



Ein anderes Format

Mapping zwischen XML und Datenbank

Helmut Knappe

Unternehmen aller Branchen setzen in allen Bereichen relationale Datenbanken ein, um für ihre Geschäftsprozesse wichtige Informationen zu speichern und zu verwalten. Zugleich ist XML unumstrittener Standard für den Datenaustausch und das Content-Management. Als Konsequenz daraus stehen Tausende Datenbank- und Anwendungsentwickler vor der Herausforderung, XML-Daten in ein relationales Format umzuwandeln und umgekehrt. Der Export von XML aus einer relationalen Datenbank wird zwar von den meisten Datenbankherstellern unterstützt. Deren Importfähigkeiten beschränken sich aber meist auf die eigene Datenbank. Der Artikel beschreibt konzeptionell den Import von XML-Dokumenten in relationale Datenbanken (Deserialisierung bzw. Unmarshalling) anhand einfacher Beispiele, und stellt kurz einen Standardsoftware-Lösungsansatz vor.



RDBMS/XML – Grundlagen

Relationale Datenbanken und XML ergänzen sich

Als XML neu war, dachten viele, XML-Datenbanken würden traditionelle relationale Datenbanksysteme (Relationale Datenbank-Management-Systeme, RDBMS) ersetzen. Inzwischen hat die Praxis jedoch gezeigt: XML und RDBMS sind komplementäre Technologien, die sich vorteilhaft ergänzen.

Relationale Datenbanken stehen für Zuverlässigkeit, Skalierbarkeit, Suchfähigkeit, Abfrage-Performanz. Sie speichern Daten effizient, ohne Redundanz, und besitzen Management- und Sicherheitsmerkmale mit Locking und Caching.

XML ist Text-basiert, von Menschen lesbar, Plattform-unabhängig und ein offener Standard. Dadurch wurde es faktisch die „Lingua franca“ aller IT-Systeme. XML erlaubt, strukturierte Daten ohne zusätzliche Information zu beschreiben, denn die Struktur ist im XML-Dokument enthalten. Dies macht seine Übertragung, Prüfung und Darstellung intuitiver als jeder andere Datenstandard.

Die Information der Welt wird in absehbarer Zukunft weiter in RDBMS gespeichert werden. Daraus ergibt sich die Notwendigkeit, XML auf relationale Datenbanken abzubilden (Mapping).

Austausch relationaler Daten zwischen Anwendungen

XML ist ein natürlicher, persistenter Weg für den Datenaustausch zwischen Anwendungen (A2A), Organisationen (B2B) und heterogenen Datenbanken. Die Zahl der Branchen mit definierten

XML Messaging-Standards ist beeindruckend, siehe http://www.xml.org/xml/industry_industrysectors.jsp. Das selbst-beschreibende XML vereinfacht die B2B-Kommunikation und dient oft als kosteneffektiver Ersatz für EDI (Electronic Data Interchange).

Eine relationale Datenbank als XML-Datenrepository

Übersteigen Größe und/oder Anzahl der XML-Dateien gewisse Grenzen, wandelt man die Information jedoch besser in ein RDBMS-Format um, in dem sie performant abgefragt werden kann. Doch die XML-Schnittstellen der RDBMS-Hersteller sind nicht portabel. Dazu kommt der fundamentale Unterschied zwischen diesen beiden Datenformaten. Deshalb erfordert der Import von XML-Dokumenten in relationale Datenbanken erheblichen Aufwand an Entwicklung und Anpassung. Zugleich wird er, um Abfragen auf Massen-XML-Daten effizient und zentralisiert von vielen Anwendungen zu nutzen, zu einem unternehmenskritischen Prozess.

Relationale Daten mit Web-Services verfügbar machen

Ein Web-Service stellt eine Programmierschnittstelle für entfernte Anwendungen über das Internet zur Verfügung, ähnlich wie ein HTML-Server eine Anwendungsschnittstelle für Browser-Clients über das Internet bietet. Der Zugriff auf Web-Services erfolgt über das XML-basierte Protokoll SOAP. Die Daten werden als XML zurückgegeben. Der Web-Service muss sie zur Speicherung in einer relationalen Datenbank transformieren.

