Ein Konzept zur Spezifikation von XSL-Transformationen und dessen Anwendung bei Bankselbstbedienungssystemen

R. DEPKE[1], M. LANGHAM[2], B. LÜTKEMEIER[1], S. THÖNE[1]
[1]Universität Paderborn, Fachbereich 17, Warburger Str. 100, 33098 Paderborn, {depke,blue,seb}@upb.de
[2]s&n AG, Klingenderstr. 5, 33100 Paderborn, mlangham@sundn.de

ZUSAMMENFASSUNG
Die Transformation von XML-Dokumenten mit XSLT-Regeln ist von großer praktischer Relevanz. Der Aufwand für die Erstellung der Regeln wächst allerdings mit der Größe der ineinander zu überführenden Dokumentformate und der Komplexität der Transformation. Mit der wachsenden Menge von Regeln wird die Übersetzung auch immer weniger nachvollziehbar. Die in diesem Artikel vorgeschlagene Spezifikation von XSL-Transformationen und die nachfolgende Generierung von XSLT-Regeln vereinfacht die Erstellung von Übersetzern und bildet gleichfalls eine besser verständliche Dokumentation der Übersetzungsvorschrift. Die praktische Relevanz des Verfahrens zeigt sich zum Beispiel bei der Konvertierung von Austauschformaten in Bankselbstbedienungssystemen.

1 EINLEITUNG

Die *extensible markup language* (XML) setzt sich zur Verarbeitung strukturierter Daten immer mehr durch [1]. XML ist ein Standard zur Festlegung von XML-Sprachen. Diese selbst sind kontextfreie Auszeichnungssprachen, die durch Grammatiken definiert werden. Jede Grammatikdefinition ist in Form einer *document type definition* (DTD) gegeben.
Das folgende Beispiel illustriert diese Begriffe. In Bild 1 ist ein XML-Dokument zu sehen, das Inhalte einer Nachricht in strukturierter Form enthält. Die Verschachtelung der Elemente Nachricht, Typ, Transaktion, Betrag und Waehrung wird durch die Einrückung deutlich.

```
<Nachricht>
   <Typisierung>
      <Typ>
         Überweisung
      </Typ>
   </Typisierung>
   <Transaktion>
      <Betrag>
         100
      </Betrag>
      <Waehrung>
        EUR
      </Waehrung >
   </Transaktion>
</Nachricht>
```

Bild 1: XML-Dokument einer Nachricht

Diese baumartige Struktur wird in einer DTD festgelegt, siehe Bild 2. Ein solches Dokument enthält kontextfreie Produktionsregeln der Grammatik, die aus einem Schlüsselwort ELEMENT, gefolgt von einem nicht terminalen Elementbezeichner und der rechten Seite der Produktionsregel bestehen.

```
<?xml version=„1.0“ encoding=“UTF-8“?>
<!ELEMENT Nachricht (Typ Transaktion)>
<!ELEMENT Typ #PCDATA>
<!ELEMENT Transaktion (Betrag Waehrung)>
<!ELEMENT Betrag (#PCDATA)>
<!ELEMENT Waehrung (#PCDATA)>
```

Bild 2: DTD für XML-Dokument in Bild 1

Ein XML-Parser erzeugt aus einem XML-Dokument einen Baum, auf dem über eine standardisierte Schnittstelle, das *document object model* (DOM), operiert werden kann.

Vielfach ist es notwendig, ein XML-Dokument in ein anderes XML-Dokument zu konvertieren. Für diese Zwecke ist eine eigene Sprache definiert worden. Das W3-Konsortium hat die Sprache *eXtensible Stylesheet Language (XSL)* definiert, die ursprünglich allein für die Erzeugung der HTML-Präsentation von XML-Dokumenten eingesetzt werden sollte. Die Sprache XSL besteht heute aus den Teilen *XSLT (XSL Transformations)* und *XSL-FO (formatting objects)*, die zwei unabhängige XML-Anwendungen sind [2]. Transformationen von einer XML-Anwendung nach HTML oder zwischen zwei XML-Sprachen werden durch XSLT-Regeln beschrieben. Die Sprache XSL-FO

soll in Zukunft HTML zur Beschreibung der Präsentation von XML-Elementen ersetzen.

Mit XSLT lassen sich unterschiedlich strukturierte XML-Dokumentformate ineinander überführen. Man kann zwar Elemente umordnen, weglassen oder hinzufügen, nicht aber vorhandene Elementinhalte verändern. Dadurch bleibt die Konsistenz und Integrität der Daten bei XSL-Transformationen erhalten [3]. Für den Transformationsteil XSLT der Sprache XSL hat das W3-Konsortium seit November 1999 eine erste Version standardisiert [4]. Damit existiert eine zuverlässige Basis für die Entwicklung von XSL-Werkzeugen.

