Seminararbeit Verteilte Architekturen und Middleware:

„Transformation von Nachrichten in Messaging Systemen zur Kommunikation in heterogenen IT-Infrastrukturen“

Student:	Christian Schwörer
Matrikel-Nummer:	[redacted]
vorgelegt am:	10. März 2008
Vorlesung:	Verteilte Architekturen und Middleware
Studiengang:	Computer Science and Media (Master)
Betreuender Professor:	Prof. Walter Kriha
Studienhalbjahr:	1. Semester

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ASCII:	American Standard Code for Information Interchange
B2B:	Business-to-Business
CRM:	Customer Relationship Management
CSV:	Comma Separated Values
EAI:	Enterprise Application Integration
EBCDIC:	Extended Binary Coded Decimals Interchange Code
HTTP:	Hypertext Transfer Protocol
JMS:	Java Message Service
OSI:	Open Systems Interconnection
SOAP:	Simple Object Access Protocol
TCP/IP:	Transmission Control Protocol/Internet Protocol
W3C:	World Wide Web Consortium
XML:	eXtensible Markup Language
XSL:	eXtensible Stylesheet Language
XSLT:	XSL Transformation

Einleitung

„Zusammenkommen ist ein Anfang.
Zusammenbleiben ist ein Fortschritt.
Zusammenarbeiten ist ein Erfolg.“
Henry Ford

Nicht nur im zwischenmenschlichen Bereich ist Zusammenarbeit ein entscheidender Erfolgsfaktor. Auch in IT-Anwendungsarchitekturen werden Insellösungen, die nicht mit anderen Anwendungen kommunizieren, zunehmend abgelöst von prozessorientierten, interoperablen und ganzheitlichen Kommunikationsstrukturen.

Diese Arbeit entstand im Rahmen der Vorlesung „Verteilte Architekturen und Middleware“ im ersten Semester des Master-Studiengangs *Computer Science and Media* an der Hochschule der Medien.

Sie beinhaltet die Vorstellung von *Messaging* als Kommunikationsform in heterogenen IT-Infrastrukturen. Dazu werden im ersten Kapitel einige Grundbegriffe und Definitionen geklärt, um dann im zweiten Kapitel das Thema *Transformation von Nachrichten in Messaging Systemen* darzustellen.

1. Einführung in Messaging

In diesem Kapitel wird *Messaging* als Bestandteil von *Enterprise Application Integration (EAI)* vorgestellt. Dabei wird nur ein grober Überblick über wichtige Begrifflichkeiten gegeben, detaillierte Informationen sind in entsprechender Fachliteratur[1)] zu finden.

1.1 Enterprise Application Integration (EAI)

Der Begriff *Enterprise Application Integration (EAI)* wurde durch das Bestreben geprägt, verschiedene Anwendungen, die ursprünglich nicht für eine Zusammenarbeit entworfen wurden und lediglich Teilaufgaben von Geschäftsprozessen abdecken, dazu zu bringen, in ganzheitlichen Geschäftsprozessen zusammenzuarbeiten. Die kooperierenden Anwendungssysteme sollten sich dabei möglichst so verhalten, als wären sie von Anfang an dafür entworfen worden, die innerbetrieblichen Geschäftprozesse eines Unternehmens vollständig zu unterstützen.[2)]

EAI kann folglich als prozessorientierte Integration von Anwendungen entlang der Wertschöpfungskette in heterogenen IT-Anwendungsarchitekturen bezeichnet werden.
Eine allgemein gehaltenere Definition beschreibt EAI folgendermaßen: „EAI is the unrestricted sharing of data and business processes among any connected applications and data sources in the enterprise."[3)]

Für die Integration heterogener Anwendungen werden *EAI-Lösungen* eingesetzt. EAI-Lösungen bestehen grundsätzlich aus einer Middleware, die um EAI-Komponenten (wie beispielsweise Geschäftsprozessmodellierung, Transformationstools und Monitoring) erweitert wurde.[4)]

Während es bei EAI um die Beschleunigung und Rationalisierung der innerbetrieblichen Informationsflüsse geht, bezeichnet in Abgrenzung dazu die *B2B-Integration* die Optimierung der Informationsflüsse zwischen Unternehmen.[5)]

---

1) Vgl. dazu ausführlich: Conrad, S. et al. (2005)

2) Vgl. Keller, W. (2002), S. 5

3) Linthicum, D. (2000), S.3

4) Vgl. Bator, F. (2006)

5) Vgl. Keller, W. (2002), S. 6 f.

1.2 Messaging und Messaging Systeme

Die Kommunikation in einer heterogenen IT-Anwendungsarchitektur mittels *Nachrichten (Messages)* hat gegenüber der Kommunikation über kompilierte Schnittstellen den Vorteil, dass eine lose Kopplung zwischen den einzelnen Anwendungen möglich ist.[6)] Sender und Empfänger müssen lediglich die Schnittstelle kennen, die benötigt wird, um Nachrichten zu senden und zu empfangen. Sie müssen sich nicht um ihren „Gegenpart" oder die verwendete Soft- und Hardware kümmern.

Asynchrones Messaging begegnet einem grundsätzlichen Problem von verteilten Systemen: Sender und Empfänger einer Nachricht müssen nicht zur selben Zeit verfügbar sein.[7)] Die zuverlässige Verteilung und Zustellung von Nachrichten zwischen Teilnehmern übernimmt ein *Messaging System*. Anwendungen können sich bei der Bearbeitung von Daten darauf konzentrieren, welche Daten sie austauschen und nicht wie der Austausch vonstatten geht.[8)]

Die Kommunikation mit einem Messaging System gestaltet sich wie in Abbildung 1 gezeigt:

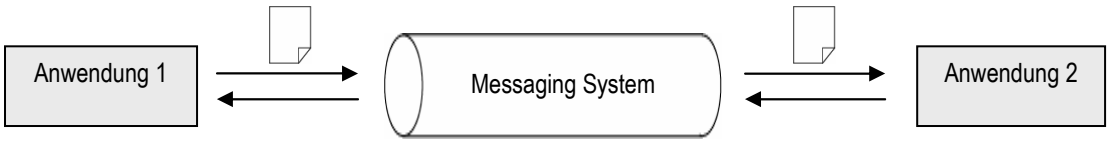


Abb. 1: Kommunikation im Messaging System

Messaging Systeme basieren auf folgenden Grundkonzepten:[9)]

- *Channels*: Sender und Empfänger werden über *Message Channels* (sozusagen „virtuelle Leitungen") verbunden, um Daten zu transportieren.
- *Messages*: Eine Nachricht ist ein atomares Paket von Daten, das über einen Channel übertragen wird.
- *Pipes und Filters*: Pipes und Filters kommen zum Einsatz, wenn während der Kommunikation Funktionen auf Nachrichten ausgeführt werden sollen, wie zum Beispiel das Ver- und Entschlüsseln von Nachrichten.
- *Routing*: Mittels eines *Message Routers* wird der Weg einer Nachricht zwischen Sender und Empfänger festgelegt.

---

6) Vgl. Herden, S. / Gomez, J. et al. (2006), S. 90

7) Vgl. Hohpe, G. / Woolf, B. et al. (2004), S. 54

8) Vgl. Herden, S. / Gomez, J. et al. (2006), S. 91

9) Vgl. Hohpe, G. / Woolf, B. et al. (2004), S. 57 f.