Relationale und hierarchische Repräsentation der Daten

Bei der Abbildung von XML auf RDBMS betrachten wir zwei unterschiedliche Arten von Datenstrukturen: die relationale Datenbankstruktur und die XML-Datenstruktur.

Eine relationale Datenbank besteht aus Tabellen, verbunden durch Relationen. Jede Tabelle besteht aus einer festen Sammlung von Spalten (auch Felder genannt), die Attributen des Datenmodells entsprechen. Jede Tabelle enthält eine unbestimmte Anzahl Zeilen (oder Datensätze). Im Idealfall hat jede Tabelle einen eindeutigen Primärschlüssel, auf den sich andere Tabellen beziehen können, indem sie einen Fremdschlüssel verwenden, der denselben Wert enthält.

Im Gegensatz dazu werden XML-Daten am besten als Baumstruktur dargestellt, die Relationen durch Einschließung (Containment) ausdrückt. Jede Verzweigung ist ein XML-Element, welches ein oder mehrere Attribute und andere Elemente enthält. Die Stärke von XML ist zugleich seine Schwäche: Es dient als vielseitiges Format mit loser Struktur und Elementen, die fast überall vorkommen können. Es kann aber auch eine feste hierarchische Struktur haben.

Mit XML fällt es schwer, Relationen zwischen Tabellen zu beschreiben. Die ausführliche Beschreibung und lose Formatierung von XML steht den Performanzstrategien der RDBMS entgegen. Deshalb ist die Abbildung eines XML-Dokuments auf eine relationale Datenbank ein komplexer Vorgang. Sie erfordert Fachkenntnisse sowohl bzgl. des Designs von RDBMS als auch XML und eine klare Methode, wie jedes Element im XML-Schema eindeutig auf das Datenbankschema abzubilden ist.

Unmarshalling an Beispielen

Der folgende Abschnitt erläutert die Prinzipien und wichtigsten Bedingungen, die beim Import von XML-Daten in relationale Datenbanken zu beachten sind.

Tabellen-basiertes Mapping

Wir beginnen mit einem kleinen Beispiel: Ein einfaches XML-Dokument listet einige Angestellte in einer Organisation auf. Es geht darum, eine Klasse von Dokumenten (mit demselben XML-Schema) in einer relationalen Datenbank abzulegen. Damit werden alle XML-Dokumente mit diesem XML-Schema im RDBMS abgelegt. Ein passendes Schema für diese Klasse von XML-Dokumenten ist.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name="MA">
    <xsd:complexType>
      <xsd:element name="MANR" type="integer" />
      <xsd:element name="MANAME" type="string" />
      <xsd:element name="FUNKTION"
        type="string" />
      <xsd:attribute name="EINTRDAT"
        type="date"
        minOccurs="0"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Die Daten können wir leicht in die relationale Tabelle **MA** mit den Feldern

```
MANR MANAME FUNKTION EINTRDAT
```

einfügen. **MANR** ist der Primärschlüssel für diese Tabelle. Finden wir keine Eindeutigkeit für eine Spalte oder Spaltenkombination, legen wir eine zusätzliche Spalte für den Primärschlüssel an. Für XML-Attribute gilt die gleiche Abbildungsvorschrift wie für XML-Elemente.

Containment abbilden

Reale XML-Dokumente haben meist eine komplexere Struktur mit mehreren Ebenen. Enthält das XML-Dokument die Adresse eines jeden Mitarbeiters, so erweitert sich das Schema um das Element **ADRESSE** innerhalb der Tabelle **MA**:

```
<xsd:element name="ADRESSE">
  <xsd:complexType>
    <xsd:attribute name="STRASSE" type="string" />
    <xsd:attribute name="STADT" type="string" />
    <xsd:attribute name="PLZ" type="string" />
  </xsd:complexType>
</xsd:element>
```