Die Transformation von XML-Dokumenten auf der Basis von XSLT ist für komplexe Dokumente und Abbildungen aufwendig, wodurch die praktische Verwendung in Frage gestellt ist. XSLT-Dokumente sind Sammlungen von Regeln und damit immer weniger verständlich je gößer sie werden. Dadurch ist die Wartbarkeit von zugehörigen Anwendungen beeinträchtigt.

Unser Ziel besteht deshalb darin, XSL-Transformationen in deklarativer Form zu spezifizieren, um die Entwicklung derartiger Transformationen zu erleichtern und gleichzeitig eine bessere Dokumentation zu gewinnen, als sie XSLT-Dokumente darstellen. Aus einer Spezifikation soll leicht ein XSLT-Dokument erzeugt werden können.

Im folgenden Abschnitt 2 werden wir genauer auf die Sprache XSLT eingehen und die Struktur der Regeln erklären. Ein Anwendungsbeispiel aus dem Bereich der Bankensoftware soll in Abschnitt 3 die praktische Relevanz von XSLT zeigen. In Abschnitt 4 gehen wir auf die Spezifikation und Generierung von XSL-Transfor-mationen im Fall der Umstrukturierung von Dokumenten ein. Wir diskutieren in einem Ausblick, wie komplexere XSL-Transformationen in Zukunft spezifiziert werden können.

2 XSLT

Die Transformation von XML-Dokumenten mit XSLT wird von einem *XSL-Prozessor* durchgeführt, der die Sprache XSLT implementieren muß. Als Eingabe erhält dieser Prozessor das zu transformierende XML-Dokument sowie ein sogenanntes *XSL-Stylesheet* Dokument. Die Bezeichnung Stylesheet ist aus dem historischen Kontext entstanden, als XSL vor allem noch für Formatierungsanweisungen gedacht war und zur Transformation nach HTML benutzt wurde. In diesem Stylesheet stehen die Regeln, nach denen der Prozessor das neue XML-Dokument aufbauen soll. Die Anweisungen für die XML-zu-XML-Übersetzung folgen dabei der Syntax von XSLT. Die Stylesheets sind ebenfalls XML-Dokumente und die für XSLT definierten Elemente bilden den XSL-Namensraum, der bei der Formulierung der XSL-Regeln benutzt werden muß. Wohlgeformte XML-Dokumente werden durch XML-Parser in Bäume (DOM - Document Object Model) umgewandelt. Nachdem ein XSL-Prozessor aus der Eingabe einen solchen Baum *(source tree)* erzeugt hat, wandelt er ihn nach den angegebenen Regeln in einen anderen XML-Baum *(result tree)* um. Für diese *Baumtransformation* gibt es Operatoren zur Selektion bestimmter Knoten, zu deren Umordnung, Neustrukturierung und anschließender Ausgabe als neues XML-Dokument. Dabei wird die Baumstruktur ausgenutzt und Knoten, die selbst wieder Nachfolgeknoten haben, sind ihrerseits Bäume, die der entsprechenden Transformation unterworfen werden.

Die XSLT-Regeln bestehen aus zwei Teilen, den *Mustern (patterns)* und den *Schablonen (templates)*. Der XSL-Prozessor versucht, im Quellbaum einen Knoten zu finden, der dem angegebenen Muster entspricht. Dazu traversiert er den Baum mit seinen Teilbäumen und vergleicht alle Knoten und Unterbäume mit der entsprechenden Struktur des Musters. Elemente, die dem Muster nicht genügen, werden übergangen und nicht bearbeitet. Wird der Prozessor jedoch fündig, so bearbeitet er den Knoten und den eventuell damit verbundenen Teilbaum. Die darin befindlichen Daten werden entsprechend der Schablone umstrukturiert und in ein neues XML-Format gebracht. Dabei ist man mit den Templates in der Lage, komplexe Strukturen zu erzeugen. Die ursprünglichen Daten können kopiert, aber auch um neue Daten und weitere Marken/Tags ergänzt werden. Handelt es sich bei dem gefundenen Knoten seinerseits um die Wurzel eines Teilbaums, so kann dieser Baum analog bearbeitet werden, indem man eine weitere Regel anwendet und nach einem neuen Muster sucht. Diese Suche erfolgt dann nur in dem Teilbaum, der sich unter dem gerade bearbeiteten Knoten befindet.

