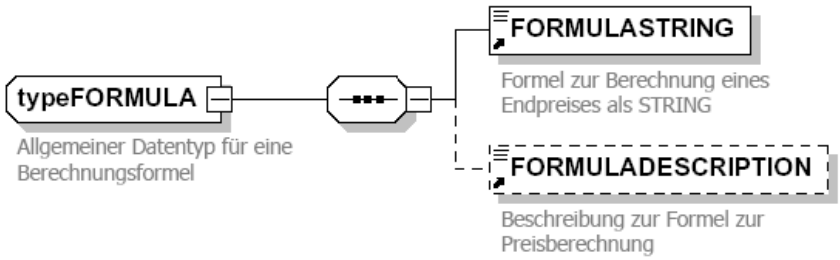


Abbildung 3.5: Aufbau des Elementtyps typeFORMULA

mal 64.000 Zeichen. Hierfür wurde aber noch keine weitere Spezifikation erstellt, weil bisher nicht in Erfahrung gebracht werden konnte, wie Formeln in der angekündigten Version 2.0 des BMEcat-Standards tatsächlich dargestellt und letztendlich auch interpretiert werden sollen und alle anderen untersuchten Katalogstandards derzeit keine Möglichkeit der Abbildung einer Preisberechnungsformel ermöglichen.

Elementtyp typeUNIT

Im Elementtyp typeUNIT können Abkürzungen der verschiedenen Verpackungseinheiten (z.B. Stück, Meter, etc.) als beliebig langer String abgelegt werden. Damit festgelegt werden kann, um was für ein Kürzel es sich handelt, wurde zur Ergänzung ein Attribut namens *type* testgelegt, das vorhanden sein muss und die folgenden drei Werte annehmen kann:

Wert	**Bemerkung**
erp	Verpackungseinheit aus dem ERP-System (oftmals landestypische Abkürzung)
iso	Verpackungseinheit, abgekürzt nach ISO-Norm (wird von den meisten ERP-Systemen so geliefert)
unece	Verpackungseinheit, abgekürzt nach UN/ECE Empfehlung Nr. 20 (wird so z.B. in BMEcat verwendet)

Tabelle 3.1: PRICEINFO: Werte des typeUNIT-Attributes *type*

Mit diesem Elementtyp wird der Tatsache Rechnung getragen, dass bei den vielen, auf dem Weg zum Katalogstandard befindlichen unterschiedlichen Systemen auch unterschiedliche Abkürzungen für Verpackungseinheiten verwendet werden können und so bei der Transformation der Daten in den Katalogstandard die passende Kurzform der Verpackungseinheit ausgewählt werden kann.
Dieser Elementtyp findet ausschließlich im ARTICLEDATA-Unterelement ORDERINFOS (siehe Seite 63) Verwendung.

3.3.3 Allgemeiner Aufbau

Das Root-Element der erstellten XML-Struktur trägt den Namen PRICEINFO und enthält insgesamt vier Unterelemente. Diese vier Unterelemente gliedern das XML-Dokument in die vier Bereiche Dokumentenkopf, Artikeldaten, Preisdaten und Varianten.

Das Unterelement HEADER nimmt dabei die Informationen auf, die in der

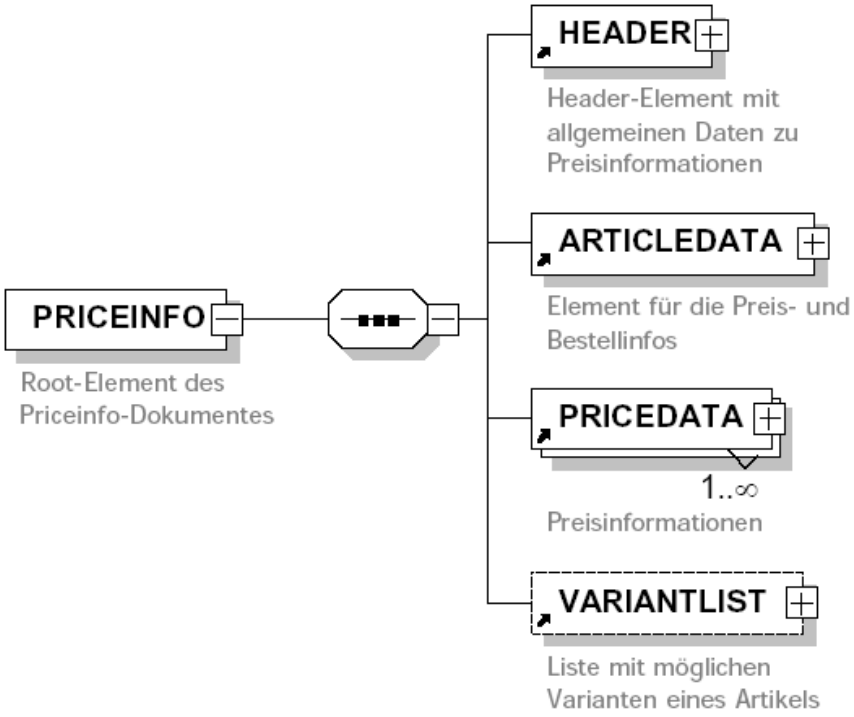


Abbildung 3.6: Aufbau des Wurzelelementes PRICEINFO mit den vier Unterelementen

Regel in einem Dokumentenkopf abgelegt werden. Es muss genau ein HEADER-Element in einem XML-Dokument nach dem erstellten XML Schema vorhanden sein. Dazu gehören allgemeine Infos zum Dokument selbst und eine ganze Reihe an Default-Werten. Eine detaillierte Beschreibung dieses Elementes beginnt auf Seite 56.

Im Unterelement ARTICLEDATA können Informationen zum jeweils in dem XML-Dokument behandelten Artikel gespeichert werden, die nicht direkt den Preis beeinflussen. Dieses Element muss genau einmal im XML-Dokument vorhanden sein. Es wird ab Seite 60 näher beschrieben.

Die variantenunabhängige Basis aller Preisinformationen kann im Unterelement PRICEDATA abgelegt werden. Es muss mindestens einmal, kann aber auch mehrmals im XML-Dokument vorkommen. Die darin enthaltenen Elemente werden ab

Seite 64 detailliert beschrieben.
Das letzte, optionale Element im Element HEADER heißt VARIANTLIST und kann unterschiedliche Varianten eines darzustellenden Produktes beschreiben. Der Aufbau dieses Elementes wird ab Seite 68 näher beschrieben. Ein PRICEINFO-XML-Dokument kann jeweils nur die Preisinformationen zu einer Artikel-Kunden-Beziehung in genau einer Sprache darstellen. Damit haben Informationen zum Artikel, zum einkaufenden Unternehmen (als Kunde) und zur verwendeten Sprache eine Art Schlüsselfunktion. Begründet ist dies in den unterschiedlichen Sichtweisen von viaMEDICI EPIM, das Informationen artikelbezogen verwaltet, und dem primären Zielstandard BMEcat, das kundenbezogene und sprachabhängige Kataloge erzeugt.

3.3.4 Das Element HEADER

Das Element HEADER enthält selbst wieder bis zu 12 Unterelemente, die grob zusammengefasst das XML-Dokument und die beteiligten Parteien identifizieren.
Das erste Unterelement heißt DOCID und muss eine eindeutige Identifikationsbezeichnung aufnehmen, die das Dokument bei seiner Erstellung erhält und es somit identifizieren kann. Der String, der von diesem element aufgenommen werden kann, darf maximal 50 Zeichen lang sein und entspricht damit nicht der als Pendant aus dem BMEcat-Standard zu bezeichnenden Katalog-ID. Geeignet als ID ist z.B. eine Kombination aus Artikelnummer, Kundenummer und Sprachkürzel.
Im zweiten Unterelement namens DOCTITLE muss dem XML-Dokument ein maximal 255 Zeichen langer Titel-String gegeben werden, der z.B. dem Namen des beschriebenen Artikels entsprechen kann.
Das Erstellungsdatum des Dokumentes muss im Unterelement DOCDATETIME dokumentiert werden. DOCDATETIME ist ein Element des Elementtyps typeDATETIME (Seite 51) und entspricht somit in seinem Aufbau dem Pendant der BMEcat-Spezifikation.
Im Unterelement DOCVALIDITY muss die Gültigkeit bzw. genauer mindestens der Startzeitpunkt der Gültigkeit des Dokumentes beschrieben sein. Das Unterelement selbst enthält mindestens ein, maximal aber zwei Elemente namens DATETIME, die vom Elementtyp typeDATETIME sind und über ein Attribut namens Type unterschieden werden müssen. Dabei steht der Attributswert *validity_start_date* für das Anfangsdatum und der Attributswert *validity_end_date* für das Endedatum des Gültigkeitszeitraumes. Die BMEcat-Spezifikation kennt in dieser Form keine Gültigkeitsangaben für das gesamte Dokument, sondern lediglich Gültigkeiten von Verträgen und Preisen.
Im Unterelement SOURCESYSTEM kann festgelegt werden, welches Softwaresystem die Quelle der innerhalb des Dokumentes gespeicherten Informationen ist. Es kann dabei einen beliebigen String mit einer maximalen Länge von 50 Zeichen aufnehmen. Es ist das Resultat aus der Anforderung, das Quellsystem im XML-

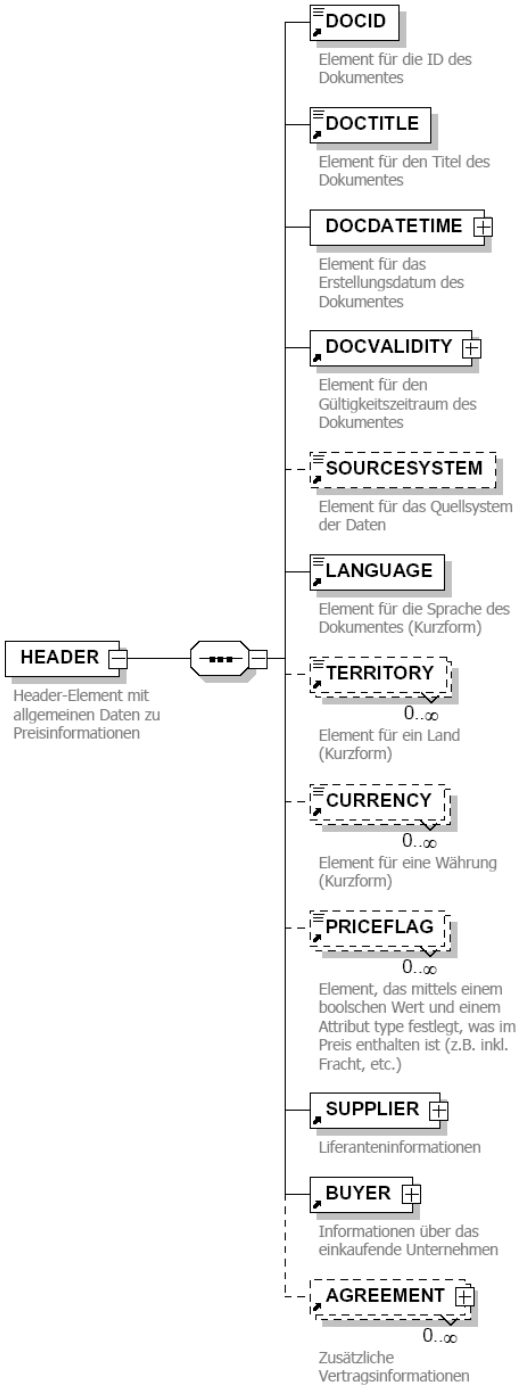


Abbildung 3.7: Aufbau des Elementes HEADER

Datenkontainer abbilden zu können.
Da in dem XML-Dokument auch sprachabhängige Informationen abgelegt werden können, muss im Unterelement LANGUAGE die Sprache, in der diese Informationen vorliegen, festgelegt werden. Das Element LANGUAGE nimmt im Gegensatz zum Gegenstück im BMEcat-Standard Strings mit einer maximalen Länge von 10 Zeichen auf, da es viele unterschiedliche Kurzformen für Sprachen gibt und an dieser Stelle der Wert aus dem ERP-System bzw. des Software-Systems viaMEDICI EPIM wichtig ist und somit die Flexibilität deutlich erhöht wird. Natürlich impliziert dieser offene Datentyp die Notwendigkeit eines Wertemappings, das aber beim Export der Daten in Richtung eMärkte (siehe Seite 90) vorgenommen werden kann.
Wie bereits im einführenden Kapitel (siehe Seite 6) beschrieben, haben Preisinformationen oftmals nur eine beschränkte geographische Gültigkeit. Diese geographische Gültigkeit kann im Unterelement TERRITORY, dass mehrmals vorhanden sein kann, festgelegt werden. Hierbei handelt es sich um einen Default-Wert, der dann greift, wenn direkt bei den Preisinformationen (siehe Seite 64) keine weiteren Informationen zur geographischen Gültigkeit hinterlegt sind. Das Element selbst nimmt Daten vom Datentyp dtCOUNTRIES (siehe Seite 50) auf und entspricht somit exakt seinem BMEcat-Gegenstück.
Ebenfalls an dieser Stelle eine Art Default-Wert ist das Element CURRENCY vom Datentyp dtCURRENCIES (siehe Seite 51). Es nimmt nach ISO normierte Kürzel für Währungen auf und kann einmal vorhanden sein. Dieses Element entspricht aus Gründen der Kompatibilität und nicht vorhandener Abweichungen in den vorgelagerten Systemen exakt seinem Pendant CURRENCY im CATALOG-Element des BMEcat-HEADERS.
Das Unterelement PRICEFLAG vom Datentyp dtBOOLEAN zeigt mittels einem Attribut *type* an, was in dem genannten Preis alles enthalten ist. Das Attribut *type* kann die in der folgenden Tabelle aufgelisteten Werte annehmen, die wie das gesamte Element dem BMEcat-Standard entnommen sind.

Wert	**Bemerkung**
incl_freight	Preis enthält Frachtkosten
incl_duty	Preis enthält Verzollung
incl_packing	Preis enthält Verpackungskosten
incl_assurance	Preis enthält Versicherung

Tabelle 3.2: PRICEINFO: Werte des PRICEFLAG-Attributes *type*

Hat das Element den Wert TRUE, ist die entsprechende Leistung im angegebenen Preis enthalten. Ist das Element mit dem entsprechenden Attributwert nicht vorhanden oder hat das Element den Wert FALSE, ist diese Leistung nicht im angegebenen Preis enthalten.
Die beiden Unterelemente SUPPLIER und BUYER beinhalten Informationen

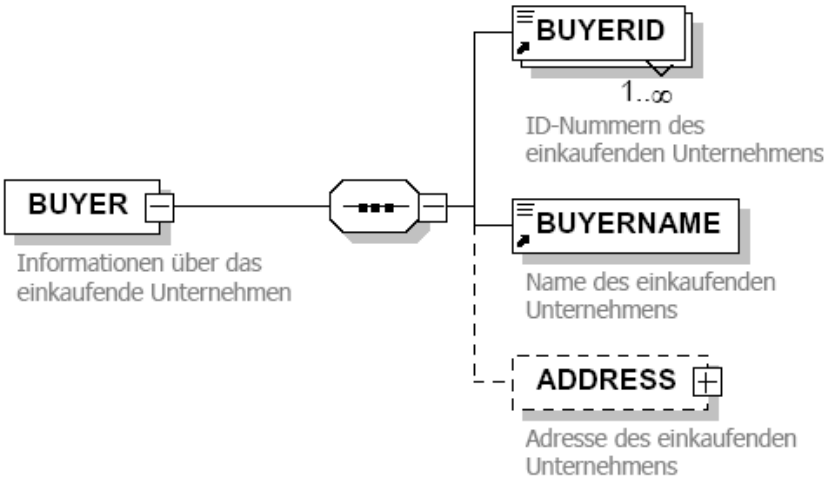


Abbildung 3.8: Aufbau des Elementes BUYER

über die beiden am Austausch der Daten beteiligten Parteien. Im Element SUPPLIER, das weiter eine oder mehrere SUPPLIERIDs, einen SUPPLIERNAME und ein ADDRESS-Element (siehe Seite 52) mit dem Attribut *type* = „supplier“ enthält, werden Informationen zum anbietenden Unternehmen gespeichert. Im Element BUYER, das ähnlich dem SUPPLIER-Element eine oder mehrere BUYERIDs, einen BUYERNAME (optional) und ebenfalls ein ADDRESS-Element allerdings mit Attribut *type* = „buyer“ enthält, werden die Informationen des einkaufenden Unternehmens hinterlegt. Wie bereits erwähnt, haben die Elemente SUPPLIERID bzw. BUYERID, die beide mehrfach vorkommen können, noch ein optionales, die verschiedenen Identifikationsnummern unterscheidendes Attribut namens *type*, das die in der folgenden Tabelle aufgelisteten Werte annehmen kann:

Wert	**Bemerkung**
duns	DUNS-Kennung
iln	ILN-Kennung
buyer_specific	spezifische Kennung des einkaufenden Unternehmens
supplier_specific	spezifische Kennung des verkaufenden Unternehmens

Tabelle 3.3: PRICEINFO: Werte des SUPPLIERID- bzw. BUYERID-Attributes *type*

Die beiden Elemente SUPPLIER und BUYER wurden in Aufbau und Struktur weitgehendst der BMEcat-Spezifikation entliehen, um die Kompatibilität zu diesem zu wahren. Da das Element BUYERID innerhalb des Dokumentes und auch beim Export in den BMEcat-Standard eine Schlüsselposition einnimmt, muss im Gegensatz zum BMEcat-Standard mindestens jeweils eine ID-Nummer angegeben werden, weil diese IDs in der Praxis und insbesondere die BUYERID in

diesem Dokument eine Schlüsselrolle einnehmen.
Das letzte Unterelement AGREEMENT enthält Informationen zu weiteren Vereinbarungen zwischen den beiden Parteien, wie z.B. zusätzliche Verträge, etc.. Es kann mehrfach vorhanden sein und somit auf mehrere Verträge verweisen. Das Element AGREEMENT ist in zwei Unterelemente aufgeteilt.
Das Element AGREMENTID muss eine eindeutigen Identifikationsstring enthalten, der die zugehörige Vereinbarung eindeutig identifiziert. Das Element DATETIME muss einmal und kann maximal zweimal vorhanden sein. Es ist wie bereits auch die anderen DATETIME-Elemente vom Datentyp typeDATETIME (siehe Seite 51) und um ein Attribut namens *type* ergänzt und gibt den Gültigkeitszeitraum einer Vereinbarung an. Das Attribut *type* muss, wenn das DATETIME-Element das Startdatum der Gültigkeit der Vereinbarung angibt, den Wert *agreement_start_date* haben. Gibt das DATETIME-Element das Endedatum des Gültigkeitszeitraumes der Vereinbarung an, muss das Attribut *type* den Wert *agreement_end_date* haben. Das Element inklusive aller Kindelemente wurde exakt aus der BMEcat-Spezifikation übernommen.

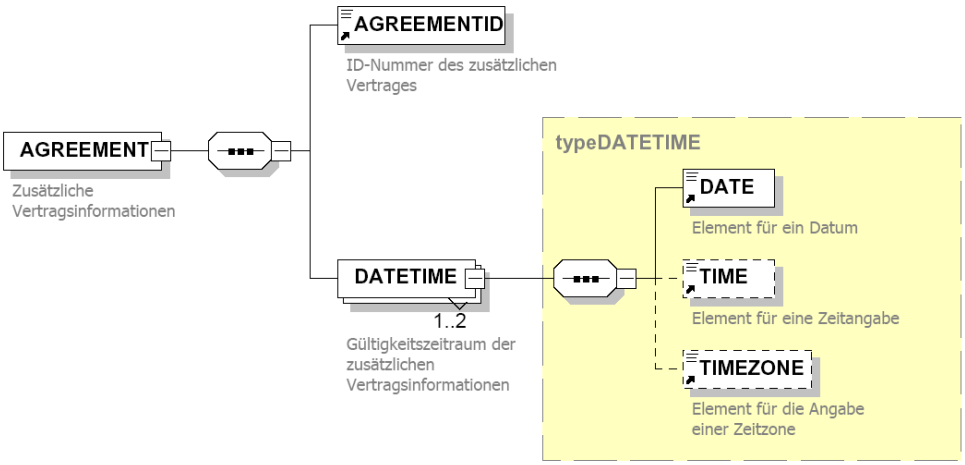


Abbildung 3.9: Aufbau des Elementes AGREEMENT

3.3.5 Das Element ARTICLEDATA

Im Element ARTICLEDATA werden Informationen zum Artikel selbst abgespeichert. Diese Informationen reichen vom Namen des Artikels bis zu Bestellinformationen. Abgesehen von den Bestellinformationen, die bisher noch nicht in viaMEDICI EPIM vorhanden waren, sind die Daten dieses Elementes bereits vorhanden. Dies hat seinen Grund darin, dass ein PRICEINFO-XML-Dokument ohne einen weiteren Zugriff auf die Daten in viaMEDICI EPIM von Menschen lesbar sein soll. Trotzdem kann hier nicht von einer Datenredundanz gesprochen

werden, weil die technischen Informationen eines Artikels, die innerhalb des Systemes viaMEDICI EPIM vorliegen, dort von den Usern auch geändert werden können. In dem erstellten Datenmodell können die aus dem ERP-System erhaltenen Daten aber nicht verändert werden und entsprechen somit original den Daten aus dem ERP-System.
Das Element selbst ist weiter in sechs Unterelemente unterteilt, die im Folgenden

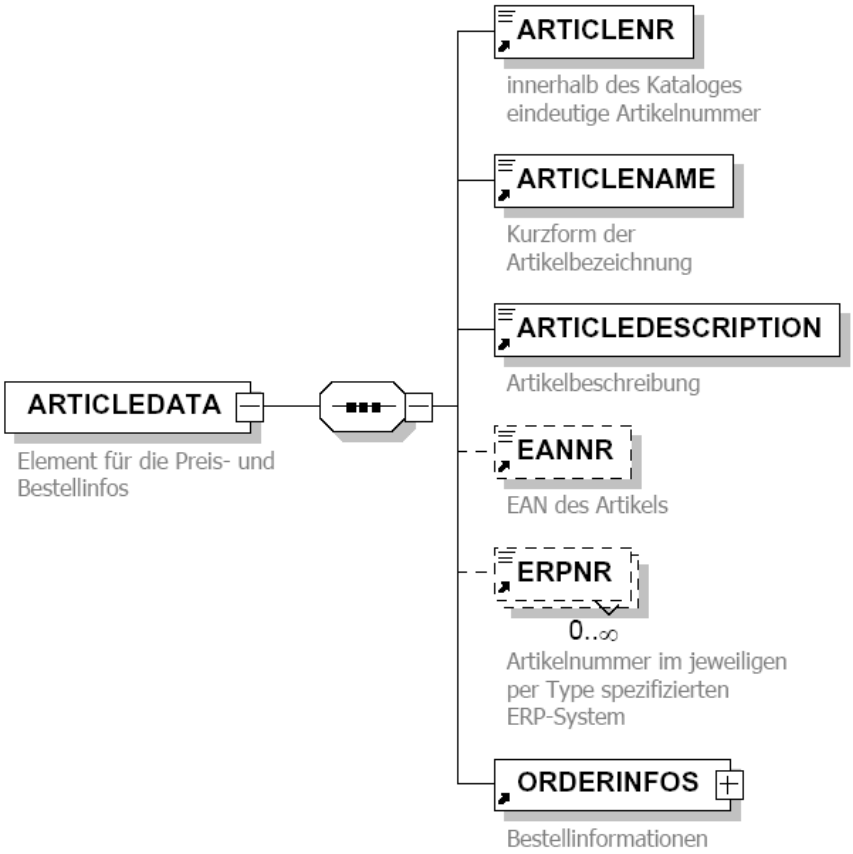


Abbildung 3.10: Aufbau des Elementes ARTICLEDATA

näher beschrieben werden.
Das erste Unterelement heißt ARTICLENR und kann eine eindeutige Artikelnummer in Form eines maximal 32 Zeichen langen Strings aufnehmen. Diese eindeutige Artikelnummer entspricht dem dritten benötigten Schlüsselwert, den ein PRICEINFO-XML-Dokument für die eindeutige Zuordnung innerhalb des Softwaresystems viaMEDICI EPIM und auch für den Export in Richtung eMarktplätze benötigt. Es ist damit auch verständlich, dass dieses Unterelement genau einmal vorhanden sein muss. Das Element entspricht damit, abgesehen vom Namen und der Anordnung innerhalb derDatenstruktur, seinem Pendant SUP-