Nach der W3C-Spezifikation hat ein komplexer Typ ein oder mehrere Elemente bzw. Attribute oder gemischten Inhalt, während ein einfacher Elementtyp einen einzelnen Datenwert enthält. Wenn

sicher ist, dass für jeden Mitarbeiter nur eine Adresse gespeichert wird, kommen die Adressdaten einfach mit in die Tabelle **MA**. Aber im allgemeinen Fall ist für jedes neue komplexe Element eine neue Tabelle zu erzeugen, in diesem Fall die Adresse:

```
ADRESSNR STRASSE STADT PLZ
```

Für die Praxis bedeutet das: Ein komplexer Typ bildet sich auf eine Tabelle mit Primärschlüssel ab, ein einfacher Typ auf eine einzelne Spalte.

Wenn ein spezifischer Wert für ein XML-Element öfter vorkommen kann, erzeugt man einen Primärschlüssel. Das Unmarshalling komplexer XML-Dokumente führt deshalb zu einer Vielzahl von Schlüssel- und Tabellen-Joins.

Reihenfolge

Die Reihenfolge der Elemente bzw. Attribute in einem XML-Schema steht unter **<xsd:Reihenfolge>**. Entspricht sie dem Primärschlüssel der Tabelle, erhalten wir dort dieselbe Reihenfolge. Sonst fügen wir eine zusätzliche Spalte mit einem Integer-Wert hinzu, der der gewünschten Reihenfolge entspricht, und sortieren danach.

Optionales XML-Element

Wir drücken ein optionales XML-Element aus, indem wir für die Spalte NULL-Werte zulassen. In unserem XML-Schema ist **EINTRDAT** ein optionales Attribut (**minOccurs = 0**). Die entsprechende Spalte muss NULLs zulassen.

Auswahl

In unserem Beispiel kann das XML-Element **FUNKTION** nur folgende Werte enthalten: **CEO**, **CTO**, **CFO**, **DIREKTOR**, **MANAGER**, **INGENIEUR**, **VERTRIEBSMA**, **BERATER** und **ASSISTENT**. Der folgende Schema-Abschnitt drückt diese Auswahl aus und nutzt dafür das Schlüsselwort **Choice**:

```
<xsd:element name="FUNKTION">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="CEO" type="xsd:string" />
      ...
      <xsd:element name="ASSISTENT"
        type="xsd:string" />
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

In der relationalen Datenbank erzeugen wir die zusätzliche Tabelle **FUNKTION** mit den Feldern

```
FUNKTIONNR FUNKTIONTITEL
```

FUNKTION stellt alle diese Werte über eine Primär-/Fremdschlüssel-Relation zur Verfügung.

Wiederholte XML-Elemente (Kardinalität)

Ein erweitertes XML-Dokument informiert über den Computer, den jeder Mitarbeiter einsetzt. Die meisten Angestellten haben nur einen Computer; doch Johann, der System-Administrator hat drei. Wenn wir die Anzahl der Computer auf drei begrenzen (**maxOccurs = 3**), genügen drei Spalten in der **MA**-Tabelle für das Computermodell eines jeden Mitarbeiters: **COMPNR1**, **COMPNR2** und **COMPNR3**. Ist dagegen die maximale Anzahl der Computer für einen Angestellten unbegrenzt (**maxOccurs = unbounded**), müssen wir die Struktur der Datenbank erneut erweitern. In der Tabelle **COMP** werden die Computer über den Fremdschlüssel **MANR** den Mitarbeitern zugeordnet:

```
COMPNR MODELL RAM OS MANR
```

Damit ist die Anzahl der Elemente unbegrenzt. Dieses Vorgehen passt in den meisten Fällen bei wiederholten Elementen.

Aber was tun, wenn das wiederholte Element verschiedene übergeordnete Elemente hat? Beispielsweise wenn Angestellte auch unternehmensfremde Computer einsetzen, Berater sich Computer leihen oder im Testlabor Computer von verschiedenen Angestellten genutzt werden? Dann definiert man eine Auswahltable, wie **MACOMP** mit den Feldern **MANR** und **COMPNR**, die jeden Computer mit seinem Anwender verbindet.

Zeiger ID/IDREF

Falls ein Unternehmen bestimmte Computer als Standard festlegt, kann das XML-Dokument die Zeiger **ID** und **IDREF** verwenden, um die Wiederholung der XML-Elemente zu vermeiden. Da Schlüssel bei relationalen Datenbanken sehr gebräuchlich sind, kann man XML-ID-Elemente auf Primärschlüssel abbilden, wenn ihre Eindeutigkeit sicher ist.

Gemischter Inhalt

Gemischter Inhalt kommt vor allem beim Content-Management häufig vor, besonders in der Medienbranche. HTML besteht fast immer aus gemischtem Inhalt, mit spezifischen Tags für bestimmte Inhalte.

Der einfachste Weg für die Umwandlung einer XML-Datei mit gemischtem Inhalt ist, die Reihenfolge der unterschiedlichen Tags nach der Reihenfolge in der neuen Tabelle auszudrücken. Dazu benötigt man eine zusätzliche Spalte, um den Typ des benutzten HTML-Tags darzustellen. Eine Look-up-Tabelle für jeden verwendeten HTML-Tag vermeidet redundante Wiederholungen.

Zyklische Verbindung

Ein XML-Schema kann eine rekursive Verbindung für dasselbe Element enthal-



ten. Wir können beispielsweise das Element `<VORGESETZTER>` zu `<MA>` hinzufügen, und dieses Element kann erneut `<MA>` enthalten. Zum Beispiel Jack, der CEO, ist Alberts Vorgesetzter, und Lionel berichtet an Albert.

Auch in diesem Fall verbindet man die unterschiedlichen Elemente über eine Look-up-Tabelle. Enthält ein XML-Schema eine zyklische Relation, so bedeutet das theoretisch eine unendliche Schachtelungstiefe. Die Praxis erlaubt aber meist die Beschränkung der möglichen Anzahl verschachtelter Elemente.

Unmarshalling praktisch umsetzen

Für das Unmarshalling eines XML-Dokuments in eine relationale Datenbank müssen wir alle Strukturen identifizieren, die wir zuvor beschrieben haben: einfache und komplexe Typen, Reihenfolge, Auswahl, Containment, Kardinalität, Zeiger, gemischter Inhalt, zyklische Relationen und sie auf ein sinnvolles relationales Datenbankschema abbilden. Dabei hilft ein Diagramm des Datenbankschemas, das alle Relationen zwischen den Tabellen zeigt.

Alle XML-Elemente und Attribute werden auf individuelle Spalten abgebildet. Einige Feldelemente stammen aus String- oder numerischen Operationen auf einem oder mehreren XML-Elementwerten. Bei der Durchführung der Abbildung sind folgende Regeln wichtig:

- ▼ Achten Sie auf die Datenintegrität in der relationalen Datenbank.
- ▼ Erzeugen Sie nur sinnvolle Tabellen und minimieren die Joins aus Performancegründen.
- ▼ Stellen Sie sicher, dass Sie alle XML-Elemente abbilden.

Entwicklungsstandards