Durch die beiden XSLT-Regeln in Bild 3 wird das verschachtelte XML-Dokument in Bild 1 in ein flaches Dokument (Bild 4) umgeformt. Das Element `xsl:template` enthält im Attribut match das Muster, das im Quelldokument gesucht werden soll. Der Inhalt des Elementes xsl:template besteht aus der Schablone, die im Zieldokument erzeugt wird. In der Schablone sind XSLT-Elemente enthalten, die die weitere Übersetzung von Elementen (`xsl:apply-templates`) oder das Einfügen von terminalen Inhalten (xsl:value-of) bewirken.

```
<xsl:template match="Nachricht">
<MsgFlat>
  <Type>
    <xsl:value-of select="Typ"/>
  </Type>
  <xsl:apply-templates select="Transaction"/>
</MsgFlat>
</xsl:template>

<xsl:template match="Transaktion">
<Amount>
  <xsl:value-of select="Betrag"/>
</Amount>
<Currency>
  <xsl:value-of select="Waehrung"/>
</Currency>
</xsl:template>
```

Bild 3: Zwei XSLT-Regeln zur Transformation des XML-Dokumentes in Bild 1

```
<MsgFlat>
   <Type>
      Überweisung
   </Type>
   <Amount>
     100
   </Amount>
   <Currency>
     EUR
   </Currency >
</MsgFlat>
```

Bild 4: Flaches XML-Dokument einer Nachricht

Im praktischen Einsatz gibt es drei Wege, nach denen XSL Stylesheets für die Transformation von XML-Daten verwendet werden können. Einerseits ist es möglich, die XML-Dokumente zusammen mit dem Stylesheet zu verschicken und die Transformation vom Empfänger vornehmen zu lassen. In einer Client-Server-Architektur läge dabei die Übersetzungsaufgabe beim Client, im Internet also beim Web-Browser. Die Konvertierung kann aber auch komplett vom Server vor der Übertragung durchgeführt werden. Als dritte Möglichkeit kann eine unabhängige Komponente die Umwandlung vornehmen, bevor die Daten auf einem Server zur Verfügung gestellt werden [3].

3 FALLSTUDIE: KONVERSION VON DATENAUSTAUSCHFORMATEN BEI BANKSELBSTBEDIENUNGSSYSTEMEN

Im Bankenbereich werden zunehmend Automaten und Geräte zur Selbstbedienung (SB) eingesetzt, über die Kunden die finanzwirtschaftlichen Dienstleistungen der Banken eigenständig in Anspruch nehmen können. Der Kunde bekommt Bargeld am Geldautomaten oder Kontostandsinformationen am Kontoauszugsdrucker. An sogenannten Ladeterminals kann er Beträge auf elektronische Zahlungsmittel wie zum Beispiel die „Geldkarte“ buchen, und SB-Terminals bieten ein umfangreiches Informations- und Dienstleistungsangebot von Überweisungen bis hin zu individuellen Beratungen. Für die Bereitstellung dieser Dienstleistungen müssen Banken eine leistungsfähige *informationstechnische Infrastruktur* aufbauen, mit der alle anfallenden Daten für die einzelnen Vorgänge aufgenommen, an operative Stellen zur Autorisierung, Buchung u.ä. übertragen und dort verarbeitet werden. Die eigentliche Datenverarbeitung kann meist nicht in den SB-Geräten selbst vorgenommen werden, da die Kundendaten, auf denen operiert werden soll, in der Regel in zentralen Datenbanksystemen gespeichert sind. Diese Datenbanken, die datenverarbeitenden Komponenten und die verteilten SB-Terminals werden durch Datenübertragungsnetze miteinander gekoppelt. Sie bilden die Infrastruktur eines Bankselbstbedienungssystems (*SB-System*), dessen Architektur im folgenden Abschnitt kurz beschrieben werden soll.

3.1 Architektur von Bankselbstbedienungssystemen

Die Architektur (Bild 5) von Bankselbstbedienungssystemen ist heute oft dezentral und resultiert aus (neuen) Geschäftsvorgängen, die nur auf Filialebene der Kreditinstitute relevant sind. Dazu zählen regional begrenzte Vorgänge und Angebote wie zum Beispiel Immobiliengeschäfte. Auch die Pflege der Kundenbeziehungen (*Customer Relationship Management, CRM*) soll nicht über ein räumlich weit entferntes Rechenzentrum abgewickelt werden. Deshalb wird auf lokaler Ebene zwischen den vielen SB-Terminals und dem zentralen Rechenzentrum eine zusätzliche Komponente, der SB-Server, als Zwischenschicht und Sammelknoten eingefügt.