- *Transformation*: Die verschiedenen involvierten Applikationen benutzen oftmals unterschiedliche Datenformate. Mit einem zwischengeschalteten Filter kann eine Umwandlung von Quellformat in Zielformat erfolgen. Dieses Konzept wird ausführlich in Kapitel 2 vorgestellt.
- *Endpoints*: Applikationen werden über *Message Endpoints* an Channels angeschlossen.

Middleware, die auf der Übertragung von Nachrichten basiert, wird als *Message oriented Middleware (MoM)* bezeichnet.[10)] MoM-Produkte realisieren eine asynchrone Kommunikation, über die Nachrichten angelegt, transformiert, weitergegeben, ausgeliefert und gespeichert werden können. Die Einordnung von Middleware in die IT-Anwendungsarchitektur zeigt Abbildung 2.

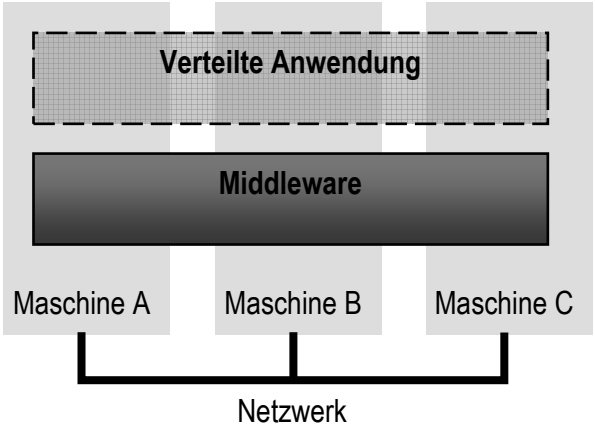


Abb. 2: Middleware in der IT-Anwendungsarchitektur[11)]

Dabei unterstützt *Message oriented Middleware* drei verschiedene Kommunikationsprotokolle: *Message Passing* (direkte Weitergabe von Nachrichten zwischen Applikationen), *Message Queueing* (indirekte Kommunikation über eine Nachrichten-Queue) und *Publish & Subscribe* (Publisher stellt Subscriber Messages zur Verfügung).[12)]
Ein kommerzielles Beispiel für eine Message oriented Middleware ist *MQSeries* von IBM.[13)]

[10)] Vgl. Weitzel, T. / Harder, T / Buxmann, P (2001), S. 66

[11)] Mit Änderungen entnommen aus: Siepmann, F. (2005), S. 3

[12)] Vgl. Keller, W. (2002), S. 86

[13)] Vgl. dazu ausführlich: www.mqseries.net

1.3 Aufbau der Nachrichten

Um mit einem Messaging System Daten zwischen Sender und Empfänger zu versenden, werden die Daten in eine oder mehrere *Nachrichten (Messages)* verpackt. Eine Nachricht besteht aus zwei grundlegenden Teilen: [14)]

- *Header*: beinhaltet Metainformationen über die gesendete Nachricht, die für das Messaging System von Bedeutung sind (zum Beispiel Sender, Empfänger, Nachrichtentyp, usw.).
- *Body*: beinhaltet die eigentlichen Daten, die übertragen werden sollen. Der Body wird vom Messaging System weitestgehend ignoriert.

Als Beispiel für den Aufbau einer Nachricht wird in Abbildung 3 eine *SOAP Message* gezeigt. Eine SOAP Message ist ein XML-Dokument mit einem Wurzelement *Envelope*. Envelope beinhaltet wie oben beschrieben Header (`env:Header`) und Body (`env:Body`).