RDBMS wie Oracle, SQL Server und DB2 bieten proprietären XML-Support. Oracle baut dazu mit Java ein objekt-relacionales Datenmodell auf. IBM DB2 Extender nutzt dazu ein *Data Access Definition File* in XML und Microsoft SQL Server erweitert SQL um die Rowset-Funktion OPENXML. Für Entwickler ist es aber wichtig, mit einer universellen Schnittstelle zu arbeiten, besonders, wenn mehrere unterschiedliche Datenbanksysteme vorliegen. Zum Beispiel mit JDBC für die relationalen Daten sowie die Standard-APIs und DOM und SAX für XML-Daten.

Unmarshalling selbst programmieren

Einfache Unmarshalling-Aufgaben können Softwareentwickler durchaus selbst

DOM, SAX, XSLT, JAXP und JAXB

DOM bietet ein Objektmodell, das jedes XML-Dokument beliebiger Struktur nachbilden kann – und Zugriff auf seinen Inhalt verleiht. Es erzeugt einen Baum im Speicher, der das Dokument als Objekte darstellt, auf dessen „Nodes“ man sich bezieht.

SAX wurde dagegen von Mitgliedern einer Mailing-Liste namens XML-Dev entwickelt. SAX ist eine Alternative oder Ergänzung zu DOM als Methode für den Zugriff auf XML-Dokumente. Im Gegensatz zu DOM ist SAX eine ereignisgesteuerte Schnittstelle, die den Aufruf bestimmter Methoden als XML-Elemente erfordert und Attribute parsed. Wenn ein DOM-basierter Parser ein XML-Dokument verarbeitet, hält er eine Baum-Darstellung des Dokuments im Speicher. SAX dagegen arbeitet das Dokument von Anfang bis Ende durch und erfasst die unterschiedlichen Elemente, die es antrifft. Für jedes Strukturelement im Dokument ruft SAX eine Methode auf, die Sie zur Verfügung stellen. Das ist ein sehr praktischer Ansatz, um große XML-Dokumente zu verarbeiten ohne den Aufwand, einen Baum im Speicher aufzubauen.

XSLT ist eine Hochsprache, um ein XML-Dokument in ein anderes XML-Dokument oder in HTML, PDF oder ein anderes Format umzuwandeln. Es wird häufig eingesetzt, um XML für die Ausgabe auf dem Bildschirm in HTML umzuwandeln.

JAXP bietet eine Spezifikation für einen Satz reiner Java APIs, die XML-Daten lesen, bearbeiten und XML-Daten erzeugen können. JAXP verwendet eigene Methoden aus DOM, SAX und XSLT, um XML-Dokumente unabhängig vom Parser konsistent zu machen.

JAXB ist die Java-Architektur für XML-Binding und wurde im Rahmen des *Java Community Process*-Programms unter JSR-31 entwickelt. Es automatisiert das Mapping zwischen XML-Dokumenten und Java-Objekten, indem es das XML-Schema in eine oder mehrere Klassen kompiliert. Als Konsequenz erzwingt es die Einhaltung der im Schema ausgedrückten Constraints und ist viel effizienter und intuitiver als SAX und DOM.

lösen. Java-Beispielcode für die wichtigsten Abschnitte eines einfachen Unmarshalling-Programms ist unter <http://www.sigs-dac.com.de/sd/publications/js/2006/03/index.html> hinterlegt. Der Code-Abschnitt liest ein XML-Dokument und füllt eine *MS Access*-Tabelle mit den Informationen aus diesem XML-Dokument. Es verwendet folgende Standard-Pakete von JDK 1.3: IO, SQL und die Standard-XML-APIs DOM, SAX und JAXP. JAXP (jaxp.jar) und der XSLT-Prozessor Xalan (xalan.jar) müssen im Classpath vorhanden sein.

Der Code öffnet das XML-Dokument und parst es mit SAX. Die Datenbankverbindung erfolgt über JDBC-Treiber. Datenbank, UserID und Passwort sind anzugeben. Dann baut das Unmarshalling-Programm aus den XML-Elementen den Datensatz in einem Puffer auf und fügt ihn in die Datenbank ein. Eine einfache Fehlerbehandlung komplettiert das Beispiel.

Für den Compiler-Lauf fügen Sie jaxp.jar und xalan.jar in Ihren Classpath ein:

```
javac -classpath C:\Java\XML\jaxp.jar;C:\Java\XML\xalan.jar Unmarshal.java
```

Beim Starten geben Sie die XML-Eingabedatei an:

```
java -classpath .; C:\Java\XML\jaxp.jar; C:\Java\XML\xalan.jar Unmarshal "C:\java\sample\Unmarshal\emp.xml"
```

Einschränkungen des Unmarshalling-Programms

- ▼ Dieses Programm zeigt eine einfache Umsetzung grundlegender Mapping-Regeln. Es ist nicht allgemein einsetzbar, sondern erfordert Änderungen für jede andere XML-Dokumentstruktur. Das Beispiel zeigt, was man tun muss, um ein spezifisches DTD oder XML-Schema zu unterstützen. Der Einsatz mehrerer XML-Schemata erfordert die Entwicklung eines allgemeinen Programms oder den Einsatz einer fertigen Lösung, die die Kosten begrenzt und dazu bessere Wartbarkeit ermöglicht.
- ▼ Die Fehlerbehandlung ist im Programmbeispiel nur rudimentär ausgeführt und verbesserungsfähig.
- ▼ Das XML-Dokument wurde zu Lernzwecken stark vereinfacht – fast jedes reale Dokument ist viel komplizierter und enthält komplexere XML-Strukturen, als bei unseren Beispielen beschrieben.
- ▼ Für die Abbildung von XML-Datentypen, die Erzeugung der Tabellen und die Bewahrung der Datenintegrität beim Einfügen neuer Sätze in die Datenbank ist zusätzlicher Code erforderlich.
- ▼ Komplexere Themen, wie *Connection Pooling*, *Data Binding*, um Java-Objekte direkt aus den XML-Dokumenten zu erzeugen, und Performance-Optimierung übersteigen den Umfang dieses Artikels.

Einsatz eines Standard-Mappers

Wie man an dem Beispiel und an den obigen Mapping-Regeln sieht, ist die Aufgabe, XML-Dokumente auf relationale Datenbanken abzubilden, keineswegs trivial. Das Unmarshalling komplexer XML-Dokumente führt zu einer komplexen Datenbankstruktur mit vielen Schlüsseln und Tabellen-Joins. Ein allgemeiner Lösungsansatz, der referentielle Integrität berücksichtigt und auf optimale Performance achtet, erfordert die Durchführung eines eigenen, umfangreichen Projekts und lässt sich keinesfalls als Hilfsaufgabe einer Anwendung erledigen. In der Praxis entsteht deshalb häufig eine sehr spezifische Lösung, die bei der nächsten Änderung/Erweiterung der Strukturen konsequenterweise erheblichen Mehraufwand verursacht.

Einen Ansatz, der die Implementierung einer XML-RDBMS-Integration in Entwicklungsprojekten sehr erleichtert, bietet beispielsweise Allora von HiT Software. Die Software besteht aus einem grafischen Mapper (s. Abb. 1), der interaktiv die Definition von XML-RDB-Mappings erlaubt, und einer Transformations-Engine. Das API unterstützt die Entwicklung der Programmteile, die Mapping-Dateien verarbeiten sowie XML-Strukturen und Datenbanken verwalten. Unterstützung für gängige Entwicklungsumgebungen (Eclipse, Websphere Studio, Oracle JDeveloper, Sun Java Studio, Borland JBuilder etc.) durch Assistenten und Designer beschleunigt die Entwicklung.