PLIER_AID in der BMEcat-Spezifikation.
Das zweite Unterelement, das ebenfalls innerhalb eines ARTICLEDATA-Elementes vorhanden sein muss, ist ARTICLENAME. Es kann mittels einem 80 Zeichen langen String einen von Menschen lesbaren Namen des Artikels aufnehmen und entspricht damit dem Element DESCRIPTION_SHORT im Element ARTICLE_DETAILS der BMEcat-Spezifikation.
Um den Artikel näher beschreiben zu können, kann das Element ARTICLE-DESCRIPTION mit einer maximal 64.000 Zeichen langen Beschreibung gefüllt werden. Auch dieses Element dient ausschließlich der besseren menschlichen Lesbarkeit. Sein Gegenstück ist im BMEcat-Standard unter dem Namen DESCRIPTION_LONG im Element ARTICLE_DETAILS zu finden.
Da die Artikelnummer oft nicht der international eindeutigen EAN-Nummer entspricht, kann eine EAN-Nummer, sofern vorhanden bzw. bekannt, im Element EANNR gespeichert werden. Dieses Element kann einen String mit einer maximalen Länge von 14 Zeichen aufnehmen. Auch hierfür gibt es mit dem Element EAN im BMEcat-Standard ein Pendant.
Damit auch ein einfaches Nachschlagen in der entsprechenden Datenquelle möglich ist, können mit dem Unterelement ERPNR auch die ERP-System spezifischen Artikelnummern abgelegt werden. Da eine Firma im schlimmsten Fall mit mehreren ERP-Systemen arbeiten kann, ist es hier auch möglich, mehrere dieser Elemente aufzuführen, die jeweils einen bis zu 50 Zeichen langen String aufnehmen können. Zur Unterscheidung der verschiedenen Quellen, muss das Attribut *type* mit einem bis zu 50 Zeichen langen String gefüllt werden, der das Quellsystem oder auch eine andere Quelle (z.B. firmenspezifisch) beschreibt und so den Kontext der Nummer festlegt.
Das letzte Unterelement heißt ORDERINFOS und muss genau einmal innerhalb eines ARTICLEDATA-Elementes vorhanden sein. ORDERINFOS dient der Aufnahme der Bestellinformationen zum jeweiligen Artikel und ist weiter nochmals in sechs Unterelemente unterteilt. Die gesamte Struktur entspricht mit Ausnahme der beiden Einheiten-Elemente ORDERUNIT und CONTENTUNIT dem Aufbau des Elementes ARTICLE_ORDER_DETAILS im ARTICLE-Element der BMEcat-Spezifikation.
Das erste Unterelement von ORDERINFO heißt ORDERUNIT und muss mindestens einmal und kann maximal drei mal vorhanden sein. Es ist vom Elementtyp typeUNIT (siehe Seite 54) und kann somit Packungseinheiten in Form eines Strings aufnehmen, der für das Element ORDERUNIT auf maximal 10 Zeichen beschränkt ist. In diesem Fall wird konkret die Einheit angegeben, in der der entsprechende Artikel auch bestellt werden kann.
Das zweite Unterelement ist ebenfalls vom Elementtyp typeUNIT und heißt CONTENTUNIT. Dieses Element ist aber im Vergleich zum vorherigen Element optional, kann aber ebenfalls bis zu drei Mal vorhanden sein. Es kann mittels einem maximal 10 Zeichen langen String die Packungseinheit der innerhalb einer Bestelleinheit befindlichen (Inhalts-)Einheiten aufnehmen.

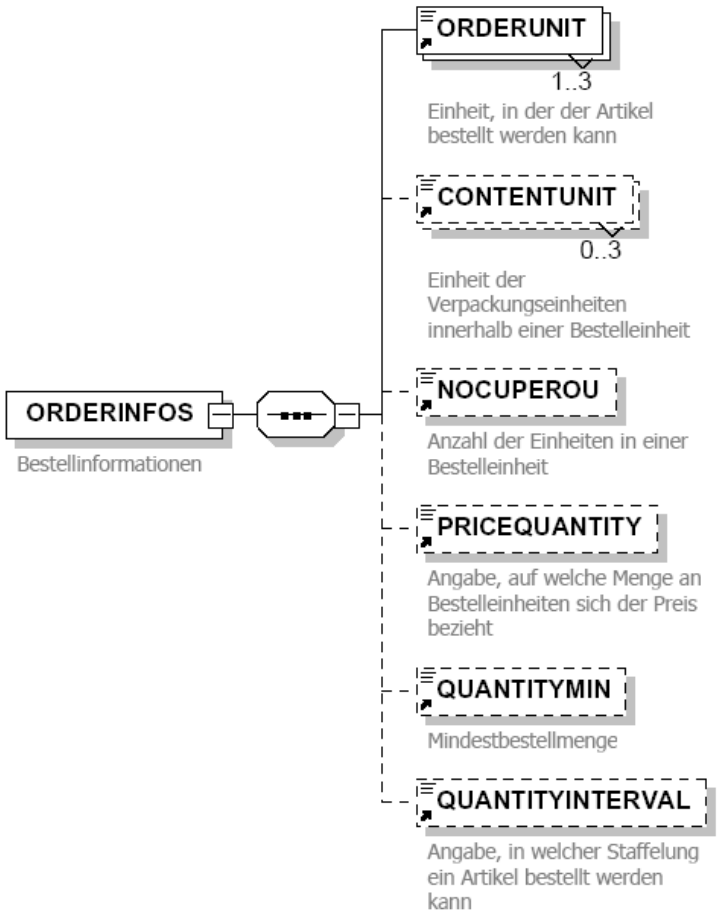


Abbildung 3.11: Aufbau des Elementes ORDERINFOS

Wie deutlich sichtbar ist, unterschieden sich diese beiden Elemente von ihren Gegenstücken in der BMEcat-Spezifikation, weil BMEcat an dieser Stelle nur mit Abkürzungen von Verpackungseinheiten nach UN/ECE Empfehlung Nr. 20 arbeitet und somit wenig Spielraum für andere Formate lässt. Die nun folgenden Elemente entsprechen aber, abgesehen vom Namen, aus Kompatibilitätsgründen den jeweiligen Pendants in der BMEcat-Spezifikation.
Das Element NOCUPEROU kann die Anzahl der in einer Bestelleinheit befindlichen Inhaltseinheiten aufnehmen. Es muss nicht vorhanden sein.
Mit dem Element PRICEQUANTITY kann festgelegt werden, auf wieviele Bestelleinheiten sich die im Element PRICEDATA aufgeführten Preise beziehen. Auch dieses Unterelement muss nicht vorhanden sein.
Zur Angabe einer Mindestbestellmenge ist weiter das optionale Element QUANTITYMIN vorgesehen.
Das letzte Element im Element ORDERINFOS ist QUANTITYINTERVALL.

Es ist optional und kann einen Bestellintervall angeben. Ist hier z.B. der Wert 2 angegeben, so können nur 2, 4 bzw. weitere Vielfache von 2 als Bestellmenge akzeptiert werden.

3.3.6 Das Element PRICEDATA

Das Element PRICEDATE ist innerhalb des Wurzel-Elementes PRICEINFO das einzige Element, das mehr als einmal vorkommen kann. Dies ist mit der Tatsache begründet, dass es zu einem Artikel aufgrund von verschiedenen Gültigkeitszeiträumen mehrere Preisblöcke geben kann. Im BMEcat-Standard ist mit dem Element ARTICLE_PRICE_DETAILS das passende Gegenstück zu finden, dass von seiner Struktur dem hiesigen Element entspricht. Das folgende Bild zeigt den Aufbau des Elementes:
Innerhalb des Elementes PRICEDATA können sich maximal drei verschiedene

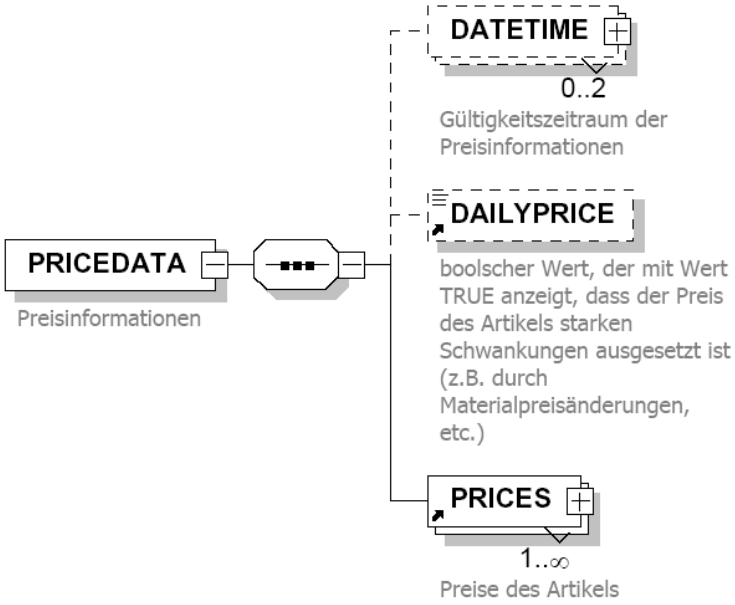


Abbildung 3.12: Aufbau des Elementes PRICEDATA

Elementtypen in unterschiedlicher Anzahl befinden.
Das erste Kindelement ist das bereits mehrfach erwähnte DATETIME-Element. Es entspricht auch an dieser Stelle dem Elementtyp typeDATETIME (siehe Seite 51), ist optional und kann maximal zwei Mal vorkommen. Es dient der Beschreibung eines Gültigkeitszeitraumes eines Preises. Die Festlegung, ob es sich bei der Angabe um den Anfangs- oder den Endzeitpunkt handelt, wird wieder über das Attribut *type* festgelegt. Ist der Wert des Attributes *price_start_date*, so handelt

es sich um den Anfangszeitpunkt der Gültigkeit und mit dem Attributwert *price_end_date* um den Endzeitpunkt.
Das Unterelement DAILYPRICE vom Datentyp dtBOOLEAN gibt mit dem Wert *TRUE* an, dass sich der angegebene Preis z.B. aufgrund ständiger Materialpreisänderungen ständig ändert.
Das letzte Kindelement PRICES muss mindestens einmal, kann aber auch beliebig oft innerhalb PRICEDATA vorkommen. Es dient der Aufnahme der tatsächlichen Preisinformationen und kann deshalb mehrmals vorkommen, weil dieses Element Preise nach unterschiedlichen Kriterien (Währung, Preistyp, etc.) und auch Staffelpreise darstellen kann. Es entspricht in seiner Funktion dem BMEcat-Element ARTICLE_PRICE, das ebenfalls den unterschiedliechen Preistypen, Rabatten und Staffelpreisen Rechnung trägt. Ein Blick auf den Aufbau des Elementes auf Seite 66 verdeutlicht dies.
Zur Abbildung der Anforderungen wie z.B. Darstellung des Preistyps, Darstellung von Staffelpreisen, etc. ist das Element PRICES wieder in maximal 10 verschiedene Elemente aufgeteilt, wovon aber maximal nur 9 gleichzeitig vorkommen dürfen.
Mit dem ersten Unterelement PRICETYPE wird der Typ des Preises festgelegt. Unter Preistyp wird hier verstanden, wie der Preis vom einkaufenden Unternehmen zu bewerten ist. Beispiele hierfür sind Netto-Listenpreis, Brutto-Listenpreis, kundenspezifischer Endpreis, etc.. Im Gegensatz zur BMEcat-Spezifikation, die einen Preistyp in Form eines Attributes dem Element ARTICLE_PRICE mitgibt, ist es in diesem XML-Modell als eigenes Element realisiert. In erster Linie soll dieses Element den aus dem ERP-System erhaltenen Preistyp in Form eines bis zu 255 Zeichen langen Strings aufnehmen können. Ein Mapping auf den von BMEcat oder einem anderen Katalogstandard erwünschten Typ soll bei der Ausgabe in Richtung Mediator erfolgen.
Das zweite, optionale Unterelement ist PRICEFLAG, das die selbe Funktion hat, wie das gleichnamige Element im HEADER (siehe Seite 58). Genauer gesagt überschreibt das Element PRICEFLAG im Element PRICES die als Default-Werte zu betrachtenden Angaben im HEADER. Die Struktur und Funktion dieses Elementes wurde direkt dem Element PRICE_FLAG im HEADER des BMEcat-Standard übernommen. Direkt bei den Preisinformationen ist diese Element in der BMEcat-Spezifikation jedoch nicht zu finden und soll an dieser Stelle im erstellten XML-Datenmodell eine höhere Flexibilität in der Darstellung von Preisen gewährleisten.
Die nun folgenden beiden Elemente BASPRICEAMOUNT und ENDPRICEAMOUNT dienen der Aufnahme der eigentlichen Preisbeträge. Dabei ist zu beachten, dass das Element BASEPRICEAMOUNT nicht vorkommen muss und einen aus dem ERP-System erhaltenen Basispreis aufnehmen kann. Da es sich bei den für das einkaufende Unternehmen ermittelten Preis meist um einen komplett berechneten Endpreis handelt, muss dafür das Element ENDPRICEAMOUNT, das eben diesen Wert aufnimmt, vorhanden sein. Das Element ENDPRICEA-

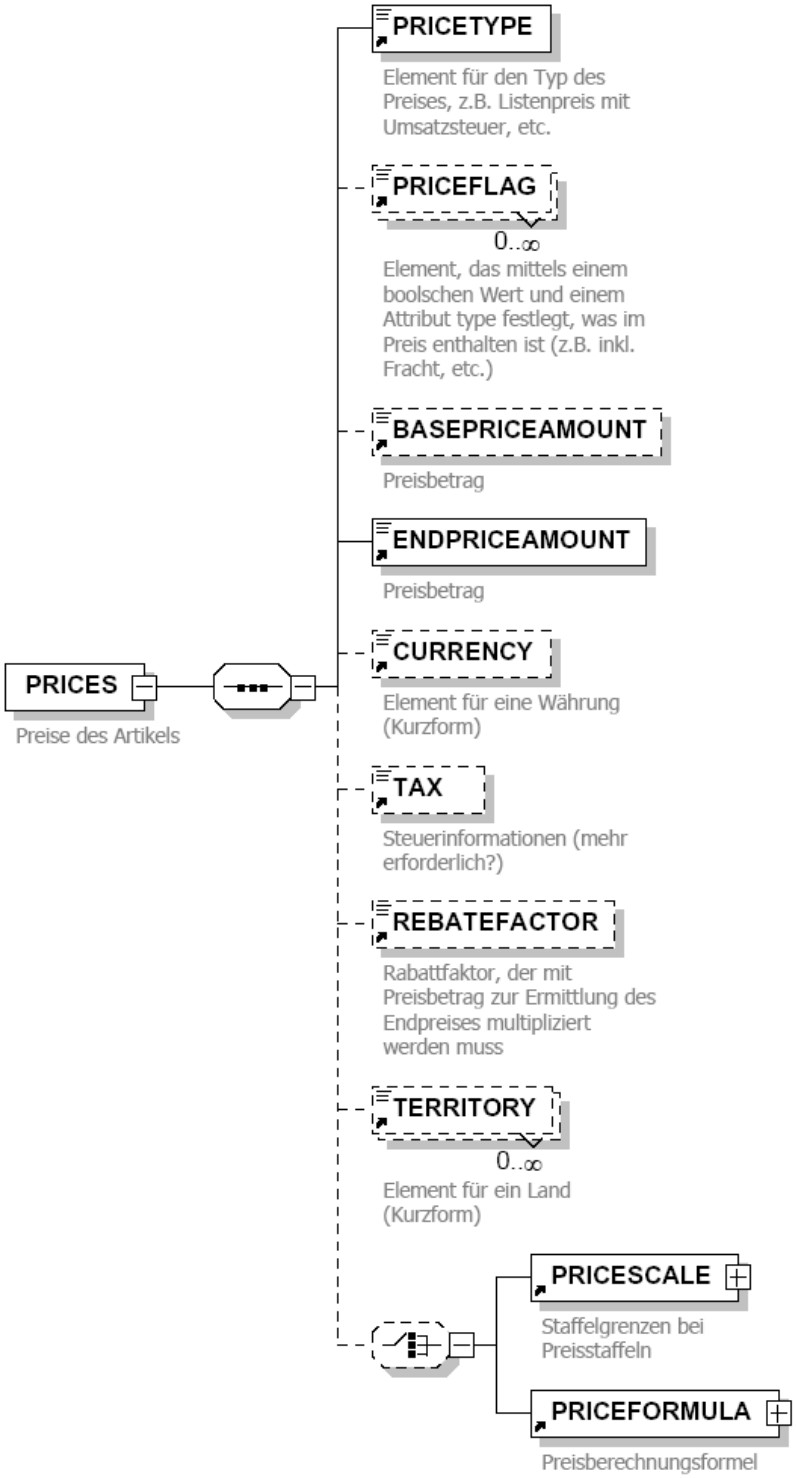


Abbildung 3.13: Aufbau des Elementes PRICES

MOUNT deckt sich damit in der Funktion mit dem Element PRICE_AMOUNT im BMEcat-Standard, wohingegen dort kein Pendant zum Element BASEPRICEAMOUNT zu finden ist.
Bei den nun folgenden Elementen CURRENCY, TAX, REBATEFAKTOR und TERRITORY findet sich jeweils ein direktes Pendant in der BMEcat-Spezifikation. Mit dem Element CURRENCY kann nochmals explizit die Währung für den angegebenen Preis festgelegt werden. Ist hier ein Wert vom Datentyp dtCURRENCY (siehe Seite 51) eingetragen, wird der als Default-Wert zu betrachtende Eintrag im HEADER überschrieben.
Im Kindelement TAX kann ein Faktor angegeben werden, der der Umsatzsteuer entspricht und der je nach Preistyp als zusätzliche Information gilt oder eben noch zur Ermittlung einer endgültigen Preisangabe beachtet werden muss.
Über das Element REBATEFAKTOR kann nochmals ein Faktor angegeben werden, der Einfluss auf den endgültigen Preis nimmt. Auch hier kann die Angabe lediglich als zusätzliche Information gelten, die bereits bei der Angabe des Preisbetrages beachtet wurde.
Die globale Verfügbarkeit kann auch innerhalb des Elementes PRICES nochmals mit dem optionalen Element TERRITORY eingeschränkt werden. Dazu können auch mehrere dieser Elemente vom Datentyp dtCOUNTRIES (siehe Seite 50) angegeben werden.
Zum Schluss kann noch über eine Auswahl zwischen den beiden Kindelementen PRICESCALE und PRICEFORMULA entschieden werden, ob es sich beim angegebenen Preis um einen festgelegten Staffelpreis handelt oder ob sich der Preis dynamisch anhand einer Preisformel berechnen lässt.
Das Element PRICESCALE ist zur Angabe der Staffelgrenzen nochmals in

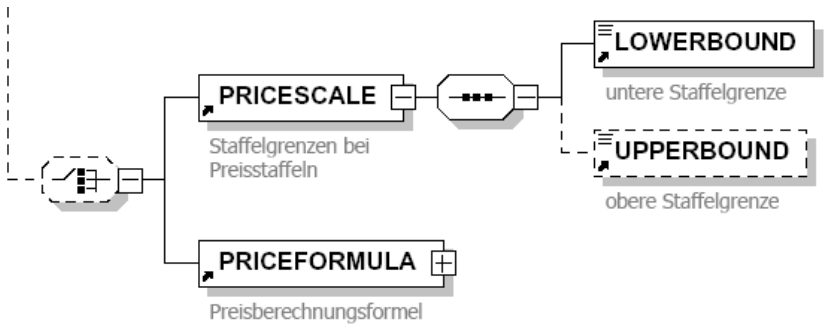


Abbildung 3.14: Die beiden PRICES-Kindelemente PRICESCALE und PRICEFORMULA

die beiden Kindelemente LOWERBOUND und UPPERBOUND aufgeteilt. LOWERBOUND muss vorhanden sein und gibt die untere Staffelgrenze an, während UPPERBOUND als optionale Information zu sehen ist und die obere Staffelgrenze festlegt. Ist das Element UPPERBOUND nicht vorhanden, begrenzt die

LOWERBOUND-Angabe des nächsten PRICES-Elementes die Preisstaffel. In der BMEcat-Spezifikation gibt es zur Darstellung der Staffelgrenzen nur das Element LOWER_BOUND, dass lediglich die untere Staffelgrenze beschreiben kann. Hier wurde das Datenmodell also dahingehend erweitert, dass auch eine Obergrenze hinterlegt werden kann, was der menschlichen Lesbarkeit dienlich ist.
Das Element PRICEFORMULA soll der angekündigten Weiterentwicklung des BMEcat-Standards Rechnung tragen und eine erste Möglichkeit bieten, eine Preisberechnungsformel abzulegen. Es entspricht dem Elementtyp typeFORMULA, der auf Seite 52 näher beschrieben wird. Im BMEcat findet sich hierzu bisher noch kein Gegenstück. Es ist aber für die kommende Version 2.0 des Standards vorgesehen, ebensolche Formeln zur Preisberechnung darstellen zu können. Mit dem Element PRICEFORMULA soll somit der Weiterentwicklung des Standards Rechnung getragen werden.

