Kooperation in Babylon

Semantische Interoperabilität von XML Schemas

Erik Wilde

XML bietet zwar durchaus das allgemein akzeptierte Verfahren zum Austausch strukturierter Daten, das in vielen Anwendungen benötigt wird, ist aber dennoch nicht ausreichend, Interoperabilität zwischen Anwendungen sicherzustellen. Probleme können auf vielen verschiedenen Ebenen entstehen, beginnend bei so grundlegenden Dingen wie Zeichencodierungen, bis hin zu Problemen des inhaltlichen Verständnisses von XML Dokumenten. Im vorliegenden Artikel soll auf den letzteren Aspekt näher eingegangen werden, also die Frage, was notwendig ist, damit der Austausch von XML nicht nur syntaktisch funktioniert, sondern auch auf einem gemeinsamen Verständnis beider Seiten basiert.

In den vorangegangenen Artikeln dieser Serie wurde XML hauptsächlich unter dem Aspekt betrachtet, wie man mit XML Schema modellieren kann ("Verwirrende Vielfalt — Modellierungsvarianten mit XML Schema", XML & Web Services Magazin 6.2003 und "Lose gekoppelt hält länger — Entwurf erweiterbarer XML Schemas", XML & Web Services Magazin 3.2004) und wie die Versionierung von XML Schemas in Verbindung mit XML Namespaces angegangen werden sollte ("Drum prüfe, wer sich ewig bindet... — Namespaces und Versionierung von XML Schemas", XML & Web Services Magazin 4.2004). Im abschliessenden vierten Teil der Artikelserie wird auf die Frage eingegangen, inwieweit bei der XML-orientierten Kommunikation von einem gemeinsamen Verständnis der Daten ausgegangen werden kann, bzw. wie sich ein solches gemeinsames Verständnis erreichen und implementieren lässt.

Versionierung als Spezialfall von B2B

Obwohl in den erwähnten vorangegangenen Artikeln viel von XML Schema und der Versionierung von XML Schemas die Rede war, soll der Blick auf XML-orientierte Applikationen diesmal etwas allgemeiner sein. Dies ist ohne weiteres möglich, da sich die Versionierung von XML Schemas einfach als einen Spezialfall von B2B Kommunikation betrachten lässt, bei dem eine ältere Applikation mit einer neueren kommuniziert und beide Systeme interoperabel sein sollten. Generell ist das Ziel von Interoperabilität zwischen XML-orientierten Applikationen, dass nicht nur XML ausgetauscht wird, das von beiden Applikationen akzeptiert wird (auf syntaktischer Ebene), sondern dass das XML von beiden Applikationen gleich verstanden wird (dies betrifft die semantische Ebene). Das klassische Beispiel in diesem Bereich sind Längenangaben, die ohne Einheit verwendet werden. Ob man diese Längenangaben dann metrisch oder englisch interpretiert, ist essentiell, und der Mars Climate Orbiter der NASA hat sich genau aufgrund eines solchen Problems mehr oder minder ungebremst in die Marsoberfläche gebohrt: Der Orbiter hatte seine Daten metrisch gesendet und interpretiert, während die NASA Steuerzentrale die Daten als englisch interpretierte und sendete.

Nehmen wir an, dass der Mars Climate Orbiter und die NASA über XML kommunizierten (was wohl eher nicht der Fall gewesen sein wird bei der zur Verfügung stehenden Bandbreite), dann war das grundlegende Problem, dass die Daten zwar oberflächlich gesehen ausgetauscht werden konnten (eine Länge mit einem numerischen Wert wurde von beiden Seiten gesendet und akzeptiert), dass diese oberflächliche, rein syntaktische Interoperabilität aber genauer betrachtet durch eine unterschiedliche Interpretation der numerischen Grösse wertlos und sogar schädlich war.

Abbildung 1 zeigt, in welcher Art die Verarbeitung von XML in einem B2B Szenario typischerweise betrachtet werden kann. Auf beiden Seiten (Applikationen A und B) muss das XML an die Applikation gebunden werden, so dass die Applikation in geeigneter Weise mit den Daten umgehen kann. Zu diesem "XML Data Binding"

Schritt gibt es verschiedenste Technologien, die natürlich auch von der jeweiligen Programmiersprache und Ausführungsumgebung abhängen, aber dieser Aspekt soll im Rahmen des vorliegenden Artikels nicht weiter behandelt werden. Interessant ist dagegen der dazwischenliegende Schritt, das "XML Mapping", bei dem das aus Applikation A stammende XML durch Abbildungsvorschriften auf das Schema der Applikation B abgebildet wird.