Die Praxis zeigt, dass der Einsatz einer solchen Mapping-Lösung (s. Abb. 2) besonders in folgenden Situationen erheblichen Entwicklungsaufwand spart:

- ▼ XML-Schemata und Datenbankstrukturen sind komplex und umfangreich. Paradebeispiele sind die vielen Standard-Schemata, die heute bereits in vielen Branchen existieren. Solche umfangreichen Schemata steigern den Aufwand für die Entwicklung einer performanten Lösung

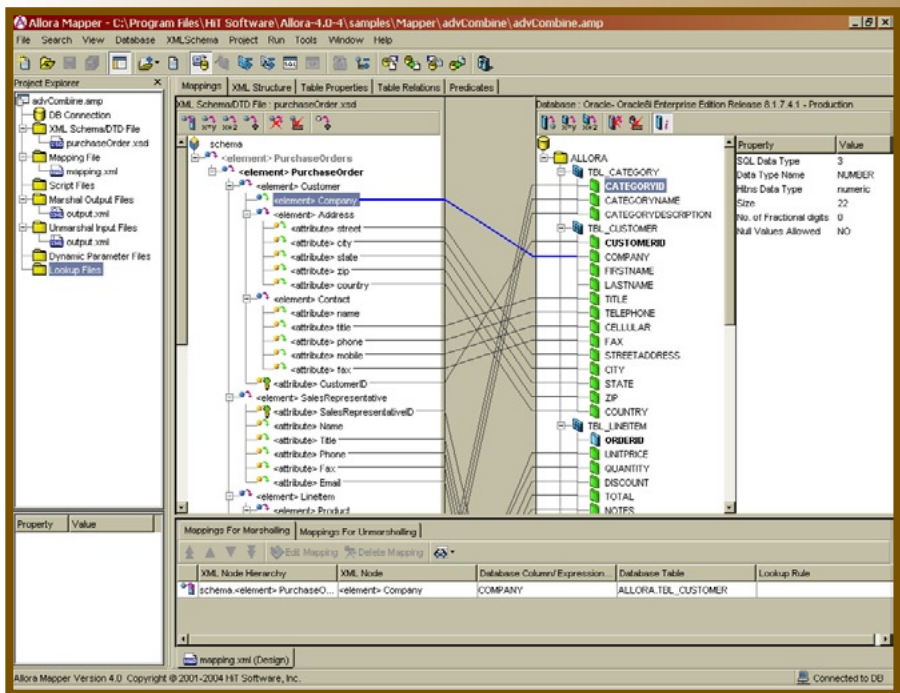


Abb. 1: Mapping zwischen XML und Datenbank – übersichtlich dargestellt in Allora

gewaltig. Allora unterstützt bereits über 20 solche Schemata und macht dadurch manche Projekte erst durchführbar.

- ▼ Mehr als ein XML-Schema ist involviert bzw. es sollen unterschiedliche XML-Dokumente importiert werden.
- ▼ Mehr als eine Datenbank-Plattform ist beteiligt – eine Hersteller-unabhängige Lösung ist nötig.
- ▼ XML-RDB-Transformationen müssen performant ablaufen – erfordert zusätzlichen Entwicklungsaufwand und viel Erfahrung.
- ▼ Änderungen der beteiligten Strukturen im Lauf der Zeit sind zu erwarten. Jede Änderung erfordert bei einer selbst programmierten Lösung hohen Pflegeaufwand, der beim Einsatz von Allora vermieden wird.

Fazit

Der Artikel führt in die Grundlagen des Imports von XML-Daten in relationale Datenbanken ein, erläutert die Herausforderungen beim Übergang vom XML-Schema zum Datenbank-schema an einfachen Beispielen und verdeutlicht so die Komplexität des Unmarshalling. Standard-Lösungsansätze wie HiT Software Allora bieten einen allgemeinen Ansatz für das Mapping unterschiedlicher XML-Dokumente auf relationale Datenbanken. Je anspruchsvoller die Aufgabenstellung ist, umso größer wird der Aufwand an Entwickler-Ressourcen, Zeit und Kosten, den der Einsatz von Allora einsparen kann.

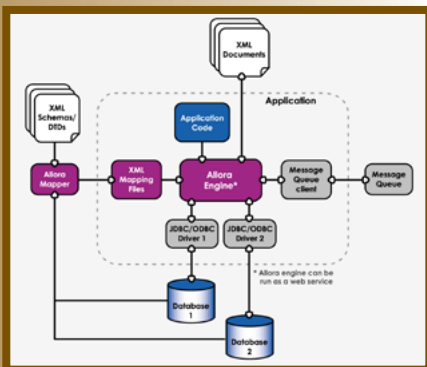


Abb. 2: Funktionsschema: XML-RDB-Transformation mit Allora



Dr.-Ing. Helmut G. Knappe ist seit mehr als 20 Jahren in der IT-Entwicklung und Projektleitung bei Datenbankanwendungen tätig. Zu seinen Schwerpunkten zählt die Integration heterogener Datenbanken. Seit 2002 ist er als Country Manager Deutschland und Österreich für den amerikanischen Softwarehersteller HiT Software Inc. tätig. Er hält zahlreiche Vorträge sowie Workshops und schreibt Artikel für Fachzeitschriften. E-Mail: helmut.knappe@hitsw.com.

Weitere Informationsquellen

<http://www.sigs-datacom.de/sd/publications/js/2006/03/index.htm>