3.3.7 Das Element VARIANTLIST

Das letzte Element innerhalb des Wurzelelementes PRICEINFO ist das Element VARIANTLIST. Es trägt den Anforderungen der Unternehmen Rechenschaft, die unterschiedliche Variationen eines Produktes anbieten. Auch soll es auf die zukünftige Aufgabe vorbereiten, Produktkonfiguratoren, wie sie insbesondere bei Automobilherstellern oft auch im Internet eingesetzt werden, direkt anzubinden. Weiter dient dieses Element als Ausblick auf Möglichkeiten der Darstellung von verschiedenen Produktvarianten, wie sie z.B. in der kommenden Version 2.0 des BMEcat-Standards angekündigt sind. Das Element VARIANTLIST dient als Kontainer für die verschiedenen Varianten eines Produktes und deren Ausprägungen. Dazu muss es mindestens ein Element VARIANT enthalten, dass dann selbst wieder die näheren Informationen zu jeweils einem Variantenattribut aufnehmen kann. Über die verschiedenen Variantenattribute mit den jeweils zugehörigen Ausprägungen lässt sich dann die Produktvariation definieren.
Über diese Variantenattribute soll es auch möglich sein, variantenabhängige Artikelnummern zu generieren. Dazu wird die im PRICEINFO-Element (siehe Seite 60) hinterlegte Artikelnummer als eine Art Basisnummer betrachtet, an das die verschiedenen und von den Variantenausprägungen abhängigen Auswahlcodes mit oder ohne Separator-Zeichen angehängt werden können.
Um dies alles gewährleisten zu können, wurde das Element VARIANTLIST folgendermaßen aufgebaut:
Das erste Kindelement heißt VARIANTID und muss eine Identifikationsbezeichnung für das jeweilige Variantenattribut aufnehmen. Diese Bezeichnung muss ein String sein, der eine maximale Länge von 10 Zeichen haben darf.
Nach der Identifikationsbezeichnung folgt das zweite erforderliche Element namens VARIANTNAME. Wie der Name des Elementes schon erraten lässt, muss dieses Element eine Klartext-Bezeichnung des Variantenattributes aufnehmen. Diese Bezeichnung muss ein String mit maximal 150 Zeichen sein.

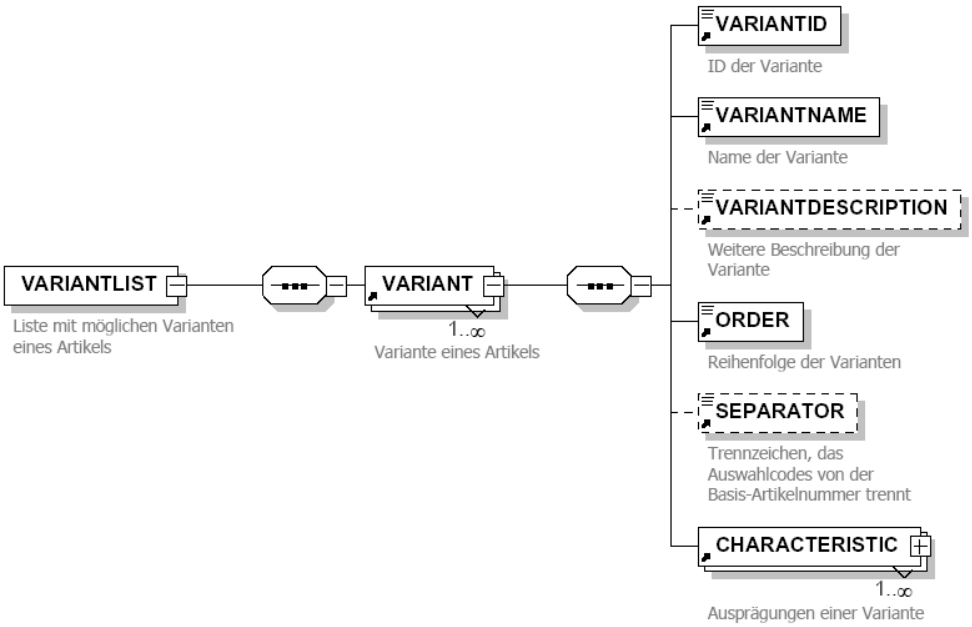


Abbildung 3.15: Aufbau des Elementes VARIANTLIST

Das erste optionale Element heißt VARIANTDESCRIPTION und kann eine weiter gehende Beschreibung des Variantenattributes aufnehmen. Wie bei den meisten Beschreibungs-Elementen nimmt auch dieses Feld einen String mit der maximalen Länge von 64.000 Zeichen auf.
Um die Reihenfolge der Variantenattribute insbesondere bei der Bildung der variantenabhängigen Artikelnummer festlegen zu können, wurde das Element ORDER eingeführt. Es kann einen beliebigen Integer-Wert aufnehmen und muss genau einmal vorhanden sein.
Das Kindelement SEPARATOR legt ein Trennzeichen fest, das bei der Bildung variantanabhängiger Artikelnummern zwischen dem bisher gebildeten Artikelnummer-String und dem durch die Auswahl der Variantenausprägung festgelegten Auswahlcode eingefügt wird. Es kann dazu einen nicht weiter festgelegten String aufnehmen.
Das letzte Unterelement trägt den Namen CHARACTERISTIC und muss mindestens einmal, kann aber auch mehrmals innerhalb jedes VARIANT-Elementes vorhanden sein. Es steht für die einzelnen Variantenausprägungen, die ein Variantenattribut annehmen kann. Aus diesem Grund ist es selbst nochmals in sechs verschiedene Elemente unterteilt, von denen nur fünf Elemente gleichzeitig vorhanden sein dürfen.
Das erste Kindelement entspricht dem Auswahlcode und heißt deshalb CHOICECODE. Es muss einen bis zu 10 Zeichen langen String aufnehmen, der eindeutig für die jeweilige Variantenausprägung steht. Bei dem Inhalt dieses Elementes han-

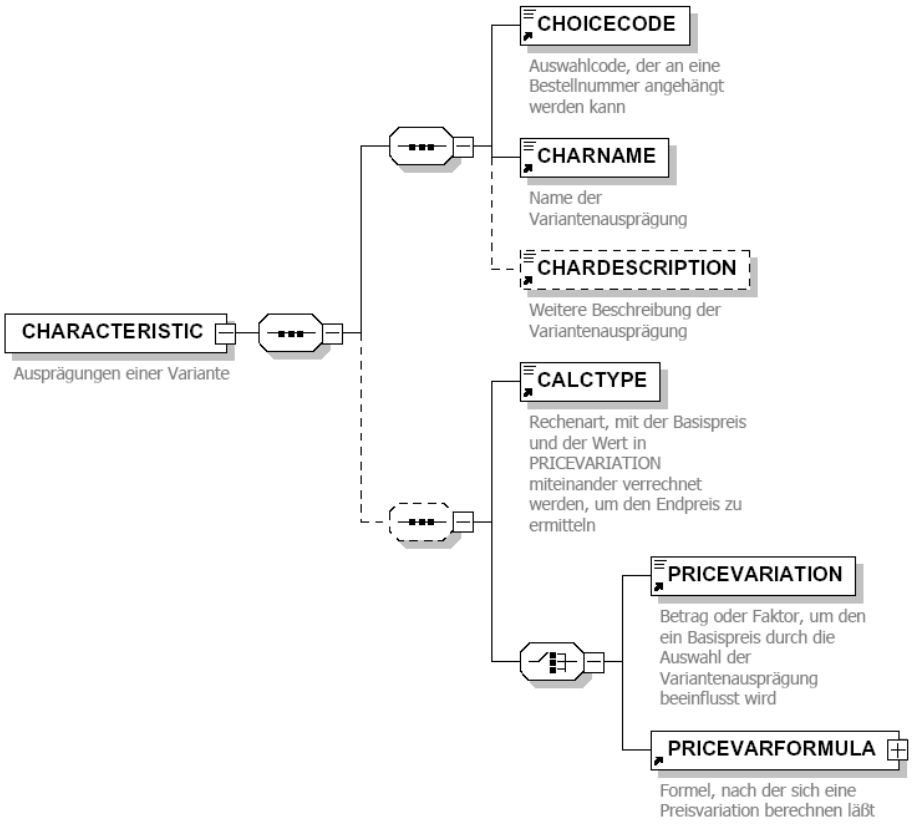


Abbildung 3.16: Aufbau des Elementes CHARACTERISTIC

delt es sich also um die Zeichenkombination, die bei Auswahl der jeweiligen Ausprägung eines Variantenattributes an die Basis-Artkelnummer angehängt wird.
Im Unterelement CHARNAME kann mit einem maximal 150 Zeichen langen String eine Klartext-Bezeichnung der jeweiligen Variantenausprägung abgelegt werden. Wie das Vorgängerelement handelt es sich hierbei auch im ein Pflichtelement. Das optionale Element CHARDESCRIPTION kann die vorherigen Angaben noch durch eine weitere Beschreibung der Ausprägung ergänzen. Es kann einen maximal 64.000 Zeichen langen String aufnehmen.
Da offensichtlich auch eine variantenabhängige Beeinflussung des Endpreises mit abgebildet werden soll, mussten hierfür auch Elemente vorgesehen werden, die diese Anforderungen erfüllen. Das erste für die Darstellung von Preisabhängigkeiten wichtige Unterelement ist CALCTYPE, das mittels einem String mit der maximalen Länge von 10 Zeichen angegeben werden muss, wenn die gesamte Sequenz mit Preisbeeinflussungsinformationen vorhanden ist. Das Element CALCTYPE kann somit eine Rechenart aufnehmen, mit der die im folgenden beschriebene Preisvariation den Endpreis beeinflusst.
Die eigentliche Preisvariation kann auf zweierlei Arten erfasst werden. Zum einen

kann mit dem Element PRICEVARIATION direkt ein fertig berechneter Faktor oder Wert hinterlegt werden, der dann im Zusammenspiel mit der Rechenart den Endpreis bestimmt.
Eine zweite Möglichkeit bietet das Element PRICEVARFORMULA, das vom Elementtyp typeFORMULA (siehe Seite 52) ist und einen Formelstring aufnehmen kann, mit dem die den Endpreis beeinflussende Preisvariation errechnet werden kann.
Eines der beiden Elemente muss zusammen mit dem Element CALCTYPE vorhanden sein, wenn Variationen den Produktendpreis beeinflussen sollen.

3.4 Fazit

Mit der PRICEINFO-XML-Datenstruktur konnte ein Datenkontainer geschaffen werden, der nahe am BMEcat-Standard bereits vom ERP-System berechnete Preisinformationen aufnehmen kann. Diese deutliche, aber abgespeckte Nähe zur BMEcat-Spezifikation ist zum einen mit dem fast ausschließlichen Einsatz dieses Katalogstandards auf dem deutschen Markt und damit auch bei den Kunden der viaMEDICI Software GmbH und zum anderen mit dessen deutlich am weitesten entwickelten Struktur des Standards begründet. Darüberhinaus ist es oftmals besser auf bewährte Strukturen zu setzen, als „das Rad neu zu erfinden“.
Mit der Möglichkeit, Produktvariationen und Preisberechnungsformeln darzustellen, schaut das Datenmodell bereits in Richtung Zukunft der Katalogstandards und trägt damit dem dort angekündigten Weiterentwicklungen Rechnung.

4 Import von Preisinformationen aus ERP-Systemen

Nachdem nun die Datenstruktur des XML-Dokumentes mittels der XML Schema Definitionssprache festgelegt wurde, ist es eine spannende Frage, wie die Daten aus einem ERP-System in dieser Datenstruktur abgelegt werden können. Da die Lösung der gesamten Problemstellung den zeitlichen Aufwand der Diplomarbeit übersteigen würde, wird hier nur ein Teilproblem gelöst: Die Transformation ERP-spezifischer Daten am Beispiel SAP in die erstellte Datenstruktur.
Die grundsätzliche Funktionsweise des Imports der Daten aus einem ERP-System wird hier am Beispiel SAP grob beschrieben. Als ERP-System für die Lösung des Problemes wurde bei der viaMEDICI Software GmbH bewusst SAP gewählt, weil das ERP-System weltweit am meisten Verbreitung hat und die viaMEDICI Software GmbH eng mit SAP kooperiert. Darüberhinaus setzt der Großteil des Kundenstammes der viaMEDICI Software GmbH SAP ein.

4.1 Funktionsweise

Zum Import von Daten aus dem ERP-System von SAP in viaMEDICI EPIM wurde eine Schnittstelle mit dem Namen viaCONNECT SAP konzipiert, die auf den SAP-Standardschnittstellen RFC (Remote Function Call) und BAPI (Business Application Programming Interfaces) aufsetzt. Über diese Schnittstelle sollen insbesondere Daten aus den SAP-Modulen MM (Materials Management - Materialwirtschaft), SD (Sales & Distribution - Verkauf und Vertrieb) und PLM (Product Lifecycle Management - Verwaltung des Produkt Lebenszyklus) in das System der viaMEDICI Software GmbH importiert werden, wobei die Daten des Moduls MM sehr stark mit den Daten der anderen beiden Module verknüpft sind, da es sich dabei um den eigentlichen Materialstamm handelt. Für die Befüllung des in Kapitel 3 (siehe Seite 40) erstellten XML-Datenkontainers sind insbesondere die Daten aus dem Modul SD interessant.
Die Schnittstelle viaCONNECT SAP verwendet den Java Connector von SAP und eine Sammlung von Java-Beans, die von viaMEDICI erstellt wurden. Diese Schnittstelle hat nun die Aufgabe, die über den Aufruf einer SAP-Funktion, genauer die Simulation einer Bestellung, erhaltenen Daten entgegenzunehmen und sie mittels generischer Java-Module in ein SAP-spezifisches XML-Dokument zu schreiben. Dabei sind die Java-Beans die treibende Kraft, die über den Java

Connector (JCO) einen BAPI-Aufruf an das SAP-System schicken. Durch diesen Aufruf über den JCO werden Proxies eingerichtet, die die vom SAP-System erhaltenen Daten zwischenpuffern. Die Java-Beans holen sich die notwendigen Daten dann aus diesen JCO-Proxies und bauen daraus ein SAP-spezifisches XML-Dokument zusammen.

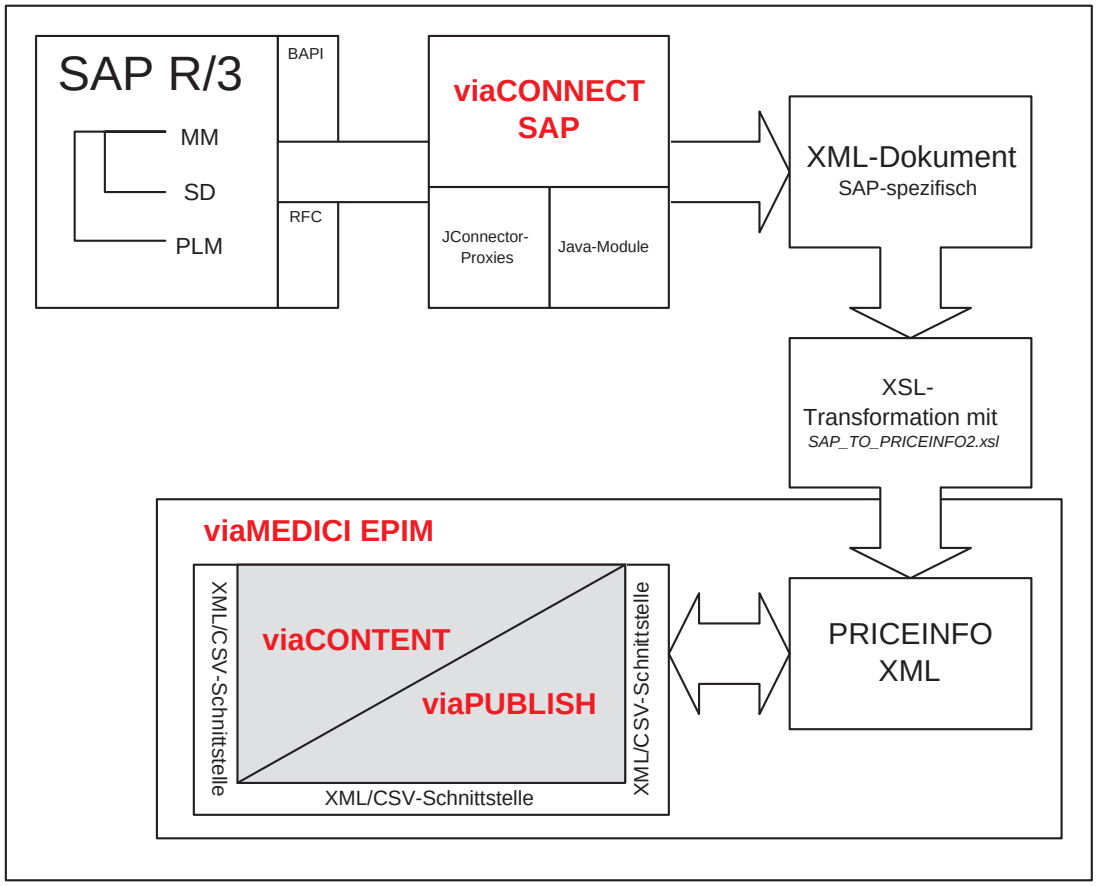


Abbildung 4.1: *Der Import von SAP-Daten in viaMEDICI EPIM*

Da der Schnittstelle über eine graphische Benutzeroberfläche bzw. eine Textdatei mehrere Informationen wie z.B. verschiedene Kunden- und Artikelnummern übergeben werden können, wird von der Schnittstelle ein SAP-spezifisches XML-Dokument zurückgegeben, das die Informationen von mehreren Kunden bzw. Artikeln beinhaltet. Das Dokument ist im Groben folgendermaßen aufgebaut:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ROOT>
3   <GUIVALUES>
4     Auflistung der Eingaben ueber die Benutzer
```

```
5   </GUIVALUES>
6   <MATERIAL>
7     <MATERIALNUMBER>Materialnummer</MATERIALNUMBER>
8     <CUSTOMER>
9       <CUSTOMERNUMBER>Kundenummer</CUSTOMERNUMBER>
10      <SALEORDER>
11        <QUANTITY>1</QUANTITY>
12        <BILLING_PARTY>
13          Daten zur rechnugsempfangenden Partei
14        </BILLING_PARTY>
15        ...
16        <SHIP_TO_PARTY>
17          Daten zur Partei, an die ausgeliefert wird
18        </SHIP_TO_PARTY>
19        <SOLD_TO_PARTY>
20          Daten zur Partei, an die verkauft wird
21        </SOLD_TO_PARTY>
22        <ORDER_SCHEDULE_EX>
23          <item>
24            berechnete zeitliche Informationen zur simulierten
                Bestellung
25          </item>
26          ...
27        </ORDER_SCHEDULE_EX>
28        <ORDER_PARTNERS>
29          <item>
30            Daten zur bestellenden Partei
31          </item>
32        </ORDER_PARTNERS>
33        <ORDER_ITEMS_OUT>
34          <item>
35            von SAP ermittelte Daten zur Bestellung
36          </item>
37        </ORDER_ITEMS_OUT>
38        <ORDER_ITEMS_IN>
39          <item>
40            an SAP zur Berechnung der Bestellung uebergebene
                Daten
41          </item>
42        </ORDER_ITEMS_IN>
43        ...
44        <ORDER_CONDITION_EX>
45          <item>
46            eigentliche Preisinformationen in Form von
                Konditionen
47          </item>
48          ...
49        </ORDER_CONDITION_EX>
50      </SALEORDER>
51      ...
52    </CUSTOMER>
```