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
     <t:transaction
           xmlns:t="http://thirdparty.example.org/transaction"
           env:encodingStyle="http://example.com/encoding"
           env:mustUnderstand="true">5</t:transaction>
 </env:Header>
 <env:Body>
   <m:chargeReservation
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
         xmlns:m="http://travelcompany.example.org/">
     <m:reservation xmlns:m="http://travelcompany.example.org/reserve">
       <m:code>FT35ZBQ</m:code>
     </m:reservation>
     <o:creditCard xmlns:o="http://mycompany.example.com/financial">
       <n:name xmlns:n="http://mycompany.example.com/employees">
           Max Mustermann
       </n:name>
       <o:number>123456789099999</o:number>
       <o:expiration>2005-02</o:expiration>
     </o:creditCard>
  </m:chargeReservation>
 </env:Body>
</env:Envelope>
```

Abb. 3: Eine SOAP Message mit Header und Body [15)]

[14)] Vgl. Herden, S. / Gomez, J. et al. (2006), S. 94

[15)] Mit Änderungen entnommen aus: W3C (Hrsg.) (2007)

Für das Messaging System sind alle Nachrichten gleich: Ein Body mit Daten, der wie im Header beschrieben übermittelt werden sollen. Für den Anwendungsentwickler gibt es jedoch verschiedene Arten von Nachrichten: Eine *Command Message* wird benutzt, um eine Prozedur in einer anderen Applikation aufzurufen, mit einer *Document Message* wird einer anderen Anwendung ein Datensatz zugesandt, eine *Event Message* informiert den Empfänger über eine Änderung und ein *Request-Reply* fordert zu einer Antwort auf.[16)]

1.4 Die eXtensible Markup Language (XML) als Nachrichtenformat

Die meisten EAI-Infrastrukturen nutzen eine Form von genormtem Nachrichtenformat. Der Einsatz der *eXtensible Markup Language (XML)* als Sprachbasis für die Nachrichten ist dabei in der Praxis weit verbreitet.

Die besonderen Eigenschaften von XML prädestinieren es für den Einsatz als Austausch- und Nachrichtenformat: [17)]

- XML besitzt eine **einfache und standardisierte Syntax,** die das Erstellen und Lesen von XML-Dokumenten erleichtert.
- Es ermöglicht auch **komplexe Datenstrukturen** abzubilden, da Elemente beliebig tief verschachtelt werden können.
- XML ist im Gegensatz zu Binärformaten sowohl **maschinen- als auch menschenlesbar** und ermöglicht deshalb unter Anderem eine einfache Fehlersuche.

Abschließend bleibt anzumerken, dass XML im Alleingang nicht die Lösung aller Kompatibilitäts-probleme ist. Es macht diesen Weg aber entschieden leichter, da mit XML ein flexibler und etablierter Sprachstandard zur Verfügung steht.[18)]

---

16) Vgl. Vgl. Hohpe, G. / Woolf, B. et al. (2004), S. 67

17) Harold E.R. / Means, W.S. (2001)

18) Weitzel, T. / Harder, T / Buxmann, P. (2001), S. 75

2. Message Transformation

Im ersten Kapitel wurde die nachrichtenbasierte Kommunikation zwischen verschiedenen Applikationen eines Unternehmens vorgestellt. Diese unterschiedlichen Anwendungen wurden im Lauf der Zeit aufgabenspezifisch erstellt und weiterentwickelt. Daraus ergibt sich das Problem, dass jedes System auch sein eigenes Datenmodell zugrunde liegen hat. Das hat zur Folge, dass jede Anwendung eine etwas andere Auffassung von bestimmten Daten hat. Zum Beispiel stehen beim Kunden-beziehungsmanagement (CRM) Kundendaten wie Adresse, Ansprechpartner, Telefonnummern usw. eher im Vordergrund, während bei einem Buchhaltungssystem Kundendaten wie Steuernummer, Zahlungskonditionen usw. von Bedeutung sind. Somit erwartet jede Anwendung, dass die Nachrichten die sie empfängt, auch ihrem spezifischen Datenformat entsprechen.

Da die lose Kopplung von Anwendungen eines der Kernprinzipien von EAI ist, sollten die bestehenden Anwendungen möglichst nicht in der Form geändert werden, dass sie dem Datenformat eines anderen Systems angepasst werden. Abgesehen davon ist das Ändern des Datenmodells einer Anwendung meist sehr aufwändig und riskant.

Die Lösung dieses Problems ist die Umwandlung von Datenformaten in das Messaging System auszulagern. In diesem Kapitel wird zuerst anhand des *Message Translators* ein Konzept hierzu vorgestellt, um anschließend anhand von zwei Beispielen eine mögliche Realisierung zu skizzieren.

2.1 Prinzip des Message Translators

Der hier vorgestellte *Message Translator* basiert auf dem entsprechenden *Enterprise Integration Pattern* von Hohpe, G. / Woolf, B. et al. (2004).[19)]