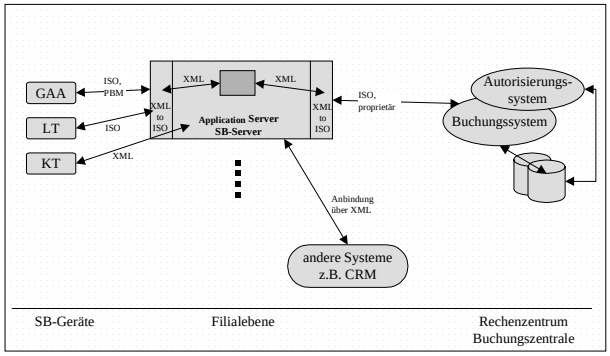


Bild 5: Dezentrale Architektur mit SB-Servern

Vorgänge, die weiterhin im zentralen Buchungssystem verarbeitet werden sollen, werden vom SB-Server an dieses weitergeleitet. Mit diesem Konzept kann der Umfang der angebotenen SB-Vorgänge um komplexere Dienste erweitert werden, die vom SB-Server in einzelne Teilaufgaben zerlegt und an die zuständigen Anwendungsprogramme verteilt werden. Dabei ist auch eine Kombination von Anwendungen im Buchungssystem mit den Applikationen des SB-Servers möglich.

Es ergibt sich damit die Notwendigkeit, daß die Anwendungen Zwischenergebnisse untereinander weiterreichen können, insbesondere wenn eine Bearbeitung in mehreren Schritten über verschiedene Komponenten erfolgen soll. Der SB-Server muß also intern mit einem flexiblen Datenaustauschkonzept ausgestattet sein. Hier bietet sich der Einsatz der XML-Technologie an, weil sie eine flexible Strukturierung der Daten und somit auch eine Abbildung komplexerer Datenkonstrukte erlaubt.

Da die zentralen Rechenzentren weiterhin Daten nach den konventionellen binären Protokollen erwarten, muß ein SB-Server bei der Kommunikation mit dem Buchungssystem seine Daten in den Formaten der vereinbarten Protokolle versenden und empfangen. Auch die SB-Geräte benutzen die bekannten Protokolle, die vor allem bei Geldausgabeautomaten (GAA) und Ladeterminals (LT) aus Sicherheitsgründen bisher noch verpflichtend vorgeschrieben sind. Einige dieser Protokolle waren ursprünglich auch nicht für

Selbstbedienungsgeräte konzipiert und berücksichtigen deren Eigenschaften teilweise nur durch proprietäre Erweiterungen. Zudem ist ein Format wie ISO 8583 (vgl. [5]) auf Binärinformationen aufgebaut und dadurch nur schwer zu interpretieren. Die vorhandene Software, die diese Formate verarbeiten kann, ist zum Teil sehr alt und lässt sich nur mit Aufwand in moderne Softwarearchitekturen integrieren.

Soll also im SB-Server intern mit XML als Datenaustauschformat gearbeitet werden, sind am Eingang und Ausgang des SB-Servers Schnittstellen zu realisieren, die die Konvertierung zwischen den Protokollformaten und XML vornehmen (vgl. Bild 5). Mit XML werden die teilweise proprietären Bankenformate durch klar strukturierte und lesbare Formate ersetzt werden. Die definierten XML-Schemata lassen sich veröffentlichen, um hierdurch eine möglichst große Unterstützung innerhalb der Industrie zu erlangen.

3.2 Konzept zur Transformation

Lütkemeier und Thöne haben ein zweiphasiges Konzept für die Transformation von (binären) Protokollen in strukturierte XML-Dokumente vorgestellt und implementiert [6], das von der Firma s&n AG in Paderborn zur Entwicklung von Software für Bankselbst-bedienungssystemen genutzt wird.

In der ersten Phase der Transformation wird aus einem (binären) Ausgangsformat ein flaches XML-Dokument, respektive ein DOM-Baum erzeugt (Bild 6). Die Aufteilung in zwei Phasen erlaubt, daß für andere Protokolle ebenfalls zuerst ein (flaches) XML-Dokument erzeugt wird. Das in der ersten Phase erzeugte, transiente XML-Dokument wird im nächsten Schritt unter Verwendung von XSLT in das XML-Zielformat transformiert. Da kein endgültiges Zielformat existiert, bietet XSLT den Vorteil, durch Anpassungen der Stylesheets flexibel auf neue Vorgaben bezüglich des Zielformats reagieren zu können.