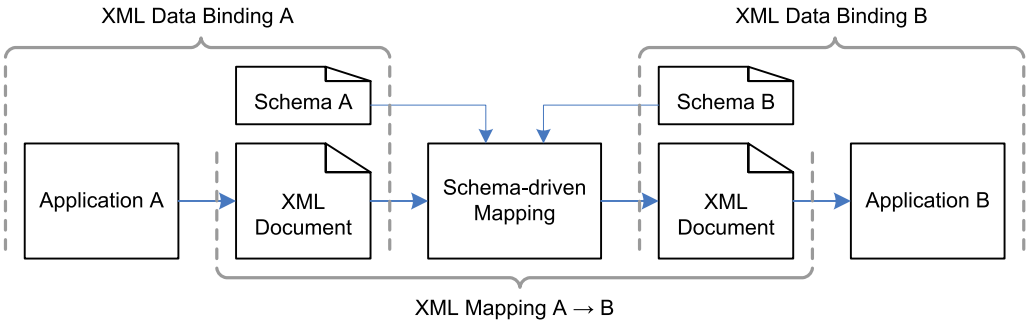

Abbildung 1: XML Data Binding und XML Mapping

Semantische Interoperabilität

Abbildung 1 stellt die Situation so dar, als sei die Abbildung zwischen beiden Instanzen ohne Probleme möglich. In einfachen Fällen kann das durchaus der Fall sein, so kann die Abbildung wie im Fall der NASA aus so einfachen Aspekten bestehen wie einer Umrechnung in andere Masseinheiten. Grundsätzlicher stellt sich aber die Frage, ob eine solche Abbildung überhaupt verlustfrei (und oft auch noch in beiden Richtungen) möglich ist, und wenn dies nicht der Fall ist, wie mit eventuell auftretenden Verlusten umzugehen ist.

Der erste Schritt in einem B2B Szenario besteht also darin, anhand der beiden Schemas (und vor allem deren Semantik, also der Bedeutung der Daten auf der Anwendungsebene) zu untersuchen, ob Instanzen der beiden Schemas verlustfrei aufeinander abgebildet werden können. Im Fall von Verlusten ist zudem zu definieren, wie damit umgegangen werden soll, also ob diese Verluste einfach als Resultat der Abbildung akzeptiert werden, oder ob der Verlust eine Meldung auslösen soll, auf die reagiert werden muss. Ein typischer Fall von akzeptablen Verlusten ist die Datenaggregation, wenn also das Ausgangsschema detailliertere Angaben enthält als das Zielschema, dieser Datenverlust aber akzeptabel oder sogar gewünscht ist. Klar ist in diesem Fall allerdings auch, dass eine verlustfreie Rückkonvertierung nicht möglich ist.

Diese semantische Abbildung von Daten aufeinander ist der erste und wichtigste Schritt zur Sicherstellung der semantischen Interoperabilität. Wichtiger als XML-Kenntnisse sind in diesem Fall genaue Kenntnisse der Anwendungen und der Bedeutung der ausgetauschten Daten. Um diesen wichtigen Schritt angemessen zu dokumentieren, sollte die Abbildung der Daten aufeinander in einem separaten Dokument festgehalten werden, idealerweise auf eine Art, die, falls gefordert, als Ausgangspunkt für Abbildungen in beiden Richtungen dienen kann. In der Praxis zeigt sich, dass viele Abbildungen symmetrisch sind, also nicht separat für beide Richtungen aufgeführt werden müssen. Für nicht-symmetrische Abbildungen sollten die Gemeinsamkeiten nach wie vor festgehalten werden, aber die Unterschiede getrennt angegeben werden.

Ein Beispiel für eine nur teilweise symmetrische Abbildung sind zwei Schemas, bei denen das eine getrennte Elemente für Tag, Monat und Jahr aufweist, währenddessen das andere ein Datumsfeld hat, in dem z.B. ein `xs:date` oder eine `xs:gYearMonth` oder ein `xs:gYear` stehen darf. Während die grundlegende Abbildung symmetrisch ist (Tag/Monat/Jahr auf der einen Seite und Datum auf der anderen Seite), ist die genaue Abbildung abhängig von der Abbildungsrichtung. Im Fall von Tag/Monat/Jahr muss geprüft werden, welches der Elemente vorhanden ist, und dann muss ein Wert des entsprechenden Typen in das Datum eingetragen

werden. In der anderen Richtung muss der Wert des Datum Feldes auf seinen Typ geprüft werden (z.B. mit der neuen XPath 2.0 `castable as` Anweisung), und ausgehend von diesem Test werden dann die notwendigen Tag/Monat/Jahr Felder mit den jeweiligen Komponenten des Datum Feldes gefüllt (z.B. mit den neuen XPath 2.0 `year-from-date/month-from-date/day-from-date` Funktionen).

Wichtig ist jedoch, dass die Beschreibung der Abbildungen nicht unbedingt von einem XSLT Spezialisten gemacht werden muss, sicher jedoch von einem Spezialisten der Anwendungen. Es empfiehlt sich daher sehr, die Beschreibung der Abbildungen nicht von vornherein mit deren Implementierung zu verknüpfen, sondern diese beiden Dinge getrennt zu sehen und zu behandeln (die oben vorgeschlagenen XPath 2.0 Funktionen sind daher mehr als Veranschaulichung gedacht und nicht als notwendiger Detaillierungsgrad auf der Stufe der Abbildungsvorschriften).

Sind die Abbildungsvorschriften festgelegt, so steht als zweite Aufgabe ihre Implementierung an. Dazu sollte das Dokument mit den Abbildungsvorschriften als Anleitung ausreichen, jedoch sind in diesem zweiten Schritt die Fähigkeiten eines Programmierers gefordert, der die Abbildungsvorschriften so umsetzt, dass sie auch effizient ausgeführt werden. Die heikle Frage an diesem Punkt ist: Ändern sich die Abbildungsvorschriften, wird das in jedem Fall eine Änderung der Implementierung erfordern, oder ist die Implementierung so flexibel und offen, dass an einer rein deklarativen Tabelle eine solche Änderung vorgenommen werden kann, und die neue Abbildungsvorschrift ohne jeglichen neuen Code ausgeführt wird. Natürlich hängt es von