Ganz allgemein formuliert ist der Message Translator als Bestandteil des Messaging Systems ein spezieller *Filter*, der zwischen verschiedenen Datenformaten übersetzt. Abbildung 4 zeigt den prinzipiellen Ablauf.
Grundsätzlich unterscheidet sich beim Message Translator die eingehende Nachricht immer von der ausgehenden Nachricht. Dabei kommt es auf die Logik des Translators an, in wiefern sich die beiden Nachrichten voneinander unterscheiden.[20)]

---

19) Vgl. dazu ausführlich: Hohpe, G. / Woolf, B. et al. (2004), S. 85 ff.

20) Vgl. Druckmüller, B. (2007), S.56

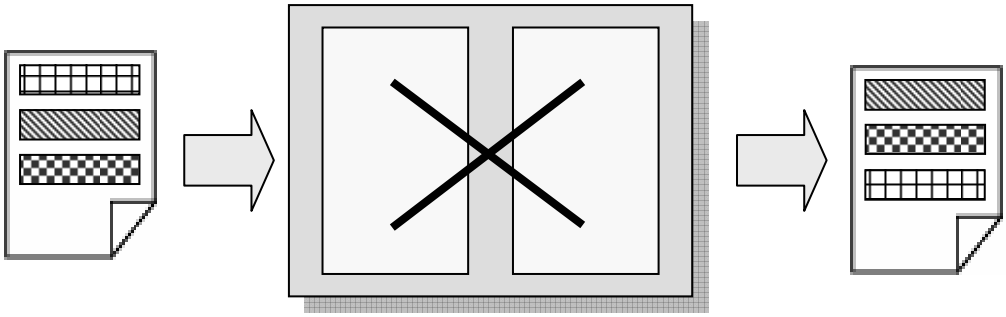


Abb. 4: Prinzipieller Ablauf des Message Translators [21)]

Transformation von Nachrichten kann auf verschiedenen Ebenen notwendig sein. So können beispielsweise Datenelemente zwar denselben Namen und Datentyp haben, liegen aber in verschiedenen Darstellungen vor (z.B. XML-Dokument vs. CSV-Datei). Oder die Datenelemente sind alle in XML-Dokumenten enthalten, es werden aber für die entsprechenden Elemente unterschiedliche Tag-Namen verwendet.

Die folgende Tabelle zeigt die verschiedenen Transformationsebenen, dargestellt über ein Schichtenmodell (angelehnt an das OSI-Referenzmodell): [22)]

Schicht	**behandelt**	**Beispiel für Umwandlung**	**Tools / Techniken**
Transport	Kommunikationsprotokolle (TCP/IP sockets, HTTP, SOAP, JMS, …)	Daten zwischen den Protokollen austauschen, ohne den Inhalt zu verändern	EAI Adapter, Channel Adapter
Datenrepräsentation	Datenformate (XML, Name-Wert-Paare, proprietäre Formate,…) Zeichensätze (ASCII, Unicode, EBCDIC, …)	Parsen der Datenrepräsentation und Rendern in ein anderes Format	XML Parser, EAI Parser / Rendering Tools
Datentypen	Feldnamen, Datentypen, Constraints	Umwandeln einer PLZ von Zahl in Zeichenkette. Verbinden von „Anrede“, „Vorname“ und „Nachname“ zu einem einzigen Namensfeld	Visuelle EAI-Transformationstools (z.B. BizTalk Mapper), XSL, Datenbank-Lookups
Datenstrukturen / Anwendungsschicht	Entitäten, Assoziationen, Kardinalitäten	Zusammenfassen von n:n-Beziehungen in Aggregationen	Strukturelle Mapping Patterns, spezifische Lösungen

21) Mit Änderungen entnommen aus: Hohpe, G. / Woolf, B. et al. (2004), S. 86

22) Vgl. Hohpe, G. / Woolf, B. et al. (2004), S. 87 f.

In der *Transportschicht* wird auf Bitstream-Ebene der Datentransfer zwischen den verschiedenen Systemen zur Verfügung gestellt. Zu ihren Aufgaben zählt mit der Kommunikation zwischen den Netzwerk-Segmenten auch die Behandlung der Netzwerk-Fehler. Einige EAI-Lösungen arbeiten auf dieser Ebene mit proprietären Transportprotokollen (z.B. *TIBCO RendezVous*), andere Lösungen setzen auf TCP/IP-basierte Protokolle (wie z.B. *SOAP*). Als Tools kommen bei dieser Schicht *Channel Adapter* [23)] in Frage.

Die *Datenrepräsentationsschicht* wird auch als *Syntax-Layer* bezeichnet. Sie definiert die Darstellung der transportierten Daten. Ein gebräuchliches Format für die Darstellung komplexer Datenstrukturen ist XML (siehe Kapitel 1.4).

Die *Datentypenschicht* behandelt strukturelle Fragen: Ist ein Datum ein numerischer Wert oder eine Zeichenkette? Beinhaltet das Datum die Angabe einer Zeitzone? Liegt eine Uhrzeit in der 24-Stunden- oder 12-Stunden-Schreibweise vor? Steht ein Personenname in mehreren Feldern („Anrede“, „Vorname“ und „Nachname“) oder in einem einzigen Feld? Solche Fragen und der Umgang damit werden meist in Form von *Data Dictionaries* abgebildet.

Bei der *Datenstrukturschicht* kann auch von der *Anwendungsschicht* gesprochen werden. Sie definiert logische Entitäten, mit denen die Anwendung arbeitet, wie beispielsweise Kunde, Adresse und Konto. Des Weiteren werden Beziehungen zwischen diesen Entitäten definiert (zum Beispiel: ein Kunde hat mehrere Konten, eine Adresse gehört zu einem Kunden, usw.). Dies entspricht im Wesentlichen der Arbeit mit Entity-Relationship-Modellen und Klassendiagrammen.

Viele Geschäftsszenarien erfordern Transformationen auf mehr als nur einer Ebene. So ist zum Beispiel folgender Geschäftsprozess denkbar: Eine Bestellung aus dem Online-Shop geht als CSV-Datei ein und muss ins XML-Format umgewandelt werden. Dies betrifft die *Datenrepräsentationsschicht*. Anschließend müssen für die Verarbeitung der Bestellung im Warenwirtschaftssystem die Felder „Straße“, „PLZ“ und „Ort“ zu einem Feld „Lieferadresse“ zusammengefasst werden. Das heißt es ist eine strukturelle Transformation in der *Datentypenschicht* notwendig. Eine solche Verkettung von Transformationen resultiert in einer Architektur, wie sie Abbildung 5 zeigt.