```
53      ...
54    </MATERIAL>
55    ...
56  </ROOT>
```

Listing 4.1: SAP-spezifisches XML-Dokument mit Bestelldaten

Hier beginnt nun der Teil des Imports von Daten aus ERP-Systemen, der im weiteren Fortlauf dieses Kapitels näher beschrieben werden soll: Die Transformation der Daten im SAP-spezifischen XML-Dokument in die in Kapitel 3 (siehe Seite 40) festgelegte XML-Datenstruktur für Preisinformationen mittels XSLT (eXtensible Stylesheet Language Transformations). Über diese Transformation gelangen die Daten in das festgelegte XML-Format und werden so in viaMEDICI EPIM abgelegt.

4.2 Transformation

Schon bevor die eigentliche Transformation des XML-Dokumentes via XSLT angegangen werden konnte, stellten sich die ersten Probleme in den Weg:
Das erste Problem, dass sich bereits bei den Vorüberlegungen zeigte, war die Frage, wie man die einzelnen XML-Dateien so strukturieren kann, dass von viaMEDICI EPIM ohne größere Probleme für die Anzeige der Daten darauf zugegriffen werden kann?
Aus der Tatsache, dass die Schnittstelle zum SAP-System nur eine XML-Datei mit den Daten von mehreren Kunden und mehreren Artikeln zurückliefert, ergab sich das zweite Problem: Wie kann die große XML-Datei ohne großen programmiertechnischen Aufwand in viele kunden- und artikelspezifische XML-Dateien zerlegt werden?
Und ein drittes Problem blieb bei der näheren Betrachtung der SAP-spezifischen XML-Datei auch nicht aus: Was passiert mit den Informationen, die die SAP-Schnittstelle nicht bzw. nicht im richtigen Format liefern kann, aber vom erstellten XML-Datenkontainer und auch später von einem Katalogdokument im BMEcat-Standard in einem bestimmten Format verlangt werden?

Zur Lösung der zuvor beschriebenen Probleme wurden zuerst Lösungsansätze gesucht, die im folgenden näher beschrieben werden sollen.

Strukturierung der Ergebnisdateien

Noch bevor Überlegungen zur technischen Realisiesrung der Zerlegung der einen großen XML-Datei in viele kleinere XML-Dateien gemacht wurden, musste die Frage geklärt werden, wie diese Dateien dann so strukturiert werden können,

dass von viaMEDICI EPIM aus ein Zugriff möglich ist. Dabei kamen bei ersten Vorüberlegungen mehre Möglichkeiten in Betracht:

- **Speicherung der Dateien ohne weitere Strukturierung**
  Der einfachste Weg, die Ergebnis-XML-Dateien abzuspeichern, wäre die Speicherung ohne weitere Strukturierung in Form von Verzeichnissen gewesen. Das heißt konkret, dass alle als Resultat der XSL-Tranformation entstandenen XML-Dateien in einem zuvor definierten Verzeichnis abgelegt werden, ohne weitere Unterordner zur besseren Strukturierung anzulegen.
  Der Vorteil dieser Methode ist sicherlich, dass eine Speicherung der XML-Dateien ohne größeren Aufwand realisiert werden kann.
  Werden aber die potentiellen Datenmengen bei mehreren 1000 Artikeln und einigen 100 Kunden betrachtet, zeigt sich sofort, dass sich beim späteren Zugriff auf die Dateien durch das Softwaresystem viaMEDICI EPIM Probleme mit der Geschwindigkeit des Zugriffes ergeben können. Darüberhinaus ist es auch für Menschen mit direktem Zugriff auf die Verzeichnisse des Servers schwer, mit diesem großen und unstrukturierten Datenaufkommen zurecht zu kommen.

- **Speicherung der Dateien nach Kunden**
  Ein zweiter Ansatz ist die Speicherung sortiert nach Kunden. Das heißt konkret, dass für jeden Kunden in einem zuvor definierten Verzeichnis jeweils ein Unterordner erstellt wird, der dann die XML-Dateien zu sämtlichen dem Kunden zugeordneten Artikel aufnimmt.
  Der klare Vorteil dieser Methode ist sicherlich, dass die Daten weiter strukturiert werden.
  Wird aber die Tatsache betrachtet, dass einem Kunden mehrere 1000 Artikel zugeordnet sind, so lässt sich leicht erkennen, dass auch hier Probleme mit der Geschwindigkeit des Zugriffes an der Tagesordnung stehen können. Darüberhinaus würde sich der Zugriff aus dem EPIM-Systen nicht so einfach gestalten, da viaMEDICI EPIM wie bereits beschrieben artikelorientiert arbeitet.

- **Speicherung der Dateien nach Artikeln**
  Eine dritte Möglichkeit bietet sich mit der Speicherung der XML-Dateien sortiert nach Artikeln. Das heißt konkret, dass für jeden Artikel in einem zuvor definierten Verzeichnis ein Unterverzeichnis angelegt wird, in dem dann die XML-Dateien für alle Kunden abgelegt werden, denen dieser Artikel auch zugeordnet ist.
  Hier zeigt sich der erste Vorteil schon darin, dass viaMEDICI EPIM artikelbezogen arbeitet und somit ein recht einfacher Zugriff auf die Daten möglich wäre. Weiter ist die Gesamtanzahl der Kunden eines Unternehmens, insbesondere bei Herstellern von z.B. Elektro- und Elektonikbauteilen, oftmals

deutlich geringer als die Anzahl der vertriebenen Artikel. Auch die automatische Erstellung der z.B. mit der Artikelnummer benannten Unterverzeichnisse sollte relativ einfach zu realisieren sein.
Sollte ein Kunde einen sehr groen Artikelbestand haben, besteht bei diesem Lösungsansatz noch die Möglichkeit, die Verzeichnisse hierarchisch zu gestalten. Dabei werden dann nicht gleich die gesamten Artikelnummern als Verzeichnisse genommen, sondern anhand der meist bei Kunden vorhandenen Nummernkreise für Artikelnummern eine hierarchische Verzeichnisstruktur geschaffen. Diese Verzeichnisstruktur hilft dann sowohl dem Softwaresystem als auch den betreuenden Menschen bei der Verwaltung und Ansicht der Daten.

Werden die drei Möglichkeiten näher betrachtet, zeigt sich gleich, dass eigentlich nur die dritte Möglichkeit für die Realisierung der Strukturierung der resultierenden Dateien in Frage kommt. Der für diese Aufgabe ausgewählte Lösungsansatz wird am Ende der Problemlösung des zweiten Problemes aufgezeigt, weil diese beiden Probleme sehr eng zusammenhängen.

Zerlegung der SAP-XML-Datei

Nachdem geklärt ist, wie die resultierenden XML-Dateien strukturiert werden können, geht es nun an das zweite Problem: Wie können große XML-Dateien in mehrere kleinere XML-Dateien zerlegt werden und das mit möglichst geringem Aufwand? Unter geringem Aufwand verstand sich dabei, die Dateien nach Möglichkeit ausschließlich via XSLT zu zerlegen. Ein Ausschlusskriterium war dabei noch, dass das zu erstellende XSL-Skript mit dem innerhalb von viaMEDICI EPIM verwendeten XSLT-Prozessor Xalan-J ([XML.Apache]) funktioniert. Als einzige realisierbare Möglichkeit wurde über eine Suche im Internet die Nutzung der in der XSLT-Spezifikation 1.0 möglichen Extensions gefunden. Extensions sind Möglichkeiten der Erweiterungen der XSLT-Spezifikation, die an dieser Stelle nicht weiter definiert sind. Das hat zur Folge, dass die tatsächlich zur Verfügung stehenden Extensions je nach verwendetem XSLT-Prozessor variieren. Xalan-J bietet für das Umleiten von XSLT-Ergebnissen eine Extension namens *redirect* ([XALAN-REDIRECT]). Diese Extension ermöglicht es mit den Elementen *open*, *write* und *close*, Dateien zu öffnen, das Resultat der XSL-Transformation zu schreiben und die Datei wieder zu schließen. Wird das *redirect*-Element *write* alleine benutzt, so wird die mit dem Attribut *select* an das Element übergebene Datei geöffnet, das Ergebnis dort hinein geschrieben und umgehend nach dem Schreiben wieder geschlossen.
Der folgende Codeausschnitt aus der XSL-Datei SAP_TO_PRICEINFO2.xsl zeigt die Realisierung des Dateisplittings:

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/
       XSL/Transform" xmlns:redirect="http://xml.apache.org/xalan/
       redirect" extension-element-prefixes="redirect" xmlns:java="
       http://xml.apache.org/xalan/java" exclude-result-prefixes="
       java">
3  <xsl:template match="ROOT">
4    <xsl:for-each select="MATERIAL">
5      <xsl:for-each select="CUSTOMER">
6        <xsl:if test="./SALEORDER[1]/BILLING_PARTY/PAYER != ''">
7          <xsl:variable name="path">Pfad zum Zielverzeichnis</xsl:
               variable>
8          <xsl:variable name="folder" select="concat(../
               MATERIALNUMBER, '\')"></xsl:variable>
9          <xsl:variable name="language" select="java:mapping.
               Mapping.getMappedString(./SALEORDER[1]/BILLING_PARTY/
               LANGU, 'langSAP')"></xsl:variable>
10         <xsl:variable name="file" select="concat(../
               MATERIALNUMBER, '_',CUSTOMERNUMBER, '_', $language)"/>
11         <redirect:write select="concat($path, $folder, $file,'.
               xml')">
12           ...
13           <!--Hier kommen die Daten, die in die Datei geschrieben
                 werden-->
14           ...
15         </redirect:write>
16       </xsl:if>
17     </xsl:for-each>
18   </xsl:for-each>
19 </xsl:template>
20 </xsl:stylesheet>
```

Listing 4.2: Realisierung des Datei-Splittings mit XSLT

In der Zeile 11 wird der Aufruf des *redirect*-Elementes *write* gezeigt, dessen Attribut *select* den mittels der XPath-Funktion *concat* und einigen in den Zeilen 7 bis 10 definierten Variablen generierten Dateinamen inklusive Pfad übergeben bekommt. Über dieses Element lassen sich auch gleich die in der Problemstellung zuvor beschriebenen Verzeichnisse für die einzelnen Artikel anlegen, indem einfach die Materialnummer mittels der Variablen *folder* (in Zeile 8) als Zielverzeichnis definiert wird.
Sobald hier aber ein anderer XSLT-Prozessor wie z.B. Saxon ([SAXON]) oder nur die C-Version des Xalan eingesetzt wird, muss nach anderen Elementen gesucht werden, die dieses Dateisplitting ermöglichen. Bei Saxon wird dieses Dateisplitting durch das Element *<saxon:output>* durchgeführt und bei der C-Version des Xalan sind keine Extensions dieser Art möglich. eine weitere Möglichkeit soll noch das Element *<xsl:document>* bieten, dass zu XSLT 1.1 gehört. Leider wurde mit diesem nach der allgemeinen Einführung von XSLT 1.1 wohl allgemeingültigen Element bei einer Transformation mit Xalan-J kein Ergebnis erzielt.

Probleme mit SAP-Daten

Die dritte Problemstellung wurde durch die aus dem SAP-System erhaltenen Daten selbst ausgelöst. Bei genauerer Betrachtung des SAP-spezifischen XML-Dokumentes wurden einige Probleme mit den Daten festgestellt:

- **Keine sinnvollen Daten vorhanden**
  Zu einigen Artikel-Kunden-Beziehungen konnten für manche Spracheinstellungen keine sinnvollen Daten gefunden werden. Dies zeigt sich im SAP-XML-Dokument so, dass im Element SALEORDER außer der ausgewählten Bestellmenge (QUANTITY) keine weiteren Elemente befinden. Stattdessen wird direkt innerhalb des SALEORDER-Elementes ein Text der Art „Material 1001467 in VkOrg 0100, VtWeg 01, Sprache DE ist nicht vorgesehen“ hinterlegt.
  Dieses Problem wurde durch eine if-Abfrage im XSL-Dokument gelöst. Die if-Abfrage ist auch im vorhergehenden Listing (siehe Seite 77) in der Zeile 6 zu sehen. Diese if-Abfrage filtert über die Abfrage des Inhaltes des Elementes PAYER im Unterelement BILLING_PARTY des ersten SALEORDER-Elementes eines Kundens alle die Informationen heraus, die keine weiteren Preis- und Bestellinformationen beinhalten.

- **Keine Daten vorhanden**
  Durch das ausschließliche Simulieren einer Bestellung in einem SAP-System, war es bisher nicht möglich, Informationen zu den Gültigkeitszeiträumen von Preisen zu erhalten. Da als System für den Test der SAP-Schnittstelle nur ein Test-System eines der Kunden der viaMEDICI Software GmbH mit Kopien der Originaldaten zur Verfügung steht, auf dem für den Test keine Schreibrechte zur Verfügung stehen, konnten weitere Möglichkeiten noch nicht geprüft werden. Zu diesen weiteren Möglichkeiten zählt auch das Anlegen von Bestellungen, über das dann auch im System hinterlegte Gültigkeitszeiträume erreicht werden sollen.
  Zum ordentlichen Test der Transformation war es hier aber notwendig, die beiden Pflichtelemente für das Erstellungsdatum und das Anfangsdatum der Gültigkeit eines PRICEINFO-XML-Dokumentes nach dem erstellten XML Schema mit sinnvollen Daten zu füllen.
  Auch hier konnte auf die Möglichkeit der Extensions des XSL-Prozessors Xalan-J zurückgegriffen werden, die es auch ermöglichen, Java-Klassen aus dem XSL-File (siehe auch [XALAN-JAVA]) heraus aufzurufen. So lies sich eine Klasse erstellen, die das aktuelle Datum im gewünschten Format „yyyy-mm-dd“ in die erstellten XML-Dateien einfügen kann.
  Das folgende Listing zeigt das Einbinden der Java-Extension und den Aufruf der Java-Klassenmethode Today.today() in der XSL-Datei:

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
       /1999/XSL/Transform" xmlns:redirect="http://xml.apache.
       org/xalan/redirect" extension-element-prefixes="redirect"
        xmlns:java="http://xml.apache.org/xalan/java" exclude-
       result-prefixes="java">
3  <xsl:template match="ROOT">
4    ...
5              <HEADER>
6                ...
7                <DOCDATETIME>
8                  <DATE><xsl:value-of select="java:Today.today
                       ()"/></DATE>
9                </DOCDATETIME>
10               ...
11             </HEADER>
12   ...
13 </xsl:template>
14 </xsl:stylesheet>
```

Listing 4.3: Aufruf von Java-Klassen in XSL-Dateien

- **Mapping von Daten** Neben den beiden zuvor geschilderten Fällen gab es auch noch den Fall, dass aus dem SAP-System erhaltene Dateien nicht mit den von viaMEDICI EPIM bzw. später auch BMEcat verlangten Datenformaten übereinstimmten. Zur Anpassung dieser Daten wurde somit ein Mapping erforderlich, dass die erhaltenen Daten in die geforderten Formate wandeln kann. Probleme gab es bei der Transformation des SAP-XML-Dokumentes lediglich bei der Sprache, die aus dem SAP-System als einzelner Buchstabe (z.B. D für deutsch) ausgegeben wurde, in viaMEDICI aber als Kombination aus zwei Buchstaben benutzt wird (z.B. de für deutsch).
  Dazu wurde ebenfalls auf die Möglichkeit der Java-Extension zurückgegriffen und eine Java-Klasse erstellt, die eine Properties-Text-Datei einlesen kann und anhand eines übergebenen Wertes den in das Zielformat passenden Gegenwert zurückgibt. Diese Java-Klasse wird im Kapitel 6 (siehe Seite 90) noch näher beschrieben, weil sie bei der Ausgabe-Transformation in Richtung elektronischer Marktplätze weit öfter Anwendung findet. Im Listing auf Seite 77 ist in der Zeile 9 das Mapping der Sprachinformation mittels der Java-Klasse *Mapping* zu sehen.

Erstellung der Transformationsdatei

Nach Lösung der genannten Probleme konnte eine funktionsfähige XSL-Datei erstellt werden, die die Daten aus dem SAP-spezifischen XML-Dokument in ein XML-Dokument transformieren kann, dass dem PRICEINFO-XML Schema ent-

spricht. Dazu wurden aus dem SAP-spezifischen Dokument die folgenden Daten verwendet:

- **HEADER der PRICEINFO-Datei**
  Zur Erzeugung des PRICEINFO-Unterelementes HEADER wurden eine Vielzahl verschiedener Daten aus dem SAP-XML-Dokument benötigt.
  Der Dateinamen, der auch gleichzeitig der DOCID und dem DOCTITLE entspricht, wurde aus dem Element MATERIALNUMBER und dem CUSTOMER-Kindelement CUSTOMERNUMBER, beides Unterelemente des /ROOT/MATERIAL-Elementes, generiert. Hinzu kommt noch das bereits via Java-Funktion gemappte Sprachkürzel, das dem Elementpfad SALEORDER/BILLING_PARTY/LANGU entnommen wurde, der ebenfalls in /ROOT/MATERIAL/CUSTOMER zu finden ist. Die drei genannten Variablen werden mit Unterstrichen getrennt in der genannten Reihenfolge aneinandergehängt und bilden so den Dateinamen.
  Das Element SUPPLIERID wird aus dem Elementpfad SALEORDER[1]/ORDER_ITEMS_OUT/item/PLANT mit dem notwendigen Inhalt versorgt. Leider handelt es sich bei den aus SAP erhaltenen IDs für Geschäftsparteien nur um SAP-interne Werte, die dann in Richtung BMEcat zwecks der Eindeutigkeit nochmals auf DUNS- oder ILN-Nummern gemappt werden müssen.
  Die Informationen zum einkaufenden Unternehmen im Element BUYER werden komplett aus dem Elementpfad SALEORDER[1]/BILLING_PARTY genommen. Dort finden sich einige Daten zum einkaufenden Unternehmen wie z.B. die Anschrift und eine Telefonnummer.
  Weitere verwertbare Daten z.B. zu weiterführenden Verträgen, etc. können dem SAP-System derzeit nicht entnommen werden und werden deshalb in der Transformations-Datei nicht weiter beachtet.

- **Das Element ARTICLEDATA**
  Das PRICEINFO-Unterelement ARTICLEDATA bezieht seine Daten aus dem SAP-Element SALEORDER[1]/ORDER_ITEMS_OUT. Dort finden sich nähere Informationen zum Artikel selbst, wie z.B. die Artikelnummer und ein Artikelkurztext, und auch zu den gewünschten Bestellinformationen. Allerdings ist auch hier die Datenausbeute nicht wirklich groß: Lediglich die Verpackungseinheit im SAP- und ISO-Format und eine Mindestbestellmenge konnten den SAP-Daten entnommen werden.

- **Die Preisinformationen**
  Die eigentlichen Preisinformationen, die im PRICEINFO-Element PRICEDATA gespeichert werden, wurden den Unterelementen item des SALEORDER-Elementes ORDER_CONDITION_EX entnommen. Dabei entspricht jedes item-Element einer neuen Preisinformation, die in einem eigenen PRICE-Element des PRICEINFO-Modelles gespeichert wird.

Auch hier konnten die Möglichkeiten des erstellten XML-Datenkontainers nicht ausgeschöpft werden, weil lediglich die Währung, der Preisbetrag und die Untere Staffelgrenze eines Staffelpreises dem SAP-XML-Dokument entnommen werden können. Hinzu kommt noch der Preistyp, der an dieser Stelle dem Konditionstyp des SAP-Dokumentes entspricht. Auch hier muss in Richtung BMEcat ein Mapping der Information erfolgen.

- **Produktvariationen**
  Zur Füllung des PRICEINFO-Elementes VARIANTLIST konnten dem SAP-XML-Dokument leider keine Daten entnommen werden. Hier wird wohl zu einem späteren Zeitpunkt eine Erweiterung der SAP-Schnittstelle notwendig werden.

Da für eine vollständige Implementierung der Transformation in das Softwaresystem viaMEDICI EPIM sowohl bei mir als auch bei den Entwicklern der viaMEDICI Software GmbH die Zeit fehlte, konnte die erstellte XSL-Datei nur mit einem direkten Konsolenaufruf des Xalan-J-Prozessors gestestet werden. Zum Einsatz kam dabei die Version 2.5.2 des XSLT-Prozessors, der zuvor nach Anleitung unter [XALAN-INSTALL] auf einem Rechner installiert und eingerichtet werden musste.
Mit dem Aufruf

```
1 java org.apache.xalan.xslt.Process -in
    SalesDistribution_20040127111456.xml -xsl SAP_TO_PRICEINFO2.
    xsl
```

Listing 4.4: Aufruf des Xalan-Prozessors

konnte dann abschließend eine erfolgreiche Testtransformation durchgeführt werden, die ein großes SAP-Ergebnis-XML-Dokument in mehrere artikel-, kunden- und sprachbezogene XML-Dateien aufteilte und diese in nach den Artikelnummern benannten Unterordnern ablegte.

4.3 Integration

Eine Integration der Transformation in viaMEDICI EPIM konnte leider nicht mehr vorgenommen werden, weil erstens aufgrund von Terminverschiebungen, die mehr oder weniger direkt diese Arbeit betrafen, nicht mehr genügend Zeit zur Verfügung stand und zweitens kein systemkundiger Entwickler der viaMEDICI Software GmbH genügend Zeit hatte, um bei der Integration beizustehen.
Aus diesem Grund kann hier nur der Vorschlag erbracht werden, die Transformation der SAP-spezifischen Daten in das im Rahmen dieser Arbeit erstellten XML-Datenmodell direkt an die Schnittstelle zum SAP-System anzugliedern. Wie das

genau gemacht werden kann, kann an dieser Stelle nicht beschrieben werden, weil zu wenig Einblicke in die Hintergründe der Schnittstelle möglich waren. Zu beachten ist hier aber auch, dass die eingesetzten Pfade im XSL-Dokument noch an die systemweit gültigen und zentral konfigurierbaren Pfadkonfigurationen angepasst werden müssen.
Es sollte auch die Notwendigkeit einer Aktualisierung der momentan im System verwendeten Versionen der XML- und XSLT-Bibliotheken überprüft werden.

4.4 Fazit

Nach dem Erhalt von Daten aus der Schnittstelle zum SAP-System hat sich sehr schnell gezeigt, dass ein einfaches Übernehmen von SAP-spezifischen Daten in einen an Katalogstandards orientierten XML-Datenkontainer nicht ausreicht, um später daraus ein einem Katalogstandard wie z.B. BMEcat entsprechendes Dokument zu erstellen. Viele wichtige Informationen werden in einem anderen Format geliefert oder sind dem ERP-System überhaupt nicht zu entlocken. Hierfür wurden in diesem Kapitel bereits einige Lösungsansätze aufgezeigt. Weitere Lösungsansätze insbesondere im Blick auf den Export der Daten in einen Katalogstandard werden dann noch im Kapitel 6 (siehe Seite 90) folgen.

5 Einbinden der Preisdaten in viaMEDICI EPIM

Nachdem die aus dem ERP-System von SAP stammenden Preis- und Bestelldaten nun in artikel-, kunden- und sprachbezogene XML-Dateien nach dem in Kapitel 3 (siehe Seite 40) beschriebenen XML Schema erstellt und in nach der Artikelnummer benannten Unterordnern abgelegt wurden, stellte sich mit der Ansicht der Daten aus dem System viaMEDICI EPIM heraus die nächste Aufgabe.

5.1 Ansicht in viaMEDICI EPIM

Für die Erstellung einer Ansichtsmöglichkeit der in XML-Kontainern vom Typ PRICEINFO enthaltenen Daten wurde in erster Linie auf bereits bestehende Konzepte und Resourcen innerhalb des Software-Systems viaMEDICI EPIM zurückgegriffen. Im Groben sollte mittels JSPs ein Browser-Fenster gestaltet werden, dass in einem Frame den Inhalt des ausgewählten Artikel-Verzeichnisses und in einem zweiten Frame dann je nach Auswahl den via XSLT in HTML transformierten Inhalt anzeigt.

5.1.1 Grundlegendes Konzept

Ansichten innerhalb von viaMEDICI EPIM werden immer nach dem gleichen Schema realisiert:
Zentrales Element einer EPIM-Ansicht ist eine *main.jsp*, die für den Aufbau des Browserfensters sorgt. In ihr wird das HTML-Frameset generiert, dass für einen strukturierte und dem Layout des Systems angepassten Aufbau des Fensters sorgt. In der Regel sind dort drei bis vier Frames zu finden, die vom Aufbau her an klassische Office-Anwendungen erinnern. Die Anzahl und letztendliche Anordnung der Frames wird in Layoutvorlagen gespeichert, die so recht einfach je nach Bedarf in die *main.jsp* geladen werden können. Eine Abbildung mit dem grundsätzlichen Aufbau einer Ansicht in viaMEDICI EPIM befindet sich auf Seite 85.
In den oberen beiden Frames können Navigationselemente wie Menüleisten und Symbolleisten geladen werden. Sowohl für Menüleisten als auch Symbolleisten gibt es bereits vorgefertigte Vorlagen die den meisten Anforderungen genügen. Sie können ebenfalls in der *main.jsp* festgelegt werden.

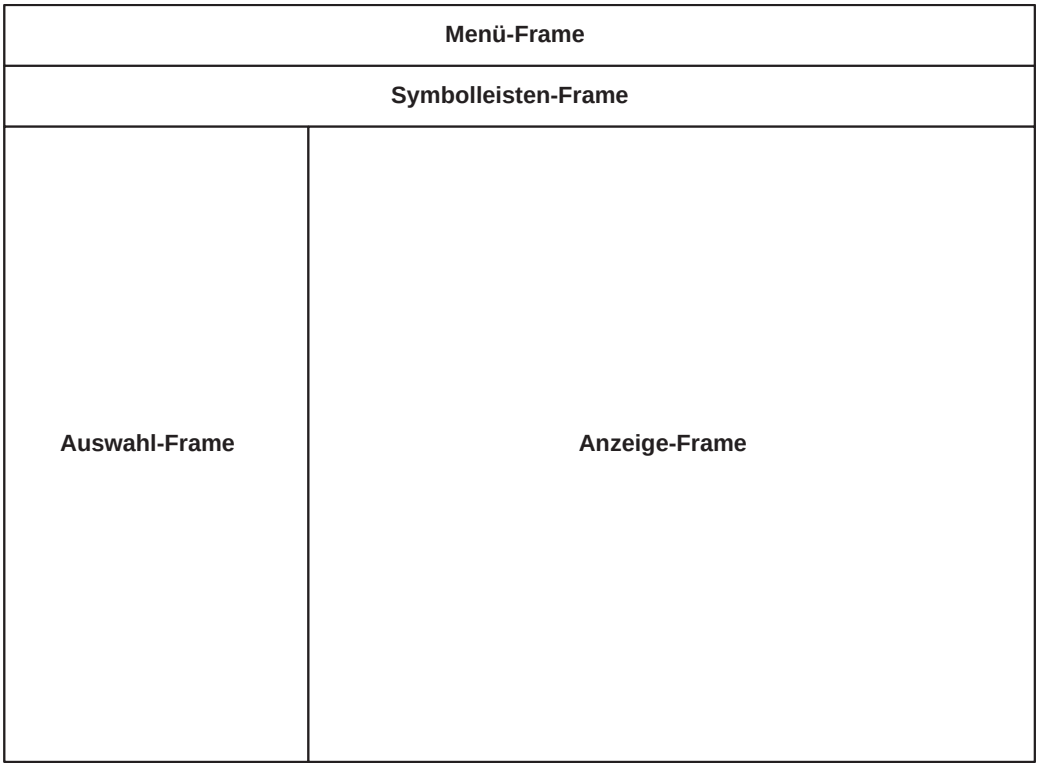


Abbildung 5.1: Grundsätzlicher Aufbau eines Framesets in viaMEDICI EPIM

Im Auswahl-Frame wird dann meist die Möglichkeit zur Navigation innerhalb der Daten angeboten. Welche Datei hier genau zu laden ist, wird wie auch im Anzeige-Frame über Resource-Files festgelegt. So ist es recht einfach möglich, je nach Kunden unterschiedliche Inhalte zu laden. Je nach getroffener Auswahl im Auswahl-Frame wird dann die entsprechende Ansicht, z.B. Formulare zur Datenpflege, in den Anzeige-Frame geladen. Sprachabhängige Texte werden innerhalb aller Seiten ebenfalls über Resource-Dateien realisiert.

5.1.2 Realisierung

Die Realisierung der Ansicht selbst teilte sich in zwei Arbeitsgebiete:

- die Erstellung einer XSL-Datei für die Transformation der in XML vorliegenden Daten in HTML
- die Erstellung bzw. Anpassung der für die Anzeige notwendigen JSP-Dateien und Java-Klassen

Die XSL-Transformation

Ziel der Erstellung einer XSL-Datei zur Transformation der XML-Daten in HTML sollte es sein, das die Daten in einem für Menschen leichter lesbaren Format am Bildschirm betrachtet werden können.
Dazu wurde auf der Grundlage einer nach dem XML Schema PRICEINFO generierten XML-Datei eine XSL-Datei geschrieben, die die Daten aus der XML-Datei in eine HTML-Datei mit einem Tabellengerüst transformiert. In diesem Tabellengerüst werden nicht nur die Daten aus der XML-Datei angezeigt sondern auch passende Beschreibungen, die bei der Identifikation der Daten helfen sollen. Diese Beschreibungen, die innerhalb der XSL-Datei festgelegt sind, erfordern bei mehrsprachigen Ansichten eine Übersetzung. Aus diesem Grund wurde festgelegt, dass die XSL-Dateien sprachabhängig sein müssen und ein Sprachkürzel mit in den Namen aufgenommen werden muss.

Die Java-Programme

In Anlehnung an das auf Seite 84 vorgestellte Grundkonzept zur Anzeige von Informationen im Software-System viaMEDICI EPIM wurden folgende JSP-Seiten erstellt:

- ***PriceinfoMain.jsp***
  Diese Datei generiert das Frameset des Browserfensters. Dabei wurde in diesem konkreten Fall ein Frameset gewählt, dass nur aus drei Frames besteht. Auf den zuvor erwähnten Frame für die Menüleiste wurde verzichtet, weil die Anzeige der Preisinformationen in einem eigenständigen Browserfenster neben dem eigentlichen System geöffnet werden soll und somit auch neben der Anzeige der Informationen keine weiteren Funktionen notwendig sind.
  Als Symbolleiste wird eine bereits existierende Vorlage geladen, die lediglich einen Button zum Schließen des Browser-Fensters enthält. Weitere Funktionalitäten sind auch hier nicht notwendig.
  Beim Laden der Datei wird weiter im Auswahl-Frame die Navigationsmöglichkeit und im Anzeige-Frame eine leere HTML-Seite mit dem Standard-Hintergrund der Anwendung geladen.

- ***PriceinfoList.jsp***
  Diese Datei enthält die Navigationsmöglichkeit und wird bereits beim Start der *PriceinfoMain.jsp* in den Auswahl-Frame geladen. Realisiert wurde die Navigation innerhalb des Artikel-Ordners mittels einem bereits existierenden Java-Applet, dass in die via JSP erzeugte HTML-Seite geladen wird.
  Das Applet namens *Hierarchy Applet* liest dabei den Inhalt des übergebenen Verzeichnisses und stellt den Inhalt in einem Navigationsfenster dar. Mit diesem Applet lässen sich weiter Menüs festlegen, die sich bei einem

Rechtsklick auf den jeweiligen Inhalt des Verzeichnisses öffenen und so weitere Aktionen erlauben. In diesem konkreten Fall wurde lediglich ein Menüeintrag „Auswahl“ festgelegt, der bei einem Klick darauf die Transformation in der Datei *PriceinfoAction.jsp* im Anzeige-Frame startet.

- ***PriceinfoEdit.jsp***
  Diese Datei wird ebenfalls direkt beim Laden der Datei *PriceinfoMain.jsp* in den Anzeige-Frame geladen. Dabei wird lediglich der Standard-Hintergrund der Anwendung geladen und angezeigt.

- ***PriceinfoAction.jsp***
  Diese Datei wird beim Klicken des Menüeintrages „Auswahl“ einer im Navigationsfenster angezeigten XML-Datei geöffnet und löst somit den Transformationsvorgang aus.
  Für die Transformation selbst wurde eine bereits bestehende Java-Klasse namens *XSLBean* um eine Funktion erweitert und als Bean in die JSP-Datei eingebunden. Die hinzugefügte Funktion namens *transformXSLToString()* erweitert die Klasse dahingehend, dass das Ergebnis der XSL-Transformation direkt als String zurückgegeben wird. Die Transformation selbst wird mittels der Xalan-API durch die Klassen TransformerFactory und Transformer realisiert. Der folgende Codeausschnitt zeigt die Transformation:

```
1       ...
2       //ein StringWriter.Objekt wird instanziiert
3       sTargetOut = new StringWriter();
4       ...
5       //eine neue Instanz einer TransformerFactory wird
           geschaffen
6       TransformerFactory tFactory = TransformerFactory.
           newInstance();
7       ...
8       //aus dieser TransformerFactory wird ein neues
           Transform-Objekt geschaffen, das die Quelldateien
           und die Ausgabemoeglichkeit uebergeben bekommt
9       Transformer transformer =
10          tFactory.newTransformer(
11          new StreamSource("file:///"+p_sXslDirectory+
               p_sXslFilename));
12      ...
13      //Ausfuehren der Transformation
14      transformer.transform(new StreamSource(new File(
           p_sFileURL)),
15          new StreamResult(sTargetOut));
16      ...
```

Listing 5.1: Aufruf der XSL-Transformation in der Java-Klasse XSLBean

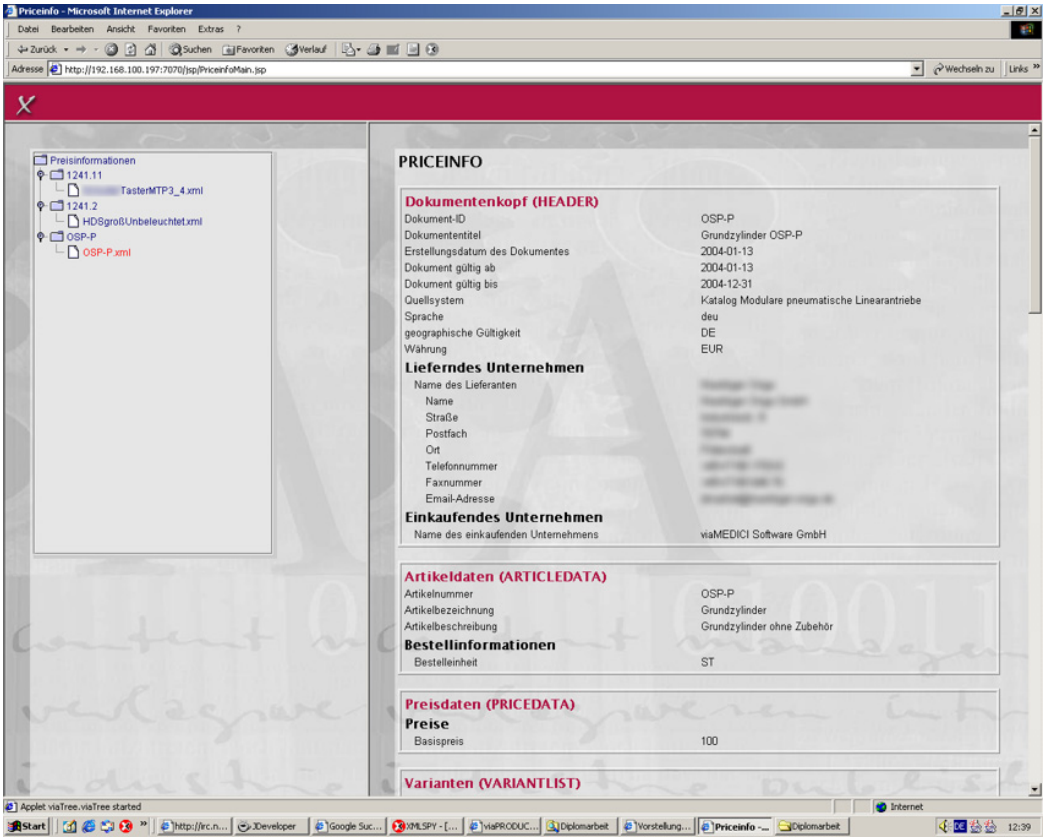


Abbildung 5.2: Screenshot vom Test der Anzeige einer PRICEINFO-XML-Datei

Mit der in die verwendete Entwicklungsumgebung Oracle JDeveloper implementierten JSP-Engine konnten die erstellten JSPs abschließend erfolgreich auf ihre Funktionalität getestet werden, wie der Screenshot des Browserfensters auf Seite 88 zeigt.

5.1.3 Integration

Zu einer Integration der angefertigten JSP-Dateien in viaMEDICI EPIM ist es aus zeitlichen Gründen nicht mehr gekommen. Damit die erstellten Ansichten in das System integriert werden können sind noch einige kleinere Arbeiten notwendig, die noch von einem systemkundigen Entwickler der viaMEDICI Software GmbH vorgenommen werden müssen:

- Der bisher noch absolut festgelegte Pfad für das Verzeichnis, in dem die Artikel-Unterverzeichnisse inklusive der XML-Dateien zu finden sind, muss noch an die Konfigurationsmöglichkeiten des Systems angepasst werden.
- In die Ansicht der technischen Informationen zu einem Artikel in viaMEDICI EPIM muss noch ein Link integriert werden, der auf das Fenster mit den

jeweiligen, dem Artikel zugehörigen Preis- und Bestellinformationen zeigt und somit eine Ansicht dieser ermöglicht.

- Die momentan ausschließlich in Deutscher Sprache vorliegende XSL-Datei für die Transformation muss noch in weitere gefragte Sprachen übersetzt werden. Diese Übersetzung wird voraussichtlich ebenfalls mit Hilfe der Java-Extensions des Xalan-J-Prozessors und einem Resource-Dokument erfolgen, was keine statische Übersetzung in mehreren XSL-Dateien notwendig macht.

5.2 Fazit

Aufgrund des bereits vorhandenen Frameworks des Systems viaMEDICI EPIM und der Tatsache, dass neben der mit einer XSL-Transformation verbundenen Anzeige keine weiteren Aktionen an diese Ansicht geknüpft waren, war es sehr einfach, eine Ansicht der XML-Dateien innerhalb des Systemes zu erstellen. Lediglich die Implementierung der erdachten und ausgearbeiteten Konzepte konnte im Rahmen dieser Arbeit nicht mehr durchgefuhrt werden.

6 Export von Preisinformationen

Nach dem Import der Daten aus dem SAP-spezifischen XML-Dokument in den erstellten XML-Datenkontainer und die Realisierung der Anzeige der Preis- und Bestellinformationen im Softwaresystem viaMEDICI EPIM bleibt nun nur noch die Beschreibung des Exports der Daten in Richtung elektronischer Marktplätze und speziell in das BMEcat-Format. Eigentlich wird in diesem Kapitel weniger der Export selbst beschrieben, weil dieser vollständig vom System-Baustein Mediator übernommen wird. Trotzdem musste überlegt werden, wie die Daten dem Mediator für den Export zugeführt werden können.

6.1 Grundlegendes Konzept

Bevor also ein Export in Richtung Katalogstandard durchgeführt werden kann, musste erst ein Konzept für die Übergabe der Daten an den Mediator entwickelt werden. Dazu muss bekannt sein, dass der Mediator mit einer Vielzahl an Informationen zurechtkommt und diese in ein Zielformat wie z.B. BMEcat übertragen kann, solange diese in einem XML-Format geliefert werden, dessen Struktur er durch eine DTD oder ein XML Schema kennt.
Durch dieses Wissen ergibt sich ganz klar, dass weitere Datenstrukturen geschaffen werden müssen, die sowohl die technischen Artikeldaten als auch die Preis- und Bestellinformationen an den Mediator weiterreichen können. Bei genauerer Überlegung und mit etwas Hintergrundinformation über den Mediator selbst, der Daten aus verschiedenen Quellen zusammenführen kann, zeichnen sich zwei grundsätzliche Methoden für die Datenweitergabe an den Mediator ab:

1. Die Daten werden so weitergegeben, wie sie von der SAP-Import-Schnittstelle erzeugt wurden.

2. Die Daten werden vor der Weitergabe an den Mediator in ein XML-Format zusammengeführt.

3. Die Daten werden nach technischen Informationen und Preis- und Bestellinformationen getrennt an den Mediator weitergegeben, der dann das Zusammenführen und die Transformation in einen Katalogstandard übernimmt.

Aus Sicht der viaMEDICI Software GmbH und des Mediators ist natürlich die zweite Möglichkeit die am besten geeignete, da diese Methode einen geringeren

Aufwand als bei Lösung 3 und eine bessere Übersichtlichkeit als bei Lösung 1 gewährleistet- Zudem ist das Interface des Mediators so ausgelegt, dass technische Informationen und Preisinformationen auf getrenntem Wege zugeführt werden können, ohne dass sich eine vorgeschaltete Logik um das Zusammenführen der Daten kümmern muss. Dadurch ist es auch einfach möglich, nur die technischen Daten oder nur die Preisinformationen zu exportieren. Aus diesen Gründen wurde die Entscheidung auch für die einfachere Variante gefällt.
Dies erfordert nun aber, dass die vielen einzelnen kunden-, artikel- und sprachabhängigen XML-Dateien mit den Preis- und Bestellinformationen zur Erstellung eines kunden- und sprachspezifischen Katalogdokumentes wieder zusammengefasst werden müssen. Dazu muss erneut eine XSL-Transformation der Daten in einen weiteren zu erstellenden XML-Kontainer erfolgen.
Ein weiteres Problem erfordert die XSL-Transformation auch noch. Wie bereits in Kapitel 4 (siehe Seite 72) beschrieben, müssen einige der aus dem SAP-System erhaltenen Daten noch in BMEcat-taugliche Formate gemappt werden. Dafür wird, wie bereits kurz beschrieben, die Java-Extension des XSLT-Prozessors Xalan-J genutzt, über die eine das Mapping durchführende Java-Klasse aufgerufen wird.

6.2 Realisierung

In diesem Unterkapitel geht es nun konkret darum, wie mittels XSL-Transformation aus vielen XML-Dateien wieder eine XML-Datei gemacht werden kann und wie das Mapping von Daten via Java-Extensions realisiert wurde. Ein erster Blick geht dabei auf das zweite erstellte XML Schema, dass das Format der für den Transport der Preis- und Bestellinformationen zum Mediator benötigten XML-Datei festlegt.

6.2.1 XML Schema für den Export der Preis- und Bestelldaten

Da für den Export der Preis- und Bestelldaten wieder alle Daten zu allen Artikeln, die einem bestimmten Kunden zugeordnet sind, benötigt werden, war es nochmals notwendig, ein XML Schema zu entwerfen, dass diesen Anforderungen gerecht wird. Da der Export der Daten in Richtung BMEcat stattfindet, wurde das Schema so entworfen, dass es zusätzlich den Anforderungen an Struktur und Datenformaten der Katalogstandards genügt. Ein Blick auf die Struktur auf Seite 92 zeigt die Ähnlichkeit zu den relevanten Teilen des BMEcat-Standards. Die XSD-Datei trägt den Namen *pricinfo_to_bmecat.xsd*.
Aufgrund dieser Ähnlichkeit wird statt einer näheren Beschreibung der hier verwendeten Elemente auf die Spezifikation des BMEcat-Standards [BMEcat1] verwiesen.

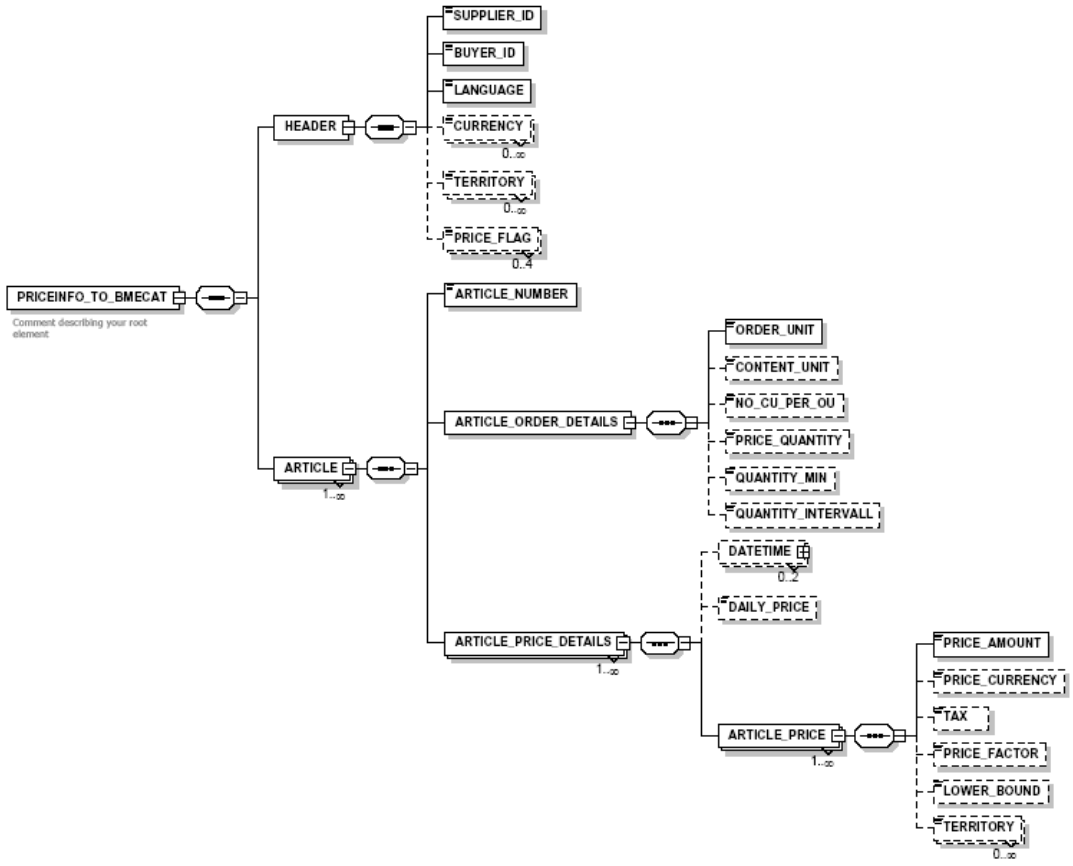


Abbildung 6.1: Struktur der XML Schema-Datei *pricinfo_to_bmecat.xsd*

6.2.2 Mapping von Daten

Da nicht alle aus dem SAP-Sytem erhaltenen Daten aus Kompatibilitätsgründen direkt in den Katalogstandard übernommen werden können, wurde eine Möglichkeit zum Mappen von Daten eines Formates auf ein anderes notwendig. Da XSL direkt nicht die Möglichkeit dazu bietet, wurde an dieser Stelle wieder auf die bereits im Kapitel 4 beschriebenen Java-Extensions des Xalan-J XSLT-Prozessors zurückgegriffen. Genauer wurde eine Java-Klasse geschaffen, der ein Datenwert übergeben wird und aus einer Resource-Datei einen passenden Gegenwert sucht und zurückgibt. Dies wurde deshalb so realisiert, weil es von Seiten der viaMEDICI Software GmbH möglich sein sollte, eine Resource-Datei zur Festlegung der Werte direkt an den Kunden zu geben. Von daher war klar, dass die Resource-Datei selbst so einfach wie möglich aufgebaut sein soll.
Um diese Aufgabe zu meistern, wurde die Java-Klasse *Mapping.java* in zwei Methoden unterteilt:

- ***loadMappingResource(String sElement)***

Die Methode *loadMappingResource(String sElement)* hat die Aufgabe, das Klassen-Objekt *props* vom Typ Properties mit den Inhalten einer Resource-Datei zu füllen. Zum Ermitteln des Namens der Resource-Datei wird der Methode die Variable *sElement* vom Typ String übergeben. Der folgende Quelltext zeigt die gesamte Funtktion:

```
1  ...
2  private static void loadMappingResource(String sElement){
3    Mapping map = new Mapping();
4    String path = "resource\\";
5    //Namen der Resource-Datei zusammenbauen
6    String resFileName = "priceinfo_"+sElement+".mapping";
7    InputStream propsFile;
8
9    propsFile = map.getClass().getResourceAsStream(path+
         resFileName);
10
11   try {
12     //Schreiben in die Klassenvariable props
13     props.load(propsFile);
14   }
15   catch (FileNotFoundException fnfe){
16     System.out.println("fnfe:"+fnfe);
17   }
18   catch (IOException ioe){
19     System.out.println("ioe:"+ioe);
20   }
21   catch (NullPointerException npe){
22     System.out.println("npe:"+npe);
23   }
24 }
25 ...
```

Listing 6.1: Methode *loadMappingResource(String sElement)* der Klasse *Mapping*

Eine Resource-Datei muss dabei folgendermaßen aufgebaut sein:

```
1 DE  = deu
2 EN  = eng
3 ...
```

Listing 6.2: Beispiel für eine Resource-Datei für das Mappen von Sprach-Daten von EPIM-intern nach BMEcat

Hier zeigt sich auch schön der einfache Aufbau einer Resource-Datei, der

es ermöglicht, die zu mappenden Werte ohne weitere Fachkenntnisse direkt vom Kunden eintragen zu lassen.

- ***getMappedString(String sKey, String sElement)***
  Die zweite Methode *getMappedString(String sKey, String sElement)* ruft die Methode *loadMappingResource(String sElement)* auf, die das Klassenobjekt *props* mit den Daten aus der Resource-Datei füllt. Da sie direkt aus dem XSL-Dokument aufgerufen werden kann, hat sie die Aufgabe, den passenden Gegenwert zu dem über die Variable *sKey* übergebenen Wert im klassenweiten Properties-Objekt *props* zu finden und den Wert als String in die aufrufende XSL-Datei zurückzugeben. Die ebenfalls übergebene Variable *sElement* wird zur Generierung des Namens der Resource-Datei direkt an die Methode *loadMappingResource(String sElement)* weitergereicht. Der folgende Quelltext-Ausschnitt zeigt die ganze Methode:

```
1  ...
2  public static String getMappedString(String sKey, String
       sElement){
3    String sResult = "";
4
5    //F"ullen des props-Objektes
6    loadMappingResource(sElement);
7
8    //Gegenwert zu sKey in props suchen und zur"uckgeben
9    sResult = props.getProperty(sKey);
10
11   return sResult;
12 }
13 ...
```

Listing 6.3: Methode *getMappedString(String sKey, String sElement)* der Klasse *Mapping*

Über diese recht einfache Java-Klasse ist es nun möglich, die Quelldaten den Anforderungen des Zielformates anzupassen. Dies war im Hinblick auf den BMEcat-Standard vor allem bei den Verpackungseinheiten (ISO zu UN/ECE) und der Sprache (internes Format zu ISO) notwendig.

6.2.3 Transformationsprozess

Aufgabe des nun folgend beschriebenen Transformationsprozesses ist es nun, alle relevanten PRICEINFO-XML-Dateien eines Kunden aus den verschiedenen Artikel-Unterordnern zusammenzusuchen und sie in ihrer Gesamtheit in das

Transport-XML-Format, das mit der Schema-Datei *pricinfo_to_bmecat.xsd* festgelegt wurde, zu transformieren.
Dazu wurde eine weitere Java-Klasse namens *PriceinfoToBMEcat* entwickelt, die diese Aufgabe übernehmen soll. Da auch hier eine Implementierung in das Software-System viaMEDICI EPIM nicht durchgeführt werden konnte, wurde die Klasse mit einer *main()*-Methode versehen, die ein eigenständiges Ausführen der Klasse erlaubt.
Die *main()*-Methode nimmt dazu einen Vector mit Artikelnummern und generiert daraus die Dateinamen der PRICEINFO-XML-Dateien, die für die Transformation in das Transport-XML-Format benötigt werden. Dazu liest sie dann alle erforderlichen Dateien ein und fügt sie zu einem großen XML-Dokument zusammen, dass für die Dauer des Transformationsprozesses temporär abgespeichert wird. Diese große, temporäre XML-Datei wird dann mittels der XSL-Datei *PRICEINFO_TO_BMECAT.xsl* in das Transport-XML-Format transfomiert.