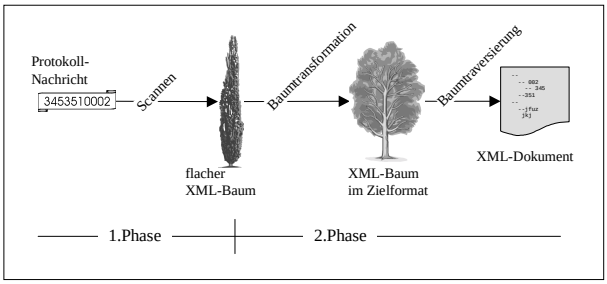


Bild 6: Die zwei Phasen der Transformation und ihre Ergebnisse

In der ersten Phase werden die Daten aus dem Protokoll eingelesen und in der internen Datenstruktur, einem DOM-Baum, abgelegt (vgl. Bild 6). Aus den Inhalten der Nachricht, die meist sequentiell eingelesen werden, muß also zunächst ein DOM-Baum aufgebaut werden. Die Tiefe des Baumes ist in diesem Zwischenergebnis sehr gering, da die Daten aus dem Protokoll noch nicht weiter strukturiert werden, sondern einfach in der sequentiellen Abfolge ihres Auftretens als neue Geschwisterknoten in den Baum eingehängt werden. Die Elemente und Marken in dem flachen DOM-Baum sind in ihrer Bezeichnung noch sehr stark an das Protokoll und die darin definierten Datenelemente gebunden. Damit ist dieser flache Baum eine direkte XML-Repräsentation der Nachricht aus dem Protokollformat, auf der die eigentliche Transformation in das gewünschte Zielformat noch durchzuführen ist.

In der zweiten Phase wird der so erzeugte flache XML-Baum in das eigentliche Zielformat umgewandelt. Dabei müssen die Daten, die bisher fast alle als einzelne Blätter direkt am Wurzelknoten hängen, zu Teilbäumen aggregiert werden. Diese Teilbäume können dann ihrerseits gruppiert und umstrukturiert werden bis schließlich die gewünschte Zielstruktur vorliegt. Dabei sind Umbenennungen und andere Manipulationen der einzelnen Elemente nötig. Aus der Nachricht in der endgültigen Fassung als DOM-Baum wird ein XML-Dokument als Ausgabe erzeugt.

Bei der Rücktransformation von XML zum herkömmlichen Protokoll sind die gleichen zwei Phasen zu durchlaufen, allerdings in umgekehrter Reihenfolge: Es wird zunächst die eingehende XML-Nachricht in eine interne DOM-Struktur eingelesen und danach in einem ersten Schritt in einen flachen XML-Baum umgeformt. Dieser Baum kann aufgrund seiner geringen Tiefe leicht in eine Folge von Datenelementen zerlegt werden, die dann entsprechend der Protokoll-spezifikationen sequentiell angeordnet und ausgegeben werden.

4 TRANSFORMATION VON XML-DOKUMENTEN MIT XSLT

Die Zuordnung von Blattpositionen von gleichen Elementen in zwei korrespondierenden XML-Dokumenten bildet eine deklarative Spezifikation der Transformation, siehe Bild 7. Der Effekt einer solchen Transformation ist die Umstrukturierung eines XML-Dokumentes mit eventueller Änderung der Elementbezeichnungen. Die Inhalte können nicht verändert werden, sie können höchsten gelöscht werden, indem das entsprechende Element nicht abgebildet wird.

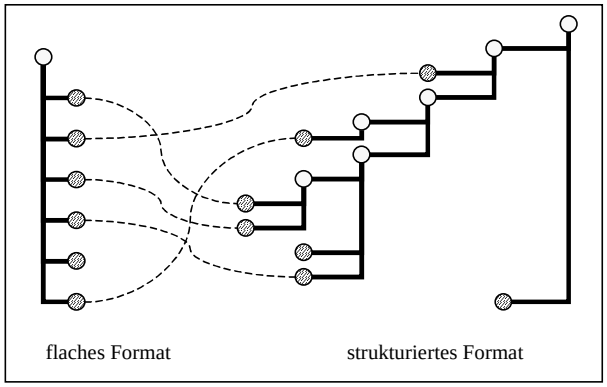


Bild 7: Zuordnung von Blattpositionen zweier XML-Dokumente für die Transformation

Krishnamurthi et al. schlagen zur Spezifikation derartiger Abbildungen die XML-Sprache XT3D vor [7]. XT3D-Dokumente enthalten sowohl ein Eingabe- als auch ein Ausgabedokument ohne Inhalte. An die Stelle der Inhalte werden XT3D-Elemente gesetzt, die im Eingabeteildokument Namen von Elementen und Attributen als Eingabemuster für die Transformation markieren. Das Ausgabeteildokument enthält ebenfalls spezielle XT3D-Elemente als Inhalt, wodurch Bildelemente von markierten Eingabeelementen festgelegt werden. Realisiert wird die Abbildung durch einen in der funktionalen Sprache Scheme implementierten Interpreter für XT3D-Spezifikationen. Damit stellt dieser Ansatz eine von XSLT unabhängige Alternative für die Transformation von XML-Dokumenten dar.