Der Vorteil eines Schichtenmodells ist dabei, dass jede Schicht für sich separat betrachtet werden kann, ohne dass man sich dabei um die anderen Schichten kümmern muss.

---

23) Vgl. dazu ausführlich: Hohpe, G. / Woolf, B. et al. (2004), S. 127 ff.

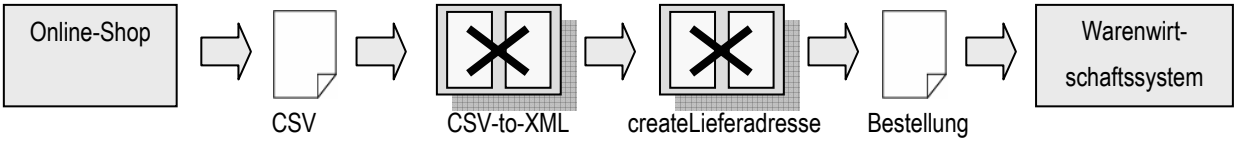


Abb. 5: Verkettung von mehreren Message Translators

Abschließend werden in diesem Kapitel noch zwei dem Message Translator sehr ähnliche Enterprise Integration Patterns kurz vorgestellt werden: *Content Filter* und *Content Enricher*.

Der *Content Filter* ist eine Spezialisierung des Message Translators. Auch hier gibt es eine eingehende und eine ausgehende (veränderte) Nachricht. Der Fokus liegt beim *Content Filter* aber in der Vereinfachung der eingehenden Nachricht. Dies kann zum Beispiel heißen, dass sicherheitskritische Daten aus der Nachricht entfernt werden. Oder die Nachrichtenstruktur wird vereinfacht, indem tief geschachtelte Daten auf die oberste Ebene kopiert werden und unwichtige Daten wegfallen.[24)]
Abbildung 6 zeigt die Darstellung eines Content Filters.

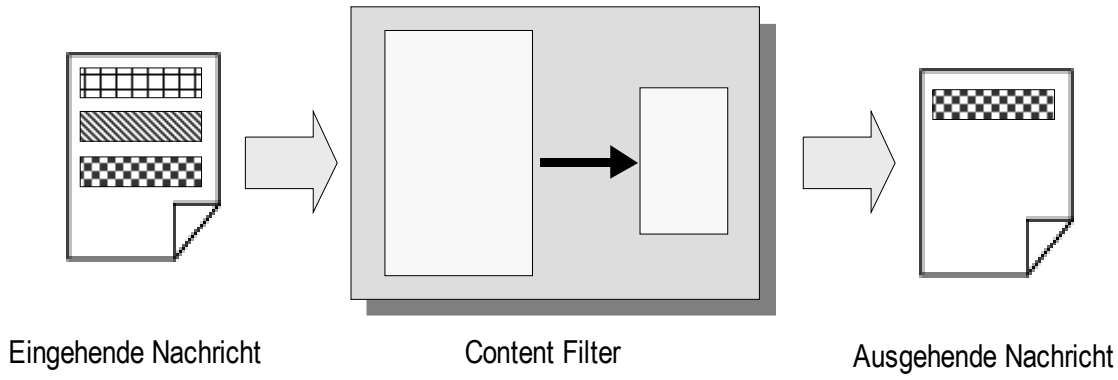


Abb. 6: Der Content Filter[25)]

Das Gegenstück zum Content Filter stellt der *Content Enricher* dar. Dabei sollen keine Daten aus einer Nachricht entfernt werden, vielmehr werden fehlende Daten hinzugefügt. Die eingehende Nachricht enthält in der Regel Informationen, wie die gewünschten Daten gefunden werden können (beispielsweise eine eindeutige ID). Der Content Enricher greift mit diesen Schlüsselinformationen auf eine externe Quelle zu und fordert die gewünschten Daten an. Diese werden dann der ausgehenden Nachricht hinzugefügt.

[24)] Vgl. Druckmüller, B. (2007), S.57 f.

[25)] Mit Änderungen entnommen aus: Hohpe, G. / Woolf, B. et al. (2004), S. 343

Das bedeutet, dass der Content Enricher kein Message Translator im eigentlichen Sinne ist, da die strukturelle Transformation der Nachricht nicht zu seinen Aufgaben gehört. Er übernimmt lediglich die Aufgabe der Datenanreicherung, die Umstrukturierung der vorhandenen Daten ist Aufgabe eines Message Translators.[26] In Abbildung 7 wird der Aufbau eines Content Enrichers dargestellt.

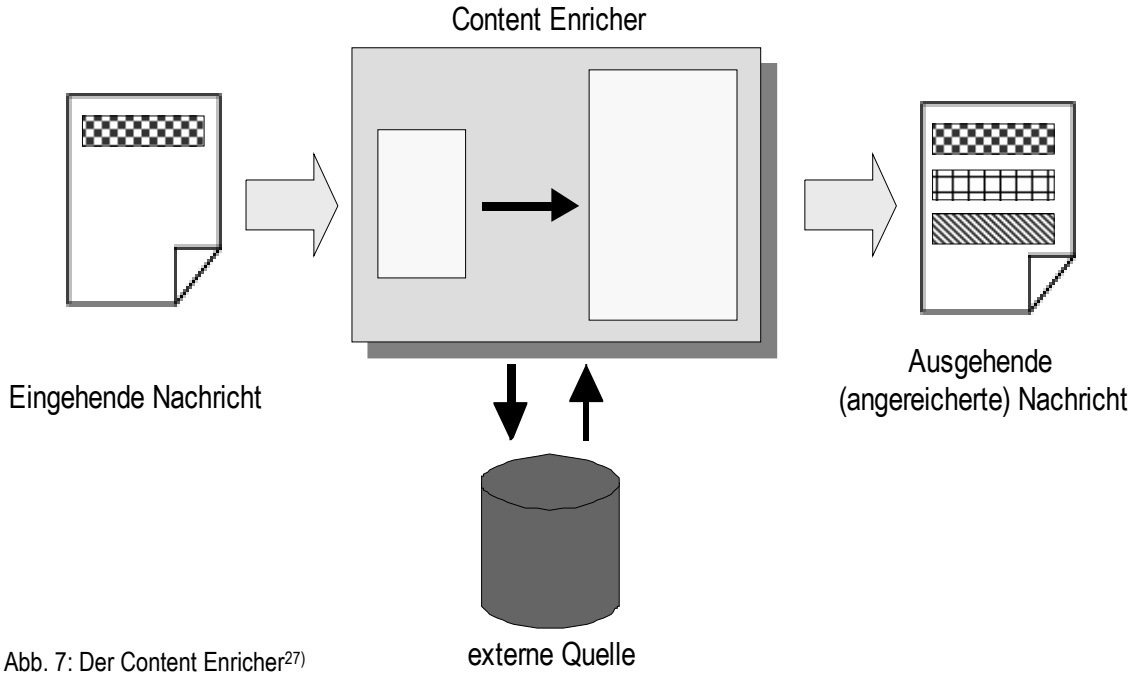


Abb. 7: Der Content Enricher[27]

2.2 Strukturelle Transformation mit XSL Transformation (XSLT)

Wie in Kapitel 1.3 beschrieben, hat sich XML als ein gängiges Nachrichtenformat etabliert. Da die Transformationen von XML-Dokumenten eine häufige Anforderung ist, hat das W3C dafür einen eigenen Standard definiert: Die *eXtensible Stylesheet Language (XSL)*.[28] Besondere Bedeutung kommt dabei dem Bestandteil *XSL Transformation (XSLT)* zu.[29]

In diesem Kapitel soll die strukturelle Transformation von Nachrichten mit XSLT vorgestellt werden. Dies betrifft Transformationen in der *Datentypenschicht*. Dabei kann im Rahmen dieser Arbeit nur eine sehr kurz gehaltene Einführung in XSLT stattfinden, die an einem praktischen Beispiel verdeutlicht wird. Für eine vollständige Referenz sei an dieser Stelle auf entsprechende Fachliteratur hingewiesen.[30]

---

[26] Vgl. Druckmüller, B. (2007), S.60

[27] Mit Änderungen entnommen aus: Hohpe, G. / Woolf, B. et al. (2004), S.339

[28] Vgl. dazu ausführlich: www.w3.org/Style/XSL/

[29] Vgl. dazu ausführlich: www.w3.org/TR/xslt

[30] Vgl. dazu ausführlich: Lenz, E. (2006) und Bongers, F. (2004)

Allgemein formuliert ist XSLT eine XML-Anwendung, die Regeln definiert, anhand derer ein XML-Dokument in ein anderes transformiert wird. Ein *XSLT-Stylesheet* enthält *Vorlagen (Templates)*. Ein *XSLT-Prozessor* vergleicht die Elemente eines Eingabe-XML-Dokuments mit den Templates in einem XSLT-Stylesheet. Wenn er ein passendes Template findet, schreibt er den Inhalt dieses Templates in einen Ausgabebaum. Ist dieser Vorgang beendet, kann der Ausgabebaum weiter in ein XML-Dokument oder ein anderes Format (beispielsweise XHTML oder einfacher Text) überführt werden. [31)]
Das Prinzip einer Transformation mit XSLT wird in Abbildung 8 skizziert.

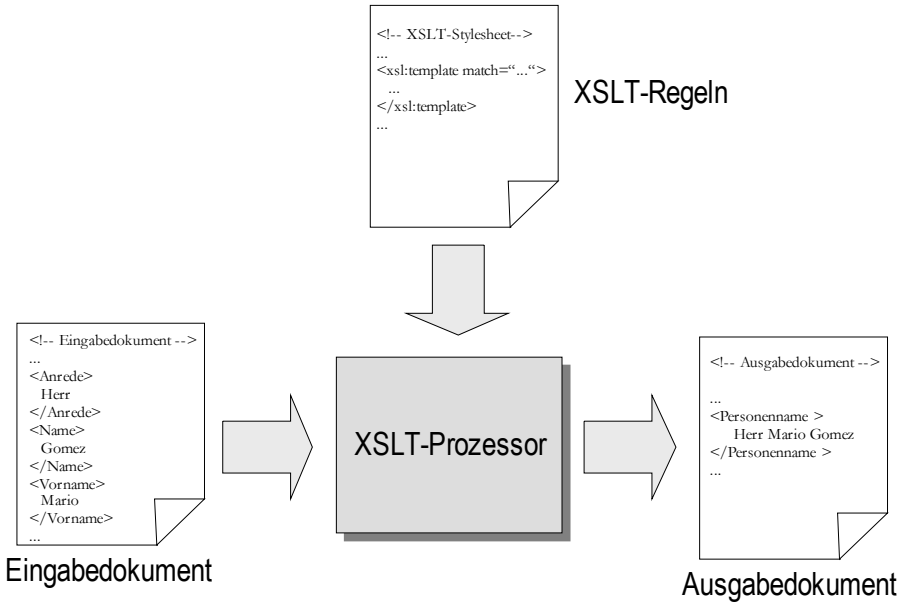


Abb.8: Prinzip der Transformation mittels XSLT [32)]

Angenommen der Message Translator aus Abbildung 5, der unter Anderem eine Lieferadresse aus mehreren Feldern zusammenfügt, soll mit XSLT realisiert werden. Das heißt, die eingehende Nachricht ist bereits im XML-Format und könnte den in Abbildung 9 gezeigten Aufbau haben.