```
1   ...
2   public static void main(String[] args) {
3     PriceinfoToBmecat priceinfoToBmecat = new PriceinfoToBmecat()
        ;
4     //Parameter f"ur den Test der Klasse
5     String directory = "C:\\work\\Entwuerfe\\";
6     String filename = "articles.txt";
7     Vector articles = new Vector();
8     String buyer = "800";
9     String language = "DE";
10    String xslFilename = "PRICEINFO_TO_BMECAT.xsl";
11    FileWriter xmlFileWriter;
12    FileWriter tmpFileWriter;
13    String xmlDecalaration = "<?xml version=\"1.0\" encoding=\"
        UTF-8\" standalone=\"yes\"?>";
14    String tmpFileName = directory+"tmp_"+buyer+".xml";
15    File tmpFile = new File(tmpFileName);
16
17    try {
18      //Initialisieren der tempor"aren XML-Datei
19      tmpFileWriter = new FileWriter(tmpFile);
20
21      tmpFileWriter.write(xmlDecalaration);
22      tmpFileWriter.write("<PRICEINFOTMP>");
23
24      //Auslesen der Artikelnummern
25      articles = priceinfoToBmecat.createArticleList(directory+
          filename);
26      for (int i = 0; i < articles.size(); i++){
27        String xmlDirectoryname = articles.get(i)+"\\";
28        String xmlFilename = articles.get(i)+"_"+buyer+"_"+
            language+".xml";
```

```
29          int c;
30          StringBuffer tmpStringBuffer = new StringBuffer();
31
32          //Lesen und Schreiben der einzelnen PRICEINFO-XML-Dateien
33          try {
34            FileReader f = new FileReader(new File(directory+
                xmlDirectoryname+xmlFilename));
35            while ((c = f.read()) != -1) {
36              tmpStringBuffer.append((char)c);
37            }
38            f.close();
39            String tmpBufferString = tmpStringBuffer.toString();
40
41            if (tmpBufferString.indexOf("<?xml version=\"1.0\"
                encoding=\"UTF-8\"?>") == 0){
42              int tmpXMLDecLength = "<?xml version=\"1.0\" encoding
                  =\"UTF-8\"?>".length();
43              tmpBufferString = tmpBufferString.substring(
                  tmpXMLDecLength);
44            }
45            tmpFileWriter.write(tmpBufferString);
46          }
47          catch(FileNotFoundException fexp){
48            System.out.println(fexp);
49          }
50          catch(IOException ioexp){
51            System.out.println(ioexp);
52          }
53        }
54        //Abschlie"sen der tempor"aren XML-Datei
55        tmpFileWriter.write("</PRICEINFOTMP>");
56        tmpFileWriter.close();
57      }
58      catch(FileNotFoundException fexp){
59        System.out.println(fexp);
60      }
61      catch(IOException ioexp){
62        System.out.println(ioexp);
63      }
64
65      //Transformation der tempor"aren XML-Datei in das Transport-
          XML-Format
66      String result = priceinfoToBmecat.transform(tmpFileName,
          directory+xslFilename);
67      //L"oschen der tempor"aren XML-Datei
68      tmpFile.delete();
69
70      //Schreiben der Transport-XML-Datei
71      try {
72        xmlFileWriter = new FileWriter(directory+buyer+"
            _PRICEINFO_TO_BMECAT.xml");
```