Falls ein XSL-Prozessor verwendet werden soll, müssen allerdings aus einer Spezifikation XSL-Regeln erzeugt werden. Dadurch wird die Implementation eines eigenen Prozessors für die Spezifikationssprache eingespart. Bei diesem Ansatz wird ein XML-Spezifikationsdokument in ein XSLT-Dokument transformiert un die Interpretergeneration ist selbst eine Transformation. In den nächsten beiden Unterabschnitten soll die Spezifikation und Generierung des zugehörigen XSLT-Dokumentes für das Problem der Restrukturierung von Dokumenten gezeigt werden. In Abschnitt 5 diskutieren wir kurz ein erweitertes Konzept.

4.1 Spezifikation der XSL-Transformation

Ein XML-Dokument, das alle gemäß der DTD möglichen Elemente tatsächlich enthält, sei im folgenden *vollständig* genannt. Ausgeschlossen vom nachfolgend beschriebenen Verfahren sind XML-Dokumente, die nicht in der Länge beschränkt sind. Für die Spezifikation der Transformation werden ein vollständiges flaches und ein vollständiges geschachteltes (strukturiertes) Dokument benötigt. In jedem Blatt des strukturierten Dokuments bildet die Pfadangabe des zugehörigen Blatts im flachen Schema den Inhalt. Die dadurch definierte Abbildungsvorschrift muß injektiv sein, damit die Abbildung wie gewünscht invertierbar ist. Zwei verschiedene Baumpfade dürfen daher nicht auf dasselbe Blatt abgebildet werden. Die Abbildung kann partiell sein, da es durchaus vorstellbar ist, daß bestimmte Dateninhalte bei der Transformation nicht ins Ziel übernommen werden sollen. Bild 8 zeigt einen Auszug aus einer Eingabedatei für den Generator.

```
<Nachricht>
   <Typisierung>
      <Typ>
         /ISO8583Msg/MsgType
      </Typ>
   </Typisierung>
   <Transaktion>
      <Betrag>
         /ISO8583Msg/AmountTransaction
      </Betrag>
   </Transaktion>
</Nachricht>
```

Bild 8: Auszug aus einer strukturierten Eingabedatei für den Generator

Hier wird festgelegt, daß die beiden Blätter mit den Pfaden */Nachricht/Typisierung/Typ* und */ISO8583Msg/MsgType* bzw. */Nachricht/Transaktion/Betrag* und */ISO8583Msg/AmountTransaction* einander zugeordnet sind. Damit ist die Abbildung bereits in beiden Richtungen definiert, da man für einen gegebenen flachen Pfad nur die Blattposition im strukturierten Dokument finden muß, an der er aufgeführt ist. Deren Pfad ist dann der gesuchte. Wie man diese Position effizient findet, wird im folgenden Abschnitt erklärt. Also muß das flache Eingabedokument keinerlei Abbildungsinformationen mehr beinhalten. Es wird dem Generator lediglich für die Generierung des XSLT-Dokuments für die Rückrichtung übergeben. Bild 9 zeigt einen Auszug aus einer solchen zweiten Eingabedatei.

```
<ISO8583Msg>
   <MsgType>
   </MsgType>
   <AmountTransaction>
   </AmountTransaction>
</ISO8583Msg>
```

Bild 9: Auszug aus einer flachen Eingabedatei für den Generator

4.2 Generierung von XSLT-Regeln

Der Generator erzeugt zunächst die XSL-Regeln für die Transformation vom flachen in das geschachtelte XML-Dokument. Dazu muß er das vollständige, strukturierte Dokument, das die Abbildung beinhaltet, parsieren, um einen DOM-Baum zu erhalten. Nachdem der Rahmen des XSL-Stylesheets, der den XSL-Namensraum definiert und das Wurzelelement <xsl:stylesheet> beinhaltet, produziert ist, beginnt der Generator mit der Erzeugung der Hauptregel. Dazu führt er eine Tiefensuche auf dem strukturierten Baum durch und traversiert ihn komplett. Für jeden inneren Knoten kopiert er dabei Start- und Endtag in die Hauptregel. Im obigen Beispiel würde also zunächst folgendes in der XSL-Datei erzeugt:

```
<xsl:stylesheet version="1.0" xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform">
<xsl:template match="ISO8583Msg">
<Nachricht>
   <Typisierung>
```

Trifft der Parser auf ein Blatt, so liest er dessen Inhalt und erzeugt eine apply-template-Anweisung, deren Muster der Blattinhalt ist, welcher den Pfad im flachen Schema angibt. Außerdem fügt er eine zusätzliche Regel am Ende des Dokuments ein, die dasselbe Muster bekommt, das Start- und Endtag des Blattes einfügt, und eine value-of-Anweisung beinhaltet, die den späteren Dateninhalt kopiert:

In der Hauptregel:
```
<xsl:apply-templates
select="/ISO8583Msg/MsgType"/>
```

Als zusätzliche Regel:
```
<xsl:template match="/ISO8583Msg/MsgType">
<Typ>
       <xsl:apply-templates select="@*"/>
       <xsl:value-of select="."/>
</Typ>
</xsl:template>
```

Gleichzeitig fügt der Generator für jedes Blatt in eine Hashtabelle ein neues Wertepaar ein, das die Zuordnung der beiden Blattpositionen bestimmt. Als Schlüssel wird der Inhalt des Blattes, also der flache Baumpfad übergeben, zugehöriger Wert ist der strukturierte Pfad dieses Blattes, der dem Generator durch die Tiefensuche bekannt ist. Diese Hashtabelle wird für die XSL-Regeln zur Transformation des verschachtelten in das flache Format benötigt.

Die XSL-Regeln für die Rücktransformation generiert. Dazu parst der Generator das vollständige flache Dokument, das ihm als zusätzliche Eingabe zur Verfügung steht. Analog zum Hinweg führt er nun eine Tiefensuche durch, während der das Stylesheet Schritt für Schritt aufgebaut wird. Wenn er auf ein Blatt trifft, benötigt er den zugehörigen Pfad im strukturierten Dokument, um ihn als Muster in der apply-Anweisung und der zusätzlichen Regel einzusetzen. Da bei der Erzeugung des ersten Stylesheets die Hashtabelle mit den Pfadzuordnungen gefüllt wurde, kann der Generator den Pfad des Blattes als Schlüssel übergeben und bekommt den strukturierten Pfad geliefert, unter dem der Pfad des Blattes stand. In obigem Beispiel trifft er beispielsweise auf das Blatt mit dem Pfad */ISO8583-Msg/MsgType*. Diesen Pfad kennt er vom ersten Durchlauf. Da er dabei in einem Blatt mit dem Pfad */Nachricht/Typisierung/Typ* gestanden hat, liefert diesen Wert auch die Hashtabelle zurück. Also kann folgendes erzeugt werden:

In der Hauptregel:
```
<xsl:apply-templates
select="/Nachricht/Typisierung/Typ/Klasse"/>
```

Als zusätzliche Regel:
```
<xsl:template
match="/Nachricht/Typisierung/Typ/Klasse">
<MsgClass>
       <xsl:apply-templates select="@*"/>
       <xsl:value-of select="."/>
</MsgClass>
</xsl:template>
```

Es ist also durch die Hashtabelle möglich, den Pfad in konstanter Zeit zu ermitteln, im Gegensatz zum linearen Aufwand, der nötig wäre, um ihn durch Suche als Blatt im strukturieren Dokument zu finden. Da diese Suche für jedes Blatt durchgeführt werden muß, hätte der Algorithmus quadratische Laufzeit bezüglich der Anzahl der Blätter des DOM-Baums. Durch die Hashtabelle ergibt sich lediglich linearer Aufwand.

Im Zusammenhang mit der Hashtabelle können zwei Arten von Unstimmigkeiten auftreten. Zum einen kann es passieren, daß für einen flachen Pfad kein Eintrag in der Hashtabelle gefunden wurde. Das bedeutet, daß er im strukturierten Dokument, das die Abbildung definiert, nicht vorkam und somit dort keine Entsprechung besitzt. Das Blatt existiert zwar im flachen Schema, da aber keine Regel erzeugt wird, wird sein Wert weder in das strukturierte übernommen, noch kann es aus dem strukturierten einen Wert erhalten. Damit ist die Abbildungsvorschrift in einem solchen Fall nicht surjektiv. Zum anderen kann es vorkommen, daß nach Beendigung der Tiefensuche noch Einträge in der Hashtabelle vorhanden sind. Es wurden also im strukturierten Schema Abbildungen auf flache Pfade vorgenommen, welche es im flachen Dokument nicht gab. Die Werte der betroffenen Blätter des strukturierten Schemas können also niemals durch eine Transformation ins flache Dokument übernommen werden. Beide Situationen führen zwar nicht zum Abbruch der Generierung, da die erzeugten Regeln ohne Fehler zur Transformation verwendet werden können, aber der Generator produziert entsprechende Meldungen, so daß der Benutzer selbst entscheiden kann, ob die Eingaben für den Generator fehlerhaft oder gewollt waren.

Der Generator hat auf diese Art und Weise zwei XSL-Dokumente erzeugt, die die Transformation zwischen zwei XML-Dokumentklassen beschreiben. Der Generator wurde in JAVA implementiert. Für diese Sprache sind leistungsfähige XML-Parser und XSL-Prozessoren verfügbar (Xerces und Xalan von xml.apache.org).