```
<message>
<header>
  <identification messageID="0711"/>
</header>
<body>
  <order no="5AT66X7">
    <article ID="1234">
      <name>MP3-Player</name>
      <price currency="EUR">100.00</price>
      <quantity>1</quantity>
    </article>
```

31) Vgl. Harold E.R. / Means, W.S. (2001), S. 139

32) Mit Änderungen entnommen aus: Lenz, E. (2006), S. 29

```
      <article ID="0851">
       <name>USB-Adapter</name>
       <price currency="EUR">12.00</price>
       <quantity>2</quantity>
     </article>
   </order>
   <purchaser>
     <lastname>Gomez</lastname>
     <firstname>Mario</firstname>
     <salutation>Herr</salutation>
     <firm>VfB Stuttgart 1893 e. V.</firm>
     <street>Mercedesstraße</street>
     <house>109</house>
     <postcode>70372</postcode>
     <city>Stuttgart</city>
     <country>Deutschland</country>
   </purchaser>
  <dateOfPurchase>
    <date>2008-01-30</date>
    <time>17:20</time>
   </dateOfPurchase>
</body>
</message>
```

Abb.9: Eingehende XML-Nachricht

Die ausgehende XML-Nachricht für das das Warenwirtschaftssystem soll dagegen die in Abbildung 10 gezeigte Struktur aufweisen.

```
<nachricht>
    <nachrichtHeader>
        <messageID>0711</messageID>
    </nachrichtHeader>
    <nachrichtBody>
        <Position>
            <Produktnummer>1234</Produktnummer>
            <Produktname>MP3-Player</Produktname>
            <Einzelpreis>100.00 EUR</Einzelpreis>
            <Anzahl>1</Anzahl>
            <GesamtpreisPosition>100.00 EUR</GesamtpreisPosition>
        </Position>
        <Position>
            <Produktnummer>0815</Produktnummer>
            <Produktname>USB-Adapter</Produktname>
            <Einzelpreis>12.00 EUR</Einzelpreis>
            <Anzahl>2</Anzahl>
            <GesamtpreisPosition>24.00 EUR</GesamtpreisPosition>
        </Position>
        <Lieferadresse>
             VfB Stuttgart 1893 e. V.<br/>
             Herr Mario Gomez<br/>
             Mercedesstraße 109<br/>
             70372 Stuttgart<br/>
             Deutschland
        </Lieferadresse>
    </nachrichtBody>
</nachricht>
```

Abb.10: Ausgehende XML-Nachricht

Es ist ersichtlich, dass in der ausgehenden Nachricht andere Tag-Namen verwendet werden, Felder zusammengeführt werden (`Lieferadresse` beinhaltet `lastname`, `firstname`, usw.), neue Tags dazukommen (`GesamtpreisPosition`) und Information wegfallen (`dateOfPurchase`).

Diese strukturellen Veränderungen können über eine Transformation mit dem XSLT-Stylesheet in Abbildung 11 erreicht werden.