```
73        xmlFileWriter.write(result);
74        xmlFileWriter.close();
75      } catch (IOException e) {
76        System.out.println("Fehler beim Erstellen der Datei");
77      }
78    }
79    ...
```

Listing 6.4: *main()*-Methode der Klasse *PriceinfoToBMEcat*

Neben der main-Methode enthält die Klasse noch zwei weitere Methoden, an die untergeordnete Aufgabenbereiche ausgegliedert wurden:

- ***createArticleList(String filename))***
  Die Methode *createArticleList(String filename)* wurde eigentlich nur für den Test der Transformation erstellt. Sie hat die Aufgabe, aus einer Textdatei eine Reihe von Artikelnummern zu lesen, die dann für die Erstellung der Dateinamen der zu transformierenden PRICEINFO-XML-Dateien herangezogen werden. Dazu werden die Artikelnummern aus der Textdatei gelesen und in einem Vector-Objekt abgelegt, dass nach Abschluss der Lese- und Schreiboperationen zurückgegeben wird.

- ***transform(String xmlFile, String xslFile)***
  Die zweite Methode *transform(String xmlFile, String xslFile)* übernimmt dann die eigentliche Transformation der temporären XML-Datei in das Transport-XML-Format. Dazu werden Objekte der gleichen Java-Klassen wie bei der Transformation in Kapitel 5 auf Seite 87 angewendet.

6.3 Integration

Wie bereits angedeutet, konnte auch hier aus bekannten Gründen keine Integration in das Software-System viaMEDICI EPIM stattfinden. Aus diesem Grund auch hier eine kurze Auflistung der Punkte, die bei einer Integration beachtet werden müssen:

- Die in der *main()*-Methode der Klasse *PriceinfoToBMEcat.java* fest deklarierten Variablen für die Sprache und den Kunden und die Liste der Artikelnummern müssen auf anderem Wege übergeben werden. Denkbar wäre ein Aufruf aus viaMEDICI EPIM heraus oder über eine graphische Benutzeroberfläche, die eine entsprechende Eingabe ermöglicht.

- Die Zusammensetzung der Dateinamen und die Pfade müssen den systemweit üblichen Namen und Pfaden angepasst werden. Evtl. sollte auch eine leichtere Konfigurierbarkeit dieser z.B. über die hinter viaMEDICI EPIM befindliche Datenbank implementiert werden.

6.4 Fazit

In diesem Kapitel wurde eine weitere XSL-Transformation beschrieben, die viele PRICEINFO-XML-Dateien zu einem weiteren XML-Kontainer für den Transport der Preis- und Bestelldaten zusammenfasst. Dieser XML-Kontainer wird dann zusammen mit einem weiteren XML-Dokument mit den technischen Informationen an den Mediator übergeben, der den eigentlichen Export in den Katalogstandard BMEcat durchführt.
Die eigentliche Schwierigkeit dabei war es zum einen, die vielen PRICEINFO-XML-Dateien zu einem größeren, in einem Zug transformierbaren XML-Format zusammen zu bauen, und zum anderen, die verschiedenen Quell-Datenformate in die vom Katalogstandard gefordeten Formate zu mappen.

7 Zusammenfassung und Ausblick

In diesem letzten Kapitel sollen die Ergebnisse der Diplomarbeit zusammengefasst werden. Darüberhinaus soll ein kurzer Ausblick zeigen, was in Zukunft auf die erstellte XML-Datenstruktur und die viaMEDICI Software GmbH zukommen wird.

7.1 Zusammenfassung

Im Rahmen dieser Diplomarbeit wurden die betriebswirtschaftlichen Grundlagen der Preisdifferenzierung aufgezeigt. Weiter wurde analysiert, wie Preisinformationen in ERP-Systemen als Quelle und in Katalogstandards als Ziel dargestellt werden können. Hierbei war es besonders schwierig an Informationen über die Darstellung von Preisinformationen in ERP-Systemen zu kommen. Einzig SAP unterhält eine umfangreiche Dokumentation im Internet, die auch frei zugänglich ist.
Als Resultat dieser Betrachtungen wurde dann ein XML-Datenmodell zur Aufnahme von Preis- und Bestellinformationen erstellt. Dieses XML-Struktur soll die zuvor genannten Informationen im Softwaresystem viaMEDICI EPIM visualisieren und einen einfacheren Export der Daten in einen Katalogstandard ermöglichen. Da als Zielformat klar der Katalogstandard BMEcat in der Version 1.2 ins Auge gefasst wurde, orientiert sich die erstellte Struktur sehr am Aufbau des Standards.
Der weitere, ausschließlich praktische Teil der Arbeit zeigt auf, wie Daten aus dem ERP-System von SAP in die erstellte XML-Datenstruktur importiert werden können, diese dann innerhalb des Systems angezeigt werden und wie ein Export in einen Katalogstandard funktionieren kann. Dabei wurden verschiedene Konzepte und Lösungen beschrieben.
Eigentlich sollte abschließend auch eine Implementierung der erdachten Konzepte und erstellten Lösungen in das Softwaresystem viaMEDICI EPIM stattfinden. Aufgrund einiger Terminverschiebungen war dies leider nicht mehr möglich und bleibt nun den Entwicklern der viaMEDICI Software GmbH überlassen. So konnte z.B. die Schnittstelle zu SAP nicht rechtzeitig begonnen werden, weil die benötigten SAP-Fachkräfte terminlich nicht früher verfügbar waren. Auch haben einige Kundenwünsche in den letzten Wochen der Diplomarbeit viele Kräfte der Firma viaMEDICI Software GmbH gebunden, so dass für die Integration kein systemkundiger Entwickler frei war, der eine dringend notendige Unterstützung hätte

geben können.
Insgesamt war die Diplomarbeit meinerseits davon geprägt, dass ich mich in einige für mich neue Techniken einarbeiten musste. So waren nicht wirklich viele Erfahrungen mit ERP-Systemen und Katalogstandards vorhanden. Auch die Kenntnisse in XML, XSL und Java mussten für diese Arbeit etwas erweitert werden. Insgesamt kann die Arbeit aber aus genau diesem Grund als interessant und lehrreich eingestuft werden und spiegelt so auch die an einen IT-Spezialisten gestellten Anforderungen im weiteren Berufsleben wieder.

7.2 Ausblick

Mit dem XML Schema *priceinfo.xsd* konnte ein Datenkontainer geschaffen werden, der den momentanen Anforderungen der Darstellung von Preis- und Bestellinformationen genügt. Es kann aber davon ausgegangen werden, dass sich diese Anforderungen in nächster Zeit verändern können. Ein Anzeichen dafür ist die in den Startlöchern befindliche Version 2.0 des Katalogstandards BMEcat, die einige Änderungen mit sich bringen wird. Ob das erstellte Datenformat auch den Anforderungen gewachsen ist, die die neue Version des Standards mit sich bringt, wird sich dann frühestens nach dessen Veröffentlichung Ende des ersten Quartals 2004 zeigen. Im Vorfeld der Veröffentlichung waren leider keine näheren Informationen zu den Vernderungen erhältlich.
Darüberhinaus stehen sicherlich noch im Hinblick auf die Abbildung von Produktvariationen und Berechnungsformeln Anpassungen oder gar weitreichende Veränderungen bevor, weil diese Möglichkeiten des PRICINFO-XML-Kontainers mit den vom SAP-System erhaltenen Daten nicht auf seine Praxistauglichkeit getestet werden konnten. Hinzu kommt noch, dass bisher auch keine Katalogstandards auf dem Markt sind, die diese Daten im Bereich des Exports aufnehmen könnten.
Auch von Seiten der Kunden der viaMEDICI Software GmbH und den Herstellern von ERP-Systemen werden sich die Anforderungen sicherlich ändern, die das erstellte XML-Datenmodell zu erfüllen hat. Es kann deshalb davon gesprochen werden, dass besagtem XML-Datenformat mit dieser Diplomarbeit „Leben“ eingehaucht wurde.
Es sollte an dieser Stelle auch genannt werden, dass es Aufgrund einiger zeitlicher Verschiebungen bei dieser Arbeit vorgelagerten Projekten, eine Integration der beschriebenen Konzepte und Quellcodes in das Softwaresystem viaMEDICI EPIM im Rahmen dieser Arbeit nicht mehr durchgeführt werden konnte. Hier sind nun die erfahrenen Entwickler der Firma viaMEDICI Software GmbH gefragt, die den auf einer CD abgelieferten Code in das Softwaresystem anpassen und integrieren müssen.

8 Anhang

Literaturverzeichnis

[Helmedag1] *Helmedag, Prof. Dr. Fritz*: Preisdifferenzierung
(http://www.tu-chemnitz.de/wirtschaft/vwl2/downloads/paper/helmedag/Preisdifferenzierung.pdf)

[Skiera1] *Skiera, Bernd*: Preispolitik und Electronic Commerce - Preisdifferenzierung im Internet
(http://www.ecommerce.wiwi.uni-frankfurt.de/skiera/publications/2000_preisdifferenzierung.pdf)

[LeukelSchmitz1] *Leukel, Jörg und Schmitz, Volker*: Modellierung von Preisinformationen in elektronischen Produktkatalogen
(http://www.bli.uni-essen.de/english/publications/2002_Modellierung_LeukelSchmitz.pdf)

[SAP1] *SAP*: SAP Help Portal
(http://help.sap.com/)

[SAP2] *SAP*: Konditionen und Preisfindung (SD-BF-PR)
(http://help.sap.com/sapdocu/core/47x200/HELPDATA/DE/dd/56165a545a11d1a7020000e829fd11/frameset.htm)

[SAP3] *SAP*: Interface Repository
(http://ifr.sap.com/catalog/query.asp)

[SAP4] *SAP*: Interface Repository: iDoc-Type COND_A01
(http://ifr.sap.com/catalog/query.asp?namespace=urn:sap-com:ifr:LO:470&type=idoc&name=COND_A01)

[JDE1] *JDEdwards*: Erweiterte Bepreisung
(Auszug aus dem Handbuch zu JDEdwards Version A7.3 vom Juni 1996)

[BMEcat1] *Renner, Thomas; Schmitz, Volker; Kelkar, Oliver; Pastoors, Thorsten und Hümpel, Klaus*: Spezifikation BMEcat Version 1.2
(http://www.bmecat.de)

[eClass1] *eCl@ss e. V.*: eCl@ss Release 5.0 Standard für Materialklassifikation und Warengruppen
(http://www.eclass.de)

[cXML1] *cXML*: Homepage cXML Version 1.2
(http://www.cxml.org)

[XMLStandards1] *Schultz; Carsten; Mainusch, Martin und Belghith, Benjamin*: deutschsprachige Zusammenfassung von cXML, xCBL und OAGIS
(http://www.wi-inf.uni-essen.de/ ifs/lehre/curriculum/is4/1-II-A.pdf)

[Ariba1] *Ariba Inc.*: Homepage der Ariba Inc., Hersteller von eBusiness-Lösungen
(http://www.ariba.com)

[OAGIS1] *Open Applications Group*: Homepage der Open Applications Group, Anbieter und Entwickler des B2B-Austauschstandards OAGIS
(http://www.openapplications.org)

[W3C1] *W3C*: World Wide Web Consortium
(http://www.w3c.org; http://www.w3c.de)

[XML1] *Kränzler, Christine*: XML/XSL - für Buch und Web
(2002; Markt&Technik-Verlag; ISBN: 3-8272-6339-5)

[XMLSchema1] *XML Schema - W3C-Empfehlung*: edition W3C.de - deutsche Übersetzungen der W3C-Empfehlungen
(http://www.edition-w3c.de)

[XMLSchema2] *van der Vlist, Eric*: XML Schema
(2003; OReilly Verlag; ISBN: 3-89721-345-1)

[UNECE1] *UNECE*: UNECE Recommendation N20: Codes for Units of Measure Used in International Trade
(http://www.unece.org/cefact/rec/rec20en.htm)

[XML.Apache] *xml.apache.org*: Homepage des Apache XML Projektes
(http://xml.apache.org)

[XSLT1] *XSLT 1.0 deutsch*: w3c-Empfehlung XSLT 1.0 in deutsch
(http://klute-thiemann.de/w3c-de/REC-xslt-20020318/)

[XSLT2] *XSLT 1.0 original*: w3c-Empfehlung XSLT 1.0 - das englische Original
(http://klute-thiemann.de/w3c-de/REC-xslt-20020318/)

[XALAN-REDIRECT] *Xalan-J Redirect-Extension*: Beschreibung der Redirect-Extension des XSLT-Prozessors Xalan-J
(http://xml.apache.org/xalan-j/extensionslib.html#redirect)

[XALAN-JAVA] *Xalan-J Java-Extension*: Beispiele fr die Java-Extension des XSLT-Prozessors Xalan-J
(http://www.cs.fh-aargau.ch/ muellerr/xml/)

[XALAN-INSTALL] *Xalan-J Installation*: Dokumentation der Installation des Xalan-Prozessors auf einem Rechner
(http://xml.apache.org/xalan-j/getstarted.html)

[SAXON] *Saxon XSLT-Prozessor*: Homepage des Saxon XSLT-Prozessors
(http://saxon.sourceforge.net/)

Abbildungsverzeichnis

Tabellenverzeichnis

Tabellen

Datenstruktur SAP E1KOMG

Im folgenden ist die detaillierte Datenstruktur des Elementes E1KOMG des SAP-iDoc-Typs COND_A1 abgebildet, die im Kapitel 2 auf Seite 18 und 21 erwähnt wird.

Feld	**Bemerkung**
KVEWE	Verwendung der Konditionstabelle
KOTABNR	Konditionstabelle
KAPPL	Anwendung
KSCHL	Konditionsart
VAKEY	Komprimierter Schlüssel der Konditionstabelle
VKORG	Verkaufsorganisation
VTWEG	Vertriebsweg
SPART	Division, Bereich
KUNNR	Kundenummer
KDGRP	Kundengruppe
PLTYP	Preislistentyp
KONDA	Preisgruppe (Kunde)
KONDM	Materialpreisgruppe
WAERK	Währung des SD-Dokumentes
MATNR	Materialnummer
BWTAR	Bewertungsart
CHARG	Charge
PRODH	Produkthierarchie
MEINS	Basis-Maßeinheit
BONUS	Bonus-Gruppe
EBONU	Abrechnungsgruppe 1 (Einkauf)
PROVG	Kommissionsgruppe
ALAND	Herkunftsland
WKREG	Region des Werkes
WKCOU	Bezirk des Werkes
WKCTY	Stadt des Werkes
LLAND	Land, in das geliefert werden soll
REGIO	Region (Bundesland, Bezirk, Provinz)
COUNC	Code des Bezirks
CITYC	Code der Stadt
TAXM1	Steuerklasse Material 1
TAXM2	Steuerklasse Material 2

Feld	**Bemerkung**
TAXM3	Steuerklasse Material 3
TAXM4	Steuerklasse Material 4
TAXM5	Steuerklasse Material 5
TAXM6	Steuerklasse Material 6
TAXM7	Steuerklasse Material 7
TAXM8	Steuerklasse Material 8
TAXM9	Steuerklasse Material 9
TAXK1	Steuerklasse Kunde 1
TAXK2	Steuerklasse Kunde 2
TAXK3	Steuerklasse Kunde 3
TAXK4	Steuerklasse Kunde 4
TAXK5	Steuerklasse Kunde 5
TAXK6	Steuerklasse Kunde 6
TAXK7	Steuerklasse Kunde 7
TAXK8	Steuerklasse Kunde 8
TAXK9	Steuerklasse Kunde 9
LIFNR	Kontonummer des Lieferanten
MATKL	Materialklasse
EKORG	Einkaufsorganisation
ESOKZ	Einkausinformationskategorie
WERKS	Werk
RESWK	lieferndes Werk bei Lagertransportbestellung
KOLIF	vorheriger Lieferant
LTSNR	Teilbereich des Lieferanten
WGLIF	Materialklasse des Lieferanten
MWSKZ	Umsatzsteuer-Code
WERKV	weiterverkaufender Betrieb
WAGRP	Materialgruppe (Einkauf und Lager)
VRKME	Verkaufseinheit
EAN11	internationale Aktikelnummer
EANNR	Europäische Artikelnummer (EAN) - veraltet
AUART	Auftragsart
MEEIN	Maßeinheit eines eingekauften Artikels/Materials
INFNR	Nummer der Einkausinformation
EVRTN	Einkaufsbelegsnummer
EVRTP	Position des Einkaufsbeleges
INCO1	Incoterms (internat. Handelsbestimmungen) Teil 1
INCO2	Incoterms (internat. Handelsbestimmungen) Teil 2
BURKS	Code des Unternehmens
MTART	Materialart

Feld	**Bemerkung**
LIFRE	unterschiedlicher Rechnungssteller
EKKOL	Gruppe von Konditionen, die das Material mit dem Lieferanten verbindet
EKKOA	Gruppe von Konditionen, die das Materiel mit dem weiteren Lieferanten verbindet
BSTME	Bestelleinheit
WGHIE	Materialgruppenhierarchie
TAXIM	Steuerkennzeichen für das Material
TAXIK	Steuerkennzeichen für Kontierung
TAXIW	Steuerkennzeichen für das Werk
TAXIL	Steuerkennzeichen für den Import
TAXIR	Steuerkennzeichen für die Region
TXJDC	Code für die Zuständigkeit der Steuerberechnung
FKART	Rechnungsart
VKORGAU	Verkaufsorganisation des Kundenauftrages
HIENR	Knoten einer Kundenhierarchie
VARCOND	abweichende Kondition
LAND1	Schlüssel des Landes
ZTERM	Schlüssel der Zahlungsbedingungen
GZOLX	bevorzugte Zone
VBELN	SD-Dokumentennummer
POSNR	Position im SD-Dokument
UPMAT	Preisreferenz für Material der Hauptposition
UKONM	Material-Preisgruppe der Hauptposition
ANZSN	Anzahl der Seriennummern
AUART_SD	Art des Verkaufsbeleges
PRODH1	Datenelement ID_PRODH1
PRODH2	Datenelement ID_PRODH2
PRODH3	Datenelement ID_PRODH3
BZIRK	Verkaufsbezirk
BRSCH	Branchenschlüssel
VKBUR	Verkaufsbüro
PRCTR	Sparte
LHIENR	Lieferantennummer der Lieferantenhierarchie
KDKGR	Kundenattribut für die Konditionsgruppe
BSTYP	Kategorie des Einkaufsbeleges
BSART	Art des Auftrages (Einkauf)
EKGRP	Einkaufsgruppe
AKTNR	Promotion/Aktion
SRVPOS	Servicenummer

Feld	**Bemerkung**
PSTYP	Kategorie der Position
HLAND	lieferndes Land
AUSFU	Exporteur im Aussenhandel
HERKL	Herkunftsland
VERLD	Land, dass die Ware versendet
STAWN	Erzeugniscode / Import Codenummer im Außenhandel
CASNR	CAS-nr. für pharmazeutische Produkte
EXPRF	Export-/Importprozedur im Außenhandel
COKON	Kundenkontingent bei Importabwicklung im Aussenhandel
COPHA	Code für pharmazeutische Produkte
COADI	Anti-Dumping-Code
HERSE	Herstellernummer für Außenhandel
KTNUM	Kontingent für Importabwicklung
PLNUM	Kontingent für Importabwicklung
PREFA	Präferenztyp für den Außenhandel
EILGR	Ländergruppen bei Importabwicklungen im Außenhandel

Tabelle 8.1: SAP: Datenstruktur der Filterinformationen E1KOMG

Listings

priceinfo_base.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="
    unqualified">
  <!-- Include-Bereich -->
  <xsd:include schemaLocation="countries.xsd"/>
  <xsd:include schemaLocation="currencies.xsd"/>
  <!-- Typdefinitionen -->
  <!-- Einfache Typen -->
  <xsd:simpleType name="dtBOOLEAN">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[Ff][Aa][Ll][Ss][Ee]|[Tt][Rr][Uu][Ee]"
          />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="dtDATETYPE">
    <xsd:restriction base="xsd:date"/>
  </xsd:simpleType>
  <xsd:simpleType name="dtTIMETYPE">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="(0[0-9]|1[0-9]|2[0-3])(:[0-5][0-9])
          (:[0-5][0-9](\.[0-9]{1,}){0,1}){0,1}"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="dtTIMEZONETYPE">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="([+-]([0-1][0-9]|2[0-3])(:[0-5][0-9]))|
          Z"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="typePUBLIC_KEY">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="64000"/>
      <xsd:minLength value="1"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="typeSUPPLIER_ID">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="50"/>
      <xsd:minLength value="1"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="typeBUYER_ID">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="50"/>
      <xsd:minLength value="1"/>
    </xsd:restriction>
  </xsd:simpleType>
```

```
44  <xsd:simpleType name="typeEANNR">
45    <xsd:restriction base="xsd:string">
46      <xsd:maxLength value="14"/>
47      <xsd:minLength value="1"/>
48    </xsd:restriction>
49  </xsd:simpleType>
50  <xsd:simpleType name="typeERPNR">
51    <xsd:restriction base="xsd:string">
52      <xsd:maxLength value="50"/>
53      <xsd:minLength value="1"/>
54    </xsd:restriction>
55  </xsd:simpleType>
56  <!-- Komplexe Typen -->
57  <xsd:complexType name="typeDATETIME">
58    <xsd:sequence>
59      <xsd:element ref="DATE"/>
60      <xsd:element ref="TIME" minOccurs="0"/>
61      <xsd:element ref="TIMEZONE" minOccurs="0"/>
62    </xsd:sequence>
63  </xsd:complexType>
64  <xsd:complexType name="typeADDRESS">
65    <xsd:sequence>
66      <xsd:element ref="NAME" minOccurs="0"/>
67      <xsd:element ref="NAME2" minOccurs="0"/>
68      <xsd:element ref="NAME3" minOccurs="0"/>
69      <xsd:element ref="CONTACT" minOccurs="0"/>
70      <xsd:element ref="STREET" minOccurs="0"/>
71      <xsd:element ref="ZIP" minOccurs="0"/>
72      <xsd:element ref="BOXNO" minOccurs="0"/>
73      <xsd:element ref="ZIPBOX" minOccurs="0"/>
74      <xsd:element ref="CITY" minOccurs="0"/>
75      <xsd:element ref="STATE" minOccurs="0"/>
76      <xsd:element ref="COUNTRY" minOccurs="0"/>
77      <xsd:element ref="PHONE" minOccurs="0"/>
78      <xsd:element ref="FAX" minOccurs="0"/>
79      <xsd:element ref="EMAIL" minOccurs="0"/>
80      <xsd:element ref="PUBLICKEY" minOccurs="0" maxOccurs="
          unbounded"/>
81      <xsd:element ref="URL" minOccurs="0"/>
82      <xsd:element ref="ADDRESSREMARKS" minOccurs="0"/>
83    </xsd:sequence>
84  </xsd:complexType>
85  <xsd:complexType name="typeFORMULA">
86    <xsd:annotation>
87      <xsd:documentation>Allgemeiner Datentyp fuer eine
          Berechnungsformel</xsd:documentation>
88    </xsd:annotation>
89    <xsd:sequence>
90      <xsd:element ref="FORMULASTRING"/>
91      <xsd:element ref="FORMULADESCRIPTION" minOccurs="0"/>
92    </xsd:sequence>
```

```
93    </xsd:complexType>
94    <xsd:complexType name="typeUNIT">
95      <xsd:annotation>
96        <xsd:documentation>Allgemeiner Datentyp fuer die Angabe von
              Verpackungseinheiten</xsd:documentation>
97      </xsd:annotation>
98      <xsd:simpleContent>
99        <xsd:extension base="xsd:string">
100         <xsd:attribute name="type" use="required">
101           <xsd:simpleType>
102             <xsd:restriction base="xsd:string">
103               <xsd:enumeration value="erp"/>
104               <xsd:enumeration value="iso"/>
105               <xsd:enumeration value="unece"/>
106             </xsd:restriction>
107           </xsd:simpleType>
108         </xsd:attribute>
109       </xsd:extension>
110     </xsd:simpleContent>
111   </xsd:complexType>
112   <!-- Elementdeklarationen -->
113   <!-- einfache Deklarationen -->
114   <xsd:element name="DATE" type="dtDATETYPE">
115     <xsd:annotation>
116       <xsd:documentation>Element fuer ein Datum</xsd:
              documentation>
117     </xsd:annotation>
118   </xsd:element>
119   <xsd:element name="TIME" type="dtTIMETYPE">
120     <xsd:annotation>
121       <xsd:documentation>Element fuer eine Zeitangabe</xsd:
              documentation>
122     </xsd:annotation>
123   </xsd:element>
124   <xsd:element name="TIMEZONE" type="dtTIMEZONETYPE">
125     <xsd:annotation>
126       <xsd:documentation>Element fuer die Angabe einer Zeitzone</
              xsd:documentation>
127     </xsd:annotation>
128   </xsd:element>
129   <xsd:element name="LANGUAGE">
130     <xsd:annotation>
131       <xsd:documentation>Element fuer die Sprache des Dokumentes
              (Kurzform)</xsd:documentation>
132     </xsd:annotation>
133     <xsd:simpleType>
134       <xsd:restriction base="xsd:string">
135         <xsd:minLength value="1"/>
136         <xsd:maxLength value="10"/>
137       </xsd:restriction>
138     </xsd:simpleType>
```

```
139   </xsd:element>
140   <xsd:element name="TERRITORY" type="dtCOUNTRIES">
141     <xsd:annotation>
142       <xsd:documentation>Element fuer ein Land (Kurzform)</xsd:
              documentation>
143     </xsd:annotation>
144   </xsd:element>
145   <xsd:element name="CURRENCY" type="dtCURRENCIES">
146     <xsd:annotation>
147       <xsd:documentation>Element fuer eine Waehrung (Kurzform)</
              xsd:documentation>
148     </xsd:annotation>
149   </xsd:element>
150   <!-- komplexe Deklarationen -->
151   <!-- komplexe Deklarationen innerhalb von Datentypen -->
152   <xsd:element name="NAME">
153     <xsd:annotation>
154       <xsd:documentation>Name</xsd:documentation>
155     </xsd:annotation>
156     <xsd:simpleType>
157       <xsd:restriction base="xsd:string">
158         <xsd:maxLength value="50"/>
159         <xsd:minLength value="1"/>
160       </xsd:restriction>
161     </xsd:simpleType>
162   </xsd:element>
163   <xsd:element name="NAME2">
164     <xsd:annotation>
165       <xsd:documentation>Name (2. Moeglichkeit)</xsd:
              documentation>
166     </xsd:annotation>
167     <xsd:simpleType>
168       <xsd:restriction base="xsd:string">
169         <xsd:maxLength value="50"/>
170         <xsd:minLength value="1"/>
171       </xsd:restriction>
172     </xsd:simpleType>
173   </xsd:element>
174   <xsd:element name="NAME3">
175     <xsd:annotation>
176       <xsd:documentation>Name (3. Moeglichkeit)</xsd:
              documentation>
177     </xsd:annotation>
178     <xsd:simpleType>
179       <xsd:restriction base="xsd:string">
180         <xsd:maxLength value="50"/>
181         <xsd:minLength value="1"/>
182       </xsd:restriction>
183     </xsd:simpleType>
184   </xsd:element>
185   <xsd:element name="CONTACT">
```

```
186       <xsd:annotation>
187         <xsd:documentation>Ansprechpartner</xsd:documentation>
188       </xsd:annotation>
189       <xsd:simpleType>
190         <xsd:restriction base="xsd:string">
191           <xsd:maxLength value="50"/>
192           <xsd:minLength value="1"/>
193         </xsd:restriction>
194       </xsd:simpleType>
195     </xsd:element>
196     <xsd:element name="STREET">
197       <xsd:annotation>
198         <xsd:documentation>Strasse</xsd:documentation>
199       </xsd:annotation>
200       <xsd:simpleType>
201         <xsd:restriction base="xsd:string">
202           <xsd:maxLength value="50"/>
203           <xsd:minLength value="1"/>
204         </xsd:restriction>
205       </xsd:simpleType>
206     </xsd:element>
207     <xsd:element name="ZIP">
208       <xsd:annotation>
209         <xsd:documentation>Postleitzahl</xsd:documentation>
210       </xsd:annotation>
211       <xsd:simpleType>
212         <xsd:restriction base="xsd:string">
213           <xsd:maxLength value="20"/>
214           <xsd:minLength value="1"/>
215         </xsd:restriction>
216       </xsd:simpleType>
217     </xsd:element>
218     <xsd:element name="BOXNO">
219       <xsd:annotation>
220         <xsd:documentation>Postfach</xsd:documentation>
221       </xsd:annotation>
222       <xsd:simpleType>
223         <xsd:restriction base="xsd:string">
224           <xsd:maxLength value="20"/>
225           <xsd:minLength value="1"/>
226         </xsd:restriction>
227       </xsd:simpleType>
228     </xsd:element>
229     <xsd:element name="ZIPBOX">
230       <xsd:annotation>
231         <xsd:documentation>PLZ fuer Postfach</xsd:documentation>
232       </xsd:annotation>
233       <xsd:simpleType>
234         <xsd:restriction base="xsd:string">
235           <xsd:maxLength value="20"/>
236           <xsd:minLength value="1"/>
```

```
237       </xsd:restriction>
238     </xsd:simpleType>
239   </xsd:element>
240   <xsd:element name="CITY">
241     <xsd:annotation>
242       <xsd:documentation>Ort</xsd:documentation>
243     </xsd:annotation>
244     <xsd:simpleType>
245       <xsd:restriction base="xsd:string">
246         <xsd:maxLength value="50"/>
247         <xsd:minLength value="1"/>
248       </xsd:restriction>
249     </xsd:simpleType>
250   </xsd:element>
251   <xsd:element name="STATE">
252     <xsd:annotation>
253       <xsd:documentation>Bundesland/Region</xsd:documentation>
254     </xsd:annotation>
255     <xsd:simpleType>
256       <xsd:restriction base="xsd:string">
257         <xsd:maxLength value="50"/>
258         <xsd:minLength value="1"/>
259       </xsd:restriction>
260     </xsd:simpleType>
261   </xsd:element>
262   <xsd:element name="COUNTRY">
263     <xsd:annotation>
264       <xsd:documentation>Land</xsd:documentation>
265     </xsd:annotation>
266     <xsd:simpleType>
267       <xsd:restriction base="xsd:string">
268         <xsd:maxLength value="50"/>
269         <xsd:minLength value="1"/>
270       </xsd:restriction>
271     </xsd:simpleType>
272   </xsd:element>
273   <xsd:element name="PHONE">
274     <xsd:annotation>
275       <xsd:documentation>Telefonnummer</xsd:documentation>
276     </xsd:annotation>
277     <xsd:simpleType>
278       <xsd:restriction base="xsd:string">
279         <xsd:maxLength value="30"/>
280         <xsd:minLength value="1"/>
281       </xsd:restriction>
282     </xsd:simpleType>
283   </xsd:element>
284   <xsd:element name="FAX">
285     <xsd:annotation>
286       <xsd:documentation>Faxnummer</xsd:documentation>
287     </xsd:annotation>
```

```
288       <xsd:simpleType>
289         <xsd:restriction base="xsd:string">
290           <xsd:maxLength value="30"/>
291           <xsd:minLength value="1"/>
292         </xsd:restriction>
293       </xsd:simpleType>
294     </xsd:element>
295     <xsd:element name="EMAIL">
296       <xsd:annotation>
297         <xsd:documentation>E-Mail-Adresse</xsd:documentation>
298       </xsd:annotation>
299       <xsd:simpleType>
300         <xsd:restriction base="xsd:string">
301           <xsd:maxLength value="100"/>
302           <xsd:minLength value="1"/>
303         </xsd:restriction>
304       </xsd:simpleType>
305     </xsd:element>
306     <xsd:element name="PUBLICKEY">
307       <xsd:annotation>
308         <xsd:documentation>oeffentlicher Schluessel (PGP)</xsd:
              documentation>
309       </xsd:annotation>
310       <xsd:complexType>
311         <xsd:simpleContent>
312           <xsd:extension base="typePUBLIC_KEY">
313             <xsd:attribute name="type" use="required">
314               <xsd:simpleType>
315                 <xsd:restriction base="xsd:string">
316                   <xsd:maxLength value="50"/>
317                   <xsd:minLength value="1"/>
318                 </xsd:restriction>
319               </xsd:simpleType>
320             </xsd:attribute>
321           </xsd:extension>
322         </xsd:simpleContent>
323       </xsd:complexType>
324     </xsd:element>
325     <xsd:element name="URL">
326       <xsd:annotation>
327         <xsd:documentation>URL des Internetauftritts</xsd:
              documentation>
328       </xsd:annotation>
329       <xsd:simpleType>
330         <xsd:restriction base="xsd:string">
331           <xsd:maxLength value="100"/>
332           <xsd:minLength value="1"/>
333         </xsd:restriction>
334       </xsd:simpleType>
335     </xsd:element>
336     <xsd:element name="ADDRESSREMARKS">
```

```
337     <xsd:annotation>
338       <xsd:documentation>Bemerkungen</xsd:documentation>
339     </xsd:annotation>
340     <xsd:simpleType>
341       <xsd:restriction base="xsd:string">
342         <xsd:maxLength value="250"/>
343         <xsd:minLength value="1"/>
344       </xsd:restriction>
345     </xsd:simpleType>
346   </xsd:element>
347   <xsd:element name="FORMULASTRING">
348     <xsd:annotation>
349       <xsd:documentation>Formel zur Berechnung eines Endpreises
              als STRING</xsd:documentation>
350     </xsd:annotation>
351     <xsd:simpleType>
352       <xsd:restriction base="xsd:string">
353         <xsd:minLength value="1"/>
354         <xsd:maxLength value="255"/>
355       </xsd:restriction>
356     </xsd:simpleType>
357   </xsd:element>
358   <xsd:element name="FORMULADESCRIPTION">
359     <xsd:annotation>
360       <xsd:documentation>Beschreibung zur Formel zur
              Preisberechnung</xsd:documentation>
361     </xsd:annotation>
362     <xsd:simpleType>
363       <xsd:restriction base="xsd:string">
364         <xsd:minLength value="1"/>
365         <xsd:maxLength value="64000"/>
366       </xsd:restriction>
367     </xsd:simpleType>
368   </xsd:element>
369   <!-- komplexe Deklarationen ausserhalb der Datentypen -->
370   <xsd:element name="DOCID">
371     <xsd:annotation>
372       <xsd:documentation>Element fuer die ID des Dokumentes</xsd:
              documentation>
373     </xsd:annotation>
374     <xsd:simpleType>
375       <xsd:restriction base="xsd:string">
376         <xsd:minLength value="1"/>
377         <xsd:maxLength value="50"/>
378       </xsd:restriction>
379     </xsd:simpleType>
380   </xsd:element>
381   <xsd:element name="DOCTITLE">
382     <xsd:annotation>
383       <xsd:documentation>Element fuer den Titel des Dokumentes</
              xsd:documentation>
```

```
384       </xsd:annotation>
385       <xsd:simpleType>
386         <xsd:restriction base="xsd:string">
387           <xsd:minLength value="1"/>
388           <xsd:maxLength value="255"/>
389         </xsd:restriction>
390       </xsd:simpleType>
391     </xsd:element>
392     <xsd:element name="DOCVALIDITY">
393       <xsd:annotation>
394         <xsd:documentation>Element fuer den Gueltigkeitszeitraum
              des Dokumentes</xsd:documentation>
395       </xsd:annotation>
396       <xsd:complexType>
397         <xsd:sequence>
398           <xsd:element name="DATETIME" maxOccurs="2">
399             <xsd:annotation>
400               <xsd:documentation>Element fuer die Aufnahme von
                    Datums- und Zeit-Informationen im Bezug auf die
                    Gueltigkeit des Dokumentes</xsd:documentation>
401             </xsd:annotation>
402             <xsd:complexType>
403               <xsd:complexContent>
404                 <xsd:extension base="typeDATETIME">
405                   <xsd:attribute name="type" use="required">
406                     <xsd:simpleType>
407                       <xsd:restriction base="xsd:NMTOKEN">
408                         <xsd:enumeration value="validity_start_date
                              "/>
409                         <xsd:enumeration value="validity_end_date"
                              />
410                       </xsd:restriction>
411                     </xsd:simpleType>
412                   </xsd:attribute>
413                 </xsd:extension>
414               </xsd:complexContent>
415             </xsd:complexType>
416           </xsd:element>
417         </xsd:sequence>
418       </xsd:complexType>
419     </xsd:element>
420     <xsd:element name="SOURCESYSTEM">
421       <xsd:annotation>
422         <xsd:documentation>Element fuer das Quellsystem der Daten</
              xsd:documentation>
423       </xsd:annotation>
424       <xsd:simpleType>
425         <xsd:restriction base="xsd:string">
426           <xsd:minLength value="1"/>
427           <xsd:maxLength value="50"/>
428         </xsd:restriction>
```

```
429       </xsd:simpleType>
430     </xsd:element>
431     <xsd:element name="PRICEFLAG">
432       <xsd:annotation>
433         <xsd:documentation>Element, das mittels einem boolschen
              Wert und einem Attribut type festlegt, was im Preis
              enthalten ist (z.B. inkl. Fracht, etc.)</xsd:
              documentation>
434       </xsd:annotation>
435       <xsd:complexType>
436         <xsd:simpleContent>
437           <xsd:extension base="dtBOOLEAN">
438             <xsd:attribute name="type" use="required">
439               <xsd:simpleType>
440                 <xsd:restriction base="xsd:NMTOKEN">
441                   <xsd:enumeration value="incl_freight"/>
442                   <xsd:enumeration value="incl_duty"/>
443                   <xsd:enumeration value="incl_packing"/>
444                   <xsd:enumeration value="incl_assurance"/>
445                 </xsd:restriction>
446               </xsd:simpleType>
447             </xsd:attribute>
448           </xsd:extension>
449         </xsd:simpleContent>
450       </xsd:complexType>
451     </xsd:element>
452     <xsd:element name="SUPPLIER">
453       <xsd:annotation>
454         <xsd:documentation>Liferanteninformationen</xsd:
              documentation>
455       </xsd:annotation>
456       <xsd:complexType>
457         <xsd:sequence>
458           <xsd:element ref="SUPPLIERID" maxOccurs="unbounded"/>
459           <xsd:element ref="SUPPLIERNAME"/>
460           <xsd:element name="ADDRESS" minOccurs="0">
461             <xsd:annotation>
462               <xsd:documentation>Adressinformationen des
                    Lieferanten</xsd:documentation>
463             </xsd:annotation>
464             <xsd:complexType>
465               <xsd:complexContent>
466                 <xsd:extension base="typeADDRESS">
467                   <xsd:attribute name="type" fixed="supplier"/>
468                 </xsd:extension>
469               </xsd:complexContent>
470             </xsd:complexType>
471           </xsd:element>
472           <!--<xsd:element ref="MIME_INFO" minOccurs="0"/>-->
473         </xsd:sequence>
474       </xsd:complexType>
```

```
475   </xsd:element>
476   <xsd:element name="SUPPLIERID">
477     <xsd:annotation>
478       <xsd:documentation>ID_Nummern des Lieferanten</xsd:
              documentation>
479     </xsd:annotation>
480     <xsd:complexType>
481       <xsd:simpleContent>
482         <xsd:extension base="typeSUPPLIER_ID">
483           <xsd:attribute name="type" use="optional">
484             <xsd:simpleType>
485               <xsd:restriction base="xsd:NMTOKEN">
486                 <xsd:enumeration value="duns"/>
487                 <xsd:enumeration value="iln"/>
488                 <xsd:enumeration value="buyer_specific"/>
489                 <xsd:enumeration value="supplier_specific"/>
490               </xsd:restriction>
491             </xsd:simpleType>
492           </xsd:attribute>
493         </xsd:extension>
494       </xsd:simpleContent>
495     </xsd:complexType>
496   </xsd:element>
497   <xsd:element name="SUPPLIERNAME">
498     <xsd:annotation>
499       <xsd:documentation>Name des Lieferanten</xsd:documentation>
500     </xsd:annotation>
501     <xsd:simpleType>
502       <xsd:restriction base="xsd:string">
503         <xsd:maxLength value="50"/>
504         <xsd:minLength value="1"/>
505       </xsd:restriction>
506     </xsd:simpleType>
507   </xsd:element>
508   <xsd:element name="BUYER">
509     <xsd:annotation>
510       <xsd:documentation>Informationen ueber das einkaufende
              Unternehmen</xsd:documentation>
511     </xsd:annotation>
512     <xsd:complexType>
513       <xsd:sequence>
514         <xsd:element ref="BUYERID" maxOccurs="unbounded"/>
515         <xsd:element ref="BUYERNAME"/>
516         <xsd:element name="ADDRESS" minOccurs="0">
517           <xsd:annotation>
518             <xsd:documentation>Adresse des einkaufenden
                    Unternehmens</xsd:documentation>
519           </xsd:annotation>
520           <xsd:complexType>
521             <xsd:complexContent>
522               <xsd:extension base="typeADDRESS">
```

```
523                 <xsd:attribute name="type" fixed="buyer"/>
524               </xsd:extension>
525             </xsd:complexContent>
526           </xsd:complexType>
527         </xsd:element>
528       </xsd:sequence>
529     </xsd:complexType>
530   </xsd:element>
531   <xsd:element name="BUYERID">
532     <xsd:annotation>
533       <xsd:documentation>ID-Nummern des einkaufenden Unternehmens
              </xsd:documentation>
534     </xsd:annotation>
535     <xsd:complexType>
536       <xsd:simpleContent>
537         <xsd:extension base="typeBUYER_ID">
538           <xsd:attribute name="type" use="optional">
539             <xsd:simpleType>
540               <xsd:restriction base="xsd:NMTOKEN">
541                 <xsd:enumeration value="duns"/>
542                 <xsd:enumeration value="iln"/>
543                 <xsd:enumeration value="buyer_specific"/>
544                 <xsd:enumeration value="supplier_specific"/>
545               </xsd:restriction>
546             </xsd:simpleType>
547           </xsd:attribute>
548         </xsd:extension>
549       </xsd:simpleContent>
550     </xsd:complexType>
551   </xsd:element>
552   <xsd:element name="BUYERNAME">
553     <xsd:annotation>
554       <xsd:documentation>Name des einkaufenden Unternehmens</xsd:
              documentation>
555     </xsd:annotation>
556     <xsd:simpleType>
557       <xsd:restriction base="xsd:string">
558         <xsd:maxLength value="50"/>
559         <xsd:minLength value="1"/>
560       </xsd:restriction>
561     </xsd:simpleType>
562   </xsd:element>
563   <xsd:element name="AGREEMENT">
564     <xsd:annotation>
565       <xsd:documentation>Zusaetzliche Vertragsinformationen</xsd:
              documentation>
566     </xsd:annotation>
567     <xsd:complexType>
568       <xsd:sequence>
569         <xsd:element ref="AGREEMENTID"/>
570         <xsd:element name="DATETIME" maxOccurs="2">
```

```
571           <xsd:annotation>
572             <xsd:documentation>Gueltigkeitszeitraum der
                    zusaetzlichen Vertragsinformationen</xsd:
                    documentation>
573           </xsd:annotation>
574           <xsd:complexType>
575             <xsd:complexContent>
576               <xsd:extension base="typeDATETIME">
577                 <xsd:attribute name="type" use="required">
578                   <xsd:simpleType>
579                     <xsd:restriction base="xsd:NMTOKEN">
580                       <xsd:enumeration value="
                              agreement_start_date"/>
581                       <xsd:enumeration value="agreement_end_date"
                              />
582                     </xsd:restriction>
583                   </xsd:simpleType>
584                 </xsd:attribute>
585               </xsd:extension>
586             </xsd:complexContent>
587           </xsd:complexType>
588         </xsd:element>
589       </xsd:sequence>
590     </xsd:complexType>
591   </xsd:element>
592   <xsd:element name="AGREEMENTID">
593     <xsd:annotation>
594       <xsd:documentation>ID-Nummer des zusaetzlichen Vertrages</
              xsd:documentation>
595     </xsd:annotation>
596     <xsd:simpleType>
597       <xsd:restriction base="xsd:string">
598         <xsd:maxLength value="50"/>
599         <xsd:minLength value="1"/>
600       </xsd:restriction>
601     </xsd:simpleType>
602   </xsd:element>
603   <xsd:element name="ARTICLENR">
604     <xsd:annotation>
605       <xsd:documentation>innerhalb des Kataloges eindeutige
              Artikelnummer</xsd:documentation>
606     </xsd:annotation>
607     <xsd:simpleType>
608       <xsd:restriction base="xsd:string">
609         <xsd:maxLength value="32"/>
610         <xsd:minLength value="1"/>
611       </xsd:restriction>
612     </xsd:simpleType>
613   </xsd:element>
614   <xsd:element name="ARTICLENAME">
615     <xsd:annotation>
```

```
616       <xsd:documentation>Kurzform der Artikelbezeichnung</xsd:
              documentation>
617     </xsd:annotation>
618     <xsd:simpleType>
619       <xsd:restriction base="xsd:string">
620         <xsd:maxLength value="80"/>
621         <xsd:minLength value="1"/>
622       </xsd:restriction>
623     </xsd:simpleType>
624   </xsd:element>
625   <xsd:element name="ARTICLEDESCRIPTION">
626     <xsd:annotation>
627       <xsd:documentation>Artikelbeschreibung</xsd:documentation>
628     </xsd:annotation>
629     <xsd:simpleType>
630       <xsd:restriction base="xsd:string">
631         <xsd:minLength value="1"/>
632         <xsd:maxLength value="64000"/>
633       </xsd:restriction>
634     </xsd:simpleType>
635   </xsd:element>
636   <xsd:element name="EANNR" type="typeEANNR">
637     <xsd:annotation>
638       <xsd:documentation>EAN des Artikels</xsd:documentation>
639     </xsd:annotation>
640   </xsd:element>
641   <xsd:element name="ERPNR">
642     <xsd:annotation>
643       <xsd:documentation>Artikelnummer im jeweiligen per Type
              spezifizierten ERP-System</xsd:documentation>
644     </xsd:annotation>
645     <xsd:complexType>
646       <xsd:simpleContent>
647         <xsd:extension base="typeERPNR">
648           <xsd:attribute name="type" use="required">
649             <xsd:simpleType>
650               <xsd:restriction base="xsd:string">
651                 <xsd:maxLength value="50"/>
652                 <xsd:minLength value="1"/>
653               </xsd:restriction>
654             </xsd:simpleType>
655           </xsd:attribute>
656         </xsd:extension>
657       </xsd:simpleContent>
658     </xsd:complexType>
659   </xsd:element>
660   <xsd:element name="ORDERINFOS">
661     <xsd:annotation>
662       <xsd:documentation>Bestellinformationen</xsd:documentation>
663     </xsd:annotation>
664     <xsd:complexType>
```

```
665       <xsd:sequence>
666         <xsd:element ref="ORDERUNIT" maxOccurs="3"/>
667         <xsd:element ref="CONTENTUNIT" minOccurs="0" maxOccurs="3
              "/>
668         <xsd:element ref="NOCUPEROU" minOccurs="0"/>
669         <xsd:element ref="PRICEQUANTITY" minOccurs="0"/>
670         <xsd:element ref="QUANTITYMIN" minOccurs="0"/>
671         <xsd:element ref="QUANTITYINTERVAL" minOccurs="0"/>
672       </xsd:sequence>
673     </xsd:complexType>
674   </xsd:element>
675   <xsd:element name="ORDERUNIT">
676     <xsd:annotation>
677       <xsd:documentation>Einheit, in der der Artikel bestellt
            werden kann</xsd:documentation>
678     </xsd:annotation>
679     <xsd:complexType>
680       <xsd:simpleContent>
681         <xsd:restriction base="typeUNIT">
682           <xsd:minLength value="1"/>
683           <xsd:maxLength value="10"/>
684         </xsd:restriction>
685       </xsd:simpleContent>
686     </xsd:complexType>
687   </xsd:element>
688   <xsd:element name="CONTENTUNIT">
689     <xsd:annotation>
690       <xsd:documentation>Einheit der Verpackungseinheiten
            innerhalb einer Bestelleinheit</xsd:documentation>
691     </xsd:annotation>
692     <xsd:complexType>
693       <xsd:simpleContent>
694         <xsd:restriction base="typeUNIT">
695           <xsd:minLength value="1"/>
696           <xsd:maxLength value="10"/>
697         </xsd:restriction>
698       </xsd:simpleContent>
699     </xsd:complexType>
700   </xsd:element>
701   <xsd:element name="NOCUPEROU" type="xsd:decimal">
702     <xsd:annotation>
703       <xsd:documentation>Anzahl der Einheiten in einer
            Bestelleinheit</xsd:documentation>
704     </xsd:annotation>
705   </xsd:element>
706   <xsd:element name="PRICEQUANTITY" type="xsd:decimal">
707     <xsd:annotation>
708       <xsd:documentation>Angabe, auf welche Menge an
            Bestelleinheiten sich der Preis bezieht</xsd:
            documentation>
709     </xsd:annotation>
```

```
710   </xsd:element>
711   <xsd:element name="QUANTITYMIN" type="xsd:integer">
712     <xsd:annotation>
713       <xsd:documentation>Mindestbestellmenge</xsd:documentation>
714     </xsd:annotation>
715   </xsd:element>
716   <xsd:element name="QUANTITYINTERVAL" type="xsd:integer">
717     <xsd:annotation>
718       <xsd:documentation>Angabe, in welcher Staffelung ein
              Artikel bestellt werden kann</xsd:documentation>
719     </xsd:annotation>
720   </xsd:element>
721   <xsd:element name="DAILYPRICE" type="dtBOOLEAN">
722     <xsd:annotation>
723       <xsd:documentation>boolscher Wert, der mit Wert TRUE
              anzeigt, dass der Preis des Artikels starken
              Schwankungen ausgesetzt ist (z.B. durch
              Materialpreisaenderungen, etc.)</xsd:documentation>
724     </xsd:annotation>
725   </xsd:element>
726   <xsd:element name="PRICES">
727     <xsd:annotation>
728       <xsd:documentation>Preise des Artikels</xsd:documentation>
729     </xsd:annotation>
730     <xsd:complexType>
731       <xsd:sequence>
732         <xsd:element ref="PRICETYPE"/>
733         <xsd:element ref="PRICEFLAG" minOccurs="0" maxOccurs="
              unbounded"/>
734         <xsd:element ref="BASEPRICEAMOUNT" minOccurs="0"/>
735         <xsd:element ref="ENDPRICEAMOUNT"/>
736         <xsd:element ref="CURRENCY" minOccurs="0"/>
737         <xsd:element ref="TAX" minOccurs="0"/>
738         <xsd:element ref="REBATEFACTOR" minOccurs="0"/>
739         <xsd:element ref="TERRITORY" minOccurs="0" maxOccurs="
              unbounded"/>
740         <xsd:choice minOccurs="0">
741           <xsd:element ref="PRICESCALE"/>
742           <xsd:element ref="PRICEFORMULA"/>
743         </xsd:choice>
744       </xsd:sequence>
745     </xsd:complexType>
746   </xsd:element>
747   <xsd:element name="PRICETYPE">
748     <xsd:annotation>
749       <xsd:documentation>Element fuer den Typ des Preises, z.B.
              Listenpreis mit Umsatzsteuer, etc.</xsd:documentation>
750     </xsd:annotation>
751     <xsd:simpleType>
752       <xsd:restriction base="xsd:string">
753         <xsd:minLength value="1"/>
```

```
754         <xsd:maxLength value="255"/>
755       </xsd:restriction>
756     </xsd:simpleType>
757   </xsd:element>
758   <xsd:element name="BASEPRICEAMOUNT" type="xsd:decimal">
759     <xsd:annotation>
760       <xsd:documentation>Preisbetrag</xsd:documentation>
761     </xsd:annotation>
762   </xsd:element>
763   <xsd:element name="ENDPRICEAMOUNT" type="xsd:decimal">
764     <xsd:annotation>
765       <xsd:documentation>Preisbetrag</xsd:documentation>
766     </xsd:annotation>
767   </xsd:element>
768   <xsd:element name="TAX" type="xsd:decimal">
769     <xsd:annotation>
770       <xsd:documentation>Steuerinformationen (mehr erforderlich?)
            </xsd:documentation>
771     </xsd:annotation>
772   </xsd:element>
773   <xsd:element name="REBATEFACTOR" type="xsd:decimal">
774     <xsd:annotation>
775       <xsd:documentation>Rabattfaktor, der mit Preisbetrag zur
            Ermittlung des Endpreises multipliziert werden muss</xsd
            :documentation>
776     </xsd:annotation>
777   </xsd:element>
778   <xsd:element name="PRICESCALE">
779     <xsd:annotation>
780       <xsd:documentation>Staffelgrenzen bei Preisstaffeln</xsd:
            documentation>
781     </xsd:annotation>
782     <xsd:complexType>
783       <xsd:sequence>
784         <xsd:element ref="LOWERBOUND"/>
785         <xsd:element ref="UPPERBOUND" minOccurs="0"/>
786       </xsd:sequence>
787     </xsd:complexType>
788   </xsd:element>
789   <xsd:element name="LOWERBOUND" type="xsd:decimal">
790     <xsd:annotation>
791       <xsd:documentation>untere Staffelgrenze</xsd:documentation>
792     </xsd:annotation>
793   </xsd:element>
794   <xsd:element name="UPPERBOUND" type="xsd:decimal">
795     <xsd:annotation>
796       <xsd:documentation>obere Staffelgrenze</xsd:documentation>
797     </xsd:annotation>
798   </xsd:element>
799   <xsd:element name="PRICEFORMULA" type="typeFORMULA">
800     <xsd:annotation>
```

```
801       <xsd:documentation>Preisberechnungsformel</xsd:
              documentation>
802     </xsd:annotation>
803   </xsd:element>
804   <xsd:element name="VARIANT">
805     <xsd:annotation>
806       <xsd:documentation>Variante eines Artikels</xsd:
              documentation>
807     </xsd:annotation>
808     <xsd:complexType>
809       <xsd:sequence>
810         <xsd:element ref="VARIANTID"/>
811         <xsd:element ref="VARIANTNAME"/>
812         <xsd:element ref="VARIANTDESCRIPTION" minOccurs="0"/>
813         <xsd:element ref="ORDER"/>
814         <xsd:element ref="SEPARATOR" minOccurs="0"/>
815         <xsd:element ref="CHARACTERISTIC" maxOccurs="unbounded"/>
816       </xsd:sequence>
817       <xsd:attribute name="type" use="required">
818         <xsd:simpleType>
819           <xsd:restriction base="xsd:NMTOKEN">
820             <xsd:enumeration value="optional"/>
821             <xsd:enumeration value="required"/>
822           </xsd:restriction>
823         </xsd:simpleType>
824       </xsd:attribute>
825     </xsd:complexType>
826   </xsd:element>
827   <xsd:element name="SEPARATOR" type="xsd:string">
828     <xsd:annotation>
829       <xsd:documentation>Trennzeichen, das Auswahlcodes von der
              Basis-Artikelnummer trennt</xsd:documentation>
830     </xsd:annotation>
831   </xsd:element>
832   <xsd:element name="VARIANTID">
833     <xsd:annotation>
834       <xsd:documentation>ID der Variante</xsd:documentation>
835     </xsd:annotation>
836     <xsd:simpleType>
837       <xsd:restriction base="xsd:string">
838         <xsd:minLength value="1"/>
839         <xsd:maxLength value="10"/>
840       </xsd:restriction>
841     </xsd:simpleType>
842   </xsd:element>
843   <xsd:element name="VARIANTNAME">
844     <xsd:annotation>
845       <xsd:documentation>Name der Variante</xsd:documentation>
846     </xsd:annotation>
847     <xsd:simpleType>
848       <xsd:restriction base="xsd:string">
```

```
849         <xsd:minLength value="1"/>
850         <xsd:maxLength value="150"/>
851       </xsd:restriction>
852     </xsd:simpleType>
853   </xsd:element>
854   <xsd:element name="VARIANTDESCRIPTION">
855     <xsd:annotation>
856       <xsd:documentation>Weitere Beschreibung der Variante</xsd:
            documentation>
857     </xsd:annotation>
858     <xsd:simpleType>
859       <xsd:restriction base="xsd:string">
860         <xsd:minLength value="1"/>
861         <xsd:maxLength value="64000"/>
862       </xsd:restriction>
863     </xsd:simpleType>
864   </xsd:element>
865   <xsd:element name="ORDER" type="xsd:integer">
866     <xsd:annotation>
867       <xsd:documentation>Reihenfolge der Varianten</xsd:
            documentation>
868     </xsd:annotation>
869   </xsd:element>
870   <xsd:element name="CHARACTERISTIC">
871     <xsd:annotation>
872       <xsd:documentation>Auspraegungen einer Variante</xsd:
            documentation>
873     </xsd:annotation>
874     <xsd:complexType>
875       <xsd:sequence>
876         <xsd:sequence>
877           <xsd:element ref="CHOICECODE"/>
878           <xsd:element ref="CHARNAME"/>
879           <xsd:element ref="CHARDESCRIPTION" minOccurs="0"/>
880         </xsd:sequence>
881         <xsd:sequence minOccurs="0">
882           <xsd:element ref="CALCTYPE"/>
883           <xsd:choice>
884             <xsd:element ref="PRICEVARIATION"/>
885             <xsd:element ref="PRICEVARFORMULA"/>
886           </xsd:choice>
887         </xsd:sequence>
888       </xsd:sequence>
889     </xsd:complexType>
890   </xsd:element>
891   <xsd:element name="CHOICECODE">
892     <xsd:annotation>
893       <xsd:documentation>Auswahlcode, der an eine Bestellnummer
            angehaengt werden kann</xsd:documentation>
894     </xsd:annotation>
895     <xsd:simpleType>
```

```
896       <xsd:restriction base="xsd:string">
897         <xsd:minLength value="1"/>
898         <xsd:maxLength value="10"/>
899       </xsd:restriction>
900     </xsd:simpleType>
901   </xsd:element>
902   <xsd:element name="CHARNAME">
903     <xsd:annotation>
904       <xsd:documentation>Name der Variantenauspraegung</xsd:
              documentation>
905     </xsd:annotation>
906     <xsd:simpleType>
907       <xsd:restriction base="xsd:string">
908         <xsd:minLength value="1"/>
909         <xsd:maxLength value="150"/>
910       </xsd:restriction>
911     </xsd:simpleType>
912   </xsd:element>
913   <xsd:element name="CHARDESCRIPTION">
914     <xsd:annotation>
915       <xsd:documentation>Weitere Beschreibung der
              Variantenauspraegung</xsd:documentation>
916     </xsd:annotation>
917     <xsd:simpleType>
918       <xsd:restriction base="xsd:string">
919         <xsd:minLength value="1"/>
920         <xsd:maxLength value="64000"/>
921       </xsd:restriction>
922     </xsd:simpleType>
923   </xsd:element>
924   <xsd:element name="PRICEVARIATION" type="xsd:decimal">
925     <xsd:annotation>
926       <xsd:documentation>Betrag oder Faktor, um den ein
              Basispreis durch die Auswahl der Variantenauspraegung
              beeinflusst wird</xsd:documentation>
927     </xsd:annotation>
928   </xsd:element>
929   <xsd:element name="PRICEVARFORMULA" type="typeFORMULA">
930     <xsd:annotation>
931       <xsd:documentation>Formel, nach der sich eine
              Preisvariation berechnen laesst</xsd:documentation>
932     </xsd:annotation>
933   </xsd:element>
934   <xsd:element name="CALCTYPE">
935     <xsd:annotation>
936       <xsd:documentation>Rechenart, mit der Basispreis und der
              Wert in PRICEVARIATION miteinander verrechnet werden, um
               den Endpreis zu ermitteln</xsd:documentation>
937     </xsd:annotation>
938     <xsd:simpleType>
939       <xsd:restriction base="xsd:string">
```

```
940         <xsd:minLength value="1"/>
941         <xsd:maxLength value="10"/>
942       </xsd:restriction>
943     </xsd:simpleType>
944   </xsd:element>
945 </xsd:schema>
```