5 ZUSAMMENFASSUNG UND AUSBLICK

Die Transformation von XML-Dokumenten auf der Basis von XSLT ist für komplexe Dokumente und Abbildungen aufwendig, wodurch die praktische Verwendung in Frage gestellt ist. Wir haben ein Konzept zur Spezifikation von restrukturierenden Transformationen vorgestellt und gezeigt, wie sich XSLT-Dokumente daraus ableiten lassen. Das Verfahren haben wir zur Erzeugung eines Übersetzers

zwischen Austauschformaten unter Bankselbstbedienungssystemen eingesetzt, wodurch die praktische Relevanz des Verfahrens deutlich geworden ist.

In weiteren Arbeiten wollen wir den bisher vorgestellten Ansatz zur Spezifikation von XSL-Transformationen erweitern. Wie gezeigt, lassen sich mit dem vorgestellten Ansatz einfach und wirksam Restrukturierungen von XML-Dokumenten beschreiben. Die von uns vorgeschlagene Spezifikation besteht aus einem XML-Dokument im Zielformat, das als Inhalte Pfadausdrücke von Elementen des Ausgangsdokuments enthält. Ein Nachteil besteht allerdings zum Beispiel darin, daß keine Bedingungen für die Abbildung einzelner Elementen angegeben werden können.

Auf Graphen lassen sich komplexe Transformationen durch die Verwendung von Tripelgraphgrammatiken spezifizieren [8] und werkzeugbasiert durchführen [9]. Tripelgraphgrammatiken erlauben die durch einen dritten *Korrespondenzgraphen* kontrollierte Transformation zweier Graphen ineinander. Spezialisiert auf Bäume könnten Werkzeuge wie PROGRES [9] für die Transformation von XML-Dokumenten genutzt werden.

Für die Spezifikation von XSL-Transformationen kann die Idee der Korrespondenzgraphen zwischen zwei Graphen allerdings auch in adaptierter Form genutzt werden. Zu diesem Zweck wird die Abbildung zweistufig durchgeführt: Das Ausgangsdokument wird in ein Korrespondenzdokument abgebildet, das die Gesamtabbildung steuert. Das Korrespondenzdokument wird in einem zweiten Schritt in das Zieldokument abgebildet.

Die praktische Relevanz von XML und XSL geht bei der Firma s&n AG über den Bereich der Selbstbedienungssysteme hinaus. So werden zum Beispiel Applikationen für das mobile Internet unter der Nutzung von XML (WAP) und XSL bereitgestellt. Mit XSL-Transformationen können Quelldokumentformate in unterschiedliche, vom Ausgabemedium abhängige Zielformate wie zum Beispiel HTML und WAP übersetzt werden. Die mögliche Realisierung dieser Transformationen mit einer Erweiterung des vorgestellten Ansatzes ist vielversprechend.

LITERATUR

[1] R. Tolksdorf : *XML und darauf basierende Standards: Die neuen Auszeichnungssprachen des Web*. In: Informatik Spektrum, Band 22, Springer-Verlag, Berlin, Dez. 1999, S. 407 ff

[2] S. Mohr (ed.): *Professional XML*. Wrox Press Inc., Chicago, 2000

[3] H. E. Rusty: *The XML Bible (Chapter 14: XSL Transformations).* IDG Books, New York, 1999

[4] W3-Konsortium: *XSL Transformations (XSLT) Version 1.0.* W3C Recommendation 16 November 1999, http://www.w3.org/TR/xslt

[5] Normenausschuß Bürowesen (Nbü) im DIN Deutsches Institut für Normung: Europäische Norm EN ISO 8583:1993 *Nachrichten in Verbindung mit Bankkarten, Beschreibung des Nachrichtenaustausches.* Beuth Verlag, Berlin, Oktober 1995

[6] B. Lütkemeier, S. Thöne: *Entwicklung eines Übersetzers von Nachrichtenaustauschformaten für Bankselbstbedienungssysteme in XML-Formate.* Studienarbeit, Universität Paderborn, Mai 2000

[7] S. Krishnamurthi, K. E. Gray, P. T. Graunke: *Transformation-by-Example for XML*. In: E. Pontelli, V. S. Costa (eds.): Proc. PADL 2000, LNCS 1753, Springer-Verlag, Berlin, 2000, S. 249-262

[8] A. Schürr: *Specification of Graph Translators with Triple Graph Grammars.* In Proc. WG 94, Int. Workshop on Graph- Theoretic Concepts in Computer Science, LNCS 903, Springer Verlag, S. 151-163, auch: Technical Report AIB 94-12, RWTH Aachen (1994)

[9] A. Schürr: *Programmed Graph Replacement Systems.* In: G. Rozenberg (ed.): Handbook on Graph Grammars: Foundations, Vol. 1, Singapur: World Scientific (1997), S. 479-546