```
<xsl:stylesheet version="2.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/message">
    <nachricht>
      <xsl:apply-templates/>
    </nachricht>
  </xsl:template>
  <xsl:template match="header">
    <nachrichtHeader>
      <messageID><xsl:value-of select="identification/@messageID"/></messageID>
    </nachrichtHeader>
  </xsl:template>
  <xsl:template match="body">
    <nachrichtBody>
      <xsl:apply-templates select="order"/>
      <xsl:apply-templates select="purchaser"/>
    </nachrichtBody>
  </xsl:template>
  <xsl:template match="purchaser">
    <Lieferadresse>
      <xsl:value-of select="firm"/><br/>
      <xsl:value-of select="concat(salutation,' ',firstname,' ',lastname)"/><br/>
      <xsl:value-of select="concat(street,' ',house)"/><br/>
      <xsl:value-of select="concat(postcode,' ',city)"/><br/>
      <xsl:value-of select="country"/>
    </Lieferadresse>
  </xsl:template>
  <xsl:template match="order/article">
    <Position>
      <Produktnummer>
        <xsl:value-of select="@ID"/>
      </Produktnummer>
      <Produktname>
        <xsl:value-of select="name"/>
      </Produktname>
      <Einzelpreis>
        <xsl:value-of select="concat(price,' ',price/@currency)"/>
       </Einzelpreis>
      <Anzahl>
        <xsl:value-of select="quantity"/>
      </Anzahl>
      <GesamtpreisPosition>
        <xsl:value-of select="concat(format-number(quantity * price,'#.00'),
                              ' ',price/@currency)"/>
      </GesamtpreisPosition>
    </Position>
  </xsl:template>
</xsl:stylesheet>
```

Abb.11: XSLT-Stylesheet zur Nachrichtentransformation

Ein Template wird durch das Element `xsl:template` mit dem Attribut `match` repräsentiert. Dieses Attribut enthält ein *XPath-Muster*[33], über das die gesuchte Eingabe im XML-Baum gefunden werden kann. Tritt das angegebene Muster auf, wird das Template angewandt, das heißt die im Template definierten Formatierungsanweisungen werden dem Ausgabebaum hinzugefügt.

Über das XSLT-Element `xsl:apply-templates` mit dem optionalen Attribut `select` wird die Verarbeitungsreihenfolge festgelegt.

Um Inhalte des Eingabebaums dem Ausgabebaum hinzuzufügen, kann der Wert mit dem XSLT-Element `xsl:value-of` berechnet werden. Dadurch wird der Text oder der Attributwert eines Elements ermittelt und in die Ausgabe eingefügt. Das Attribut `select` gibt über einen XPath-Ausdruck den Knoten an, dessen Wert ausgelesen werden soll.

Über verschiedene XPath-Funktionen[32] (z.B. `concat`) werden im oberen Beispiel die Felder `salutation`, `firstname`, `lastname`, `street`, `house`, `postcode`, `city` und `country` der eingehenden Nachricht zusammengefasst und in das Element `Lieferadresse` eingefügt.

Das Ausfiltern der Information `dateOfPurchase` entspricht dem im Kapitel 2.1 vorgestellten Prinzip des *Content Filters*, bei dem für das Zielsystem unrelevante Informationen entfernt werden.

Damit das zusätzliche Element `GesamtpreisPosition` hinzugefügt werden kann, bedarf es einer gewissen Programmlogik. Über die mathematischen Funktionen von XPath wird aus der Anzahl (`quantity`) und dem Einzelpreis (`price`) der Gesamtpreis einer Position berechnet. Diese Funktionalität geht bereits über die eigentliche Funktion der Datenumstrukturierung eines Message Translators hinaus.

Abschließend lässt sich feststellen, dass sich XSLT zur Realisierung einer strukturellen Transformation hervorragend eignet. Alle notwendigen Umformungen konnten in dem gezeigten (zugegebenermaßen recht trivialen) Beispiel vorgenommen werden.

---

[33] Vgl. dazu ausführlich: Bongers, F. (2004)

[32] Vgl. Skulschus, M. / Wiederstein, M . (2005), S. 117 ff.

2.3 Microsoft BizTalk Mapper als visuelles Transformationstool

Die Umwandlung mittels XSLT im vorhergehenden Kapitel erfüllt zwar alle Anforderungen eines Message Translators, die Erstellung eines XSLT-Stylesheets und die Einrichtung eines XSLT-Prozessors erfordert aber einige Einarbeitung in das Thema und genauere Kenntnisse der Transformationssprache XSLT. Deshalb stellen die meisten Anbieter von EAI-Lösungen grafische Transformationseditoren zur Verfügung. Diese Tools stellen die Struktur von Eingangs- und Ausgangsformat gegenüber und geben dem Anwender die Möglichkeit, über grafische Assoziationen *Mappings* zwischen den Elementen zu definieren.

BizTalk Server ist eine XML-basierte EAI-Lösung von Microsoft.[33)] Der BizTalk Server dient der Integration, Verwaltung und Automatisierung von Geschäftsprozessen, in dem er Messages abruft, transformiert und zustellt und damit die Kommunikation zwischen verschiedenen Systemen durchführt und überwacht. Die aktuell verfügbare Version ist *Microsoft BizTalk Server 2006 R2*.

Das visuelle Datentransformationstool des BizTalk Servers nennt sich *BizTalk Mapper*. In der aktuellen Version ist der BizTalk Mapper als Modul in die Entwicklungsumgebung *Microsoft Visual Studio* eingebunden.

BizTalk Mapper arbeitet mit *XML Schemata*, das bedeutet über eine grafische Oberfläche können den Schema-Elementen des Quellformats die Schema-Elemente des Zielformats zugeordnet werden. Damit lassen sich Daten (Elemente und Attribute), die von einem Format ins andere übernommen werden sollen, über so genannte *Links* verbinden. Zusätzlich stehen *Functoids* zu Verfügung. Darunter versteht man Transformationselemente, die weitergehende Mappings ermöglichen. Die über 70 standardmäßig bereitgestellten Functoids reichen von Zeichenkettenoperationen über mathematische Funktionen und logische Operatoren bis hin zu Datenbank Functoids.[34)] Zusätzlich gibt es noch die Möglichkeit über *Script Functoids* eigene Subroutinen mit *MS VBScript* zu erstellen.