Listing 8.1: XML Schema: priceinfo_base.xsd

priceinfo.xsd

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified" attributeFormDefault="
       unqualified">
3    <xsd:include schemaLocation="priceinfo_base.xsd"/>
4    <xsd:element name="PRICEINFO">
5      <xsd:annotation>
6        <xsd:documentation>Root-Element des Priceinfo-Dokumentes</
             xsd:documentation>
7      </xsd:annotation>
8      <xsd:complexType>
9        <xsd:sequence>
10         <xsd:element ref="HEADER"/>
11         <xsd:element ref="ARTICLEDATA"/>
12         <xsd:element ref="PRICEDATA" maxOccurs="unbounded"/>
13         <xsd:element ref="VARIANTLIST" minOccurs="0"/>
14       </xsd:sequence>
15     </xsd:complexType>
16   </xsd:element>
17   <xsd:element name="HEADER">
18     <xsd:annotation>
19       <xsd:documentation>Header-Element mit allgemeinen Daten zu
             Preisinformationen </xsd:documentation>
20     </xsd:annotation>
21     <xsd:complexType>
22       <xsd:sequence>
23         <xsd:element ref="DOCID"/>
24         <xsd:element ref="DOCTITLE"/>
25         <xsd:element name="DOCDATETIME">
26           <xsd:annotation>
27             <xsd:documentation>Element fuer das Erstellungsdatum
                   des Dokumentes</xsd:documentation>
28           </xsd:annotation>
29           <xsd:complexType>
30             <xsd:complexContent>
31               <xsd:extension base="typeDATETIME">
32                 <xsd:attribute name="type" fixed="creation_date"
                       />
```

```
33              </xsd:extension>
34            </xsd:complexContent>
35          </xsd:complexType>
36        </xsd:element>
37        <xsd:element ref="DOCVALIDITY"/>
38        <xsd:element ref="SOURCESYSTEM" minOccurs="0"/>
39        <xsd:element ref="LANGUAGE"/>
40        <xsd:element ref="TERRITORY" minOccurs="0" maxOccurs="
             unbounded"/>
41        <xsd:element ref="CURRENCY" minOccurs="0" maxOccurs="
             unbounded"/>
42        <xsd:element ref="PRICEFLAG" minOccurs="0" maxOccurs="
             unbounded"/>
43        <xsd:element ref="SUPPLIER"/>
44        <xsd:element ref="BUYER"/>
45        <xsd:element ref="AGREEMENT" minOccurs="0" maxOccurs="
             unbounded"/>
46      </xsd:sequence>
47    </xsd:complexType>
48  </xsd:element>
49  <xsd:element name="ARTICLEDATA">
50    <xsd:annotation>
51      <xsd:documentation>Element fuer die Preis- und Bestellinfos
           </xsd:documentation>
52    </xsd:annotation>
53    <xsd:complexType>
54      <xsd:sequence>
55        <xsd:element ref="ARTICLENR"/>
56        <xsd:element ref="ARTICLENAME"/>
57        <xsd:element ref="ARTICLEDESCRIPTION" minOccurs="0"/>
58        <xsd:element ref="EANNR" minOccurs="0"/>
59        <xsd:element ref="ERPNR" minOccurs="0" maxOccurs="
             unbounded"/>
60        <xsd:element ref="ORDERINFOS"/>
61      </xsd:sequence>
62    </xsd:complexType>
63  </xsd:element>
64  <xsd:element name="PRICEDATA">
65    <xsd:annotation>
66      <xsd:documentation>Preisinformationen</xsd:documentation>
67    </xsd:annotation>
68    <xsd:complexType>
69      <xsd:sequence>
70        <xsd:element name="DATETIME" minOccurs="0" maxOccurs="2">
71          <xsd:annotation>
72            <xsd:documentation>Gueltigkeitszeitraum der
                 Preisinformationen</xsd:documentation>
73          </xsd:annotation>
74          <xsd:complexType>
75            <xsd:complexContent>
76              <xsd:extension base="typeDATETIME">
```

```
77                  <xsd:attribute name="type" use="required">
78                    <xsd:simpleType>
79                      <xsd:restriction base="xsd:NMTOKEN">
80                        <xsd:enumeration value="price_start_date"/>
81                        <xsd:enumeration value="price_end_date"/>
82                      </xsd:restriction>
83                    </xsd:simpleType>
84                  </xsd:attribute>
85                </xsd:extension>
86              </xsd:complexContent>
87            </xsd:complexType>
88          </xsd:element>
89          <xsd:element ref="DAILYPRICE" minOccurs="0"/>
90          <xsd:element ref="PRICES" maxOccurs="unbounded"/>
91        </xsd:sequence>
92      </xsd:complexType>
93    </xsd:element>
94    <xsd:element name="VARIANTLIST">
95      <xsd:annotation>
96        <xsd:documentation>Liste mit moeglichen Varianten eines
              Artikels</xsd:documentation>
97      </xsd:annotation>
98      <xsd:complexType>
99        <xsd:sequence>
100         <xsd:element ref="VARIANT" maxOccurs="unbounded"/>
101       </xsd:sequence>
102     </xsd:complexType>
103   </xsd:element>
104 </xsd:schema>
```

Listing 8.2: XML Schema: priceinfo.xsd