Die so definierten *Mapping-Regeln* werden vom BizTalk Mapper in XSLT umgewandelt (wobei momentan nur XSLT 1.0 unterstützt wird). Das Tool erlaubt zwischen der grafischen und der textuellen Ansicht des generierten XSLT-Stylesheets umzuschalten.

---

33) Vgl. dazu ausführlich: www.microsoft.com/germany/biztalk

34) Vgl. dazu ausführlich: Probert, D. (2008)

Für das Transformationsbeispiel aus Kapitel 2.2 sieht die Erstellung der Mapping-Regeln mit dem BizTalk Mapper wie in Abbildung 12 gezeigt aus (eine größere Darstellung findet sich in Anlage 1 im Anhang der Arbeit).

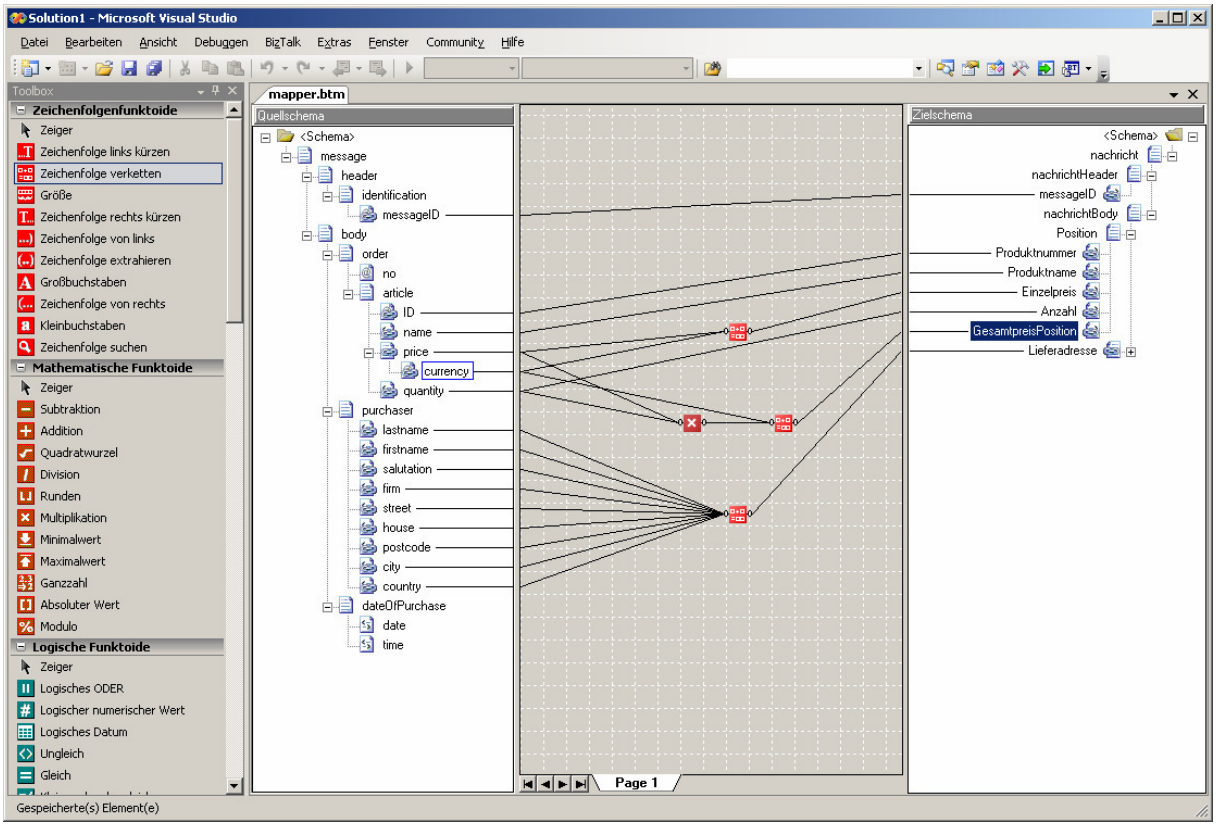


Abb.12: Transformation mit dem Transformationstool BizTalk Mapper

Für die Abbildung von Daten, die direkt übernommen werden können (wie zum Beispiel das Element `price`, dessen Inhalt unverändert in `Einzelpreis` übergeht), werden direkte *Links* erstellt.

Für Zusammenfassungen und Berechnungen werden die *Functoids* benutzt. So kann zum Beispiel die Berechnung von `GesamtpreisPosition` über die zwei *Functoids* „Multiplikation“ und „Zeichenfolge verketten“ abgebildet werden. Zuerst werden die Elemente `price` und `quantity` multipliziert, um anschließend das Attribut `currency` anzuhängen. Durch diese Verkettung ergibt sich die Mapping-Regel wie sie im Bildausschnitt in Abbildung 13 zu sehen ist.

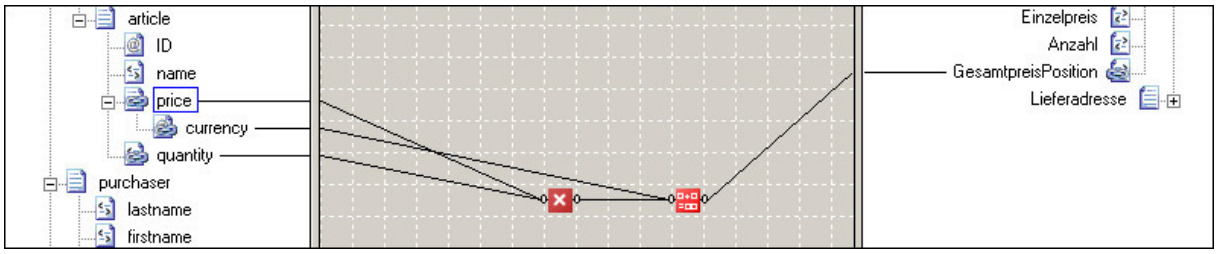


Abb.13: Verkettung von Functoids zur Berechnung von GesamtpreisPosition

Die Konfiguration eines Functoids zeigt Abbildung 14 exemplarisch für die mathematische Operation „Multiplikation“.

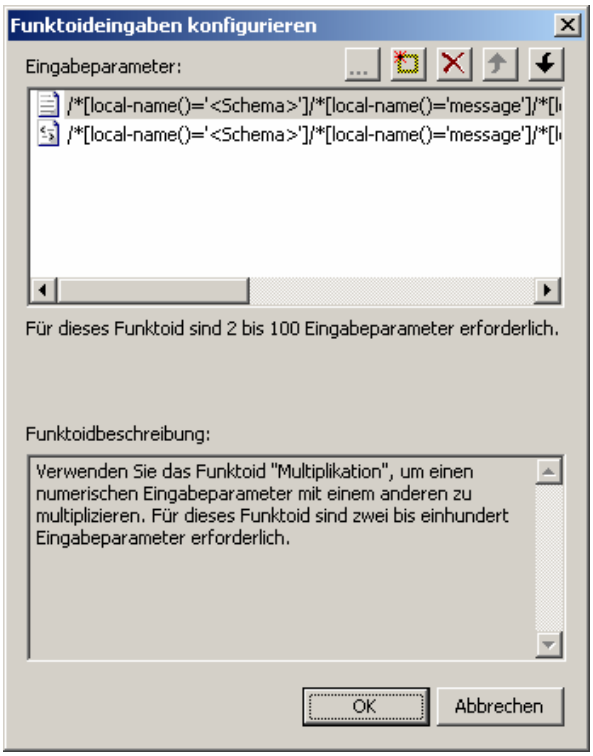


Abb.14: Konfiguration des Functoids „Multiplikation“

Bei der Konfiguration des Functoids „Multiplikation“ können zwischen zwei und einhundert Eingabeparameter angegeben werden. Diese Parameter entsprechen den Elementen im Baum des XML-Schemas.

Es wird klar ersichtlich, dass diese visuelle Form der Festlegung von Transformationsregeln im Vergleich zur manuellen Definition eines XSLT-Stylesheets schneller und einfacher durchzuführen ist. Die Einarbeitungszeit ist entschieden kürzer und die Definition von Transformationsregeln kann auch von „Technik-Laien“ durchgeführt werden, selbst wenn komplexere Transformationen wie in dem gezeigten Beispiel notwendig werden.

Fazit

Diese Arbeit beleuchtete das Thema *Messaging* als Instrument zur Kommunikation in heterogenen Unternehmensnetzen aus dem Blickwinkel der Enteprise Application Integration (EAI). Dabei wurde die Notwendigkeit aber auch die Möglichkeiten der *Message Transformation* aufgezeigt.

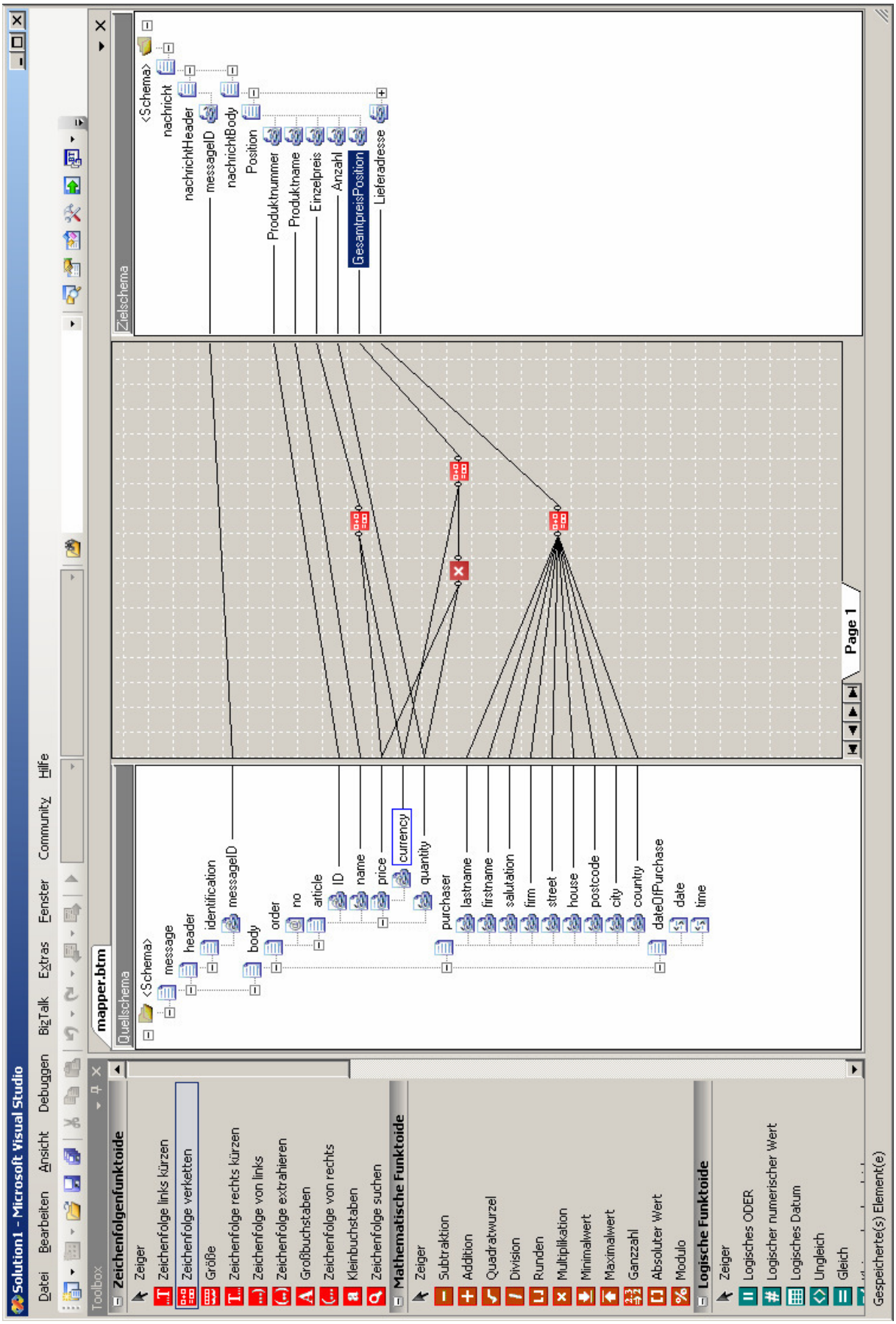
Für das grundsätzliche Verständnis der strukturellen Transformation von Nachrichten ist die Arbeit mit XSLT-Stylesheets sicherlich sehr hilfreich, im produktiven Einsatz jedoch können visuelle Tools wie der *Contivo Mapper* [34)] oder der in Kapitel 2.3 vorgestellte BizTalk Mapper wertvolle Dienste leisten.

[35)] Vgl. dazu ausführlich: www.contivo.com/products/mapper.html

Anhang

Anlagenverzeichnis

Anlage 1: Transformation mit dem Transformationstool BizTalk Mapper



Literaturverzeichnis

Conrad, S. et al. (2005)	Enterprise Application Integration Grundlagen - Konzepte - Entwurfsmuster – Praxisbeispiele Spektrum Akademischer Verlag 2005
Bator, F. (2006)	Enterprise Application Integration http://www.devmatic-it.com/article_en.jsf?article=eai 19.02.2008
Bongers, F. (2004)	XSLT 2.0 und XPath 2.0 Galileo Press-Verlag
Druckenmüller, B. (2007)	Parametrisierung von EAI Patterns http://elib.uni-stuttgart.de/opus/volltexte/2007/3049/ 08.02.2008
Herden, S. / Gomez, J. et al. (2006)	Software-Architekturen für das E-Business: Enterprise-Application-Integration mit Verteilten Systemen Springer-Verlag 2006
Harold E.R. / Means, W.S. (2001)	XML in a Nutshell Deutsche Ausgabe O'Reilly Verlag 2001
Hohpe, G. / Woolf, B. et al. (2004)	Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions Addison-Wesley Verlag 2004
Keller, W. (2002)	Enterprise Application Integration Erfahrungen aus der Praxis Dpunkt-Verlag 2002
Linthicum, D. (2000), S.3	Enterprise Application Integration 3. Auflage Addison-Wesley Verlag 2000

Lebender, M. / Ondrusch, N. et al. (2003)	Business Integration Software - Werkzeuge, Anbieter, Lösungen Fraunhofer-Institut für Arbeitswirtschaft und Organisation IAO, Stuttgart 2003 http://www.ebi.iao.fraunhofer.de/docs/ BusinessIntegrationSoftware_MediaVision.pdf 21.02.2008
Lenz, E. (2006)	XSLT 1.0 – kurz & gut Deutsche Ausgabe O'Reilly Verlag 2006
Probert, D. (2008)	Understanding the BizTalk Mapper: Part 2 - Functoids Overview http://www.bizbert.com/bizbert/2008/02/08/Understanding +The+BizTalk+Mapper+Part+2+Functoids+Overview.aspx 28.02.2008
Siepmann, F. (2005)	Message Oriented Middleware am Beispiel von XMLBlaster http://www.techfak.uni-bielefeld.de/~swrede/ xml-isy/ talks/ mom-xmlblaster.pdf 10.02.2008
Skulschus, M. / Wiederstein, M . (2005)	XSLT 2.0 – Fortgeschrittene Anwendungen mitp-Verlag 2005
Weitzel, T. / Harder, T / Buxmann, P. (2001)	Electronic Business und EDI mit XML Dpunkt-Verlag 2001
W3C (Hrsg.) (2007)	SOAP Version 1.2 Part 0: Primer (Second Edition) http://www.w3.org/TR/soap12-part0/ 10.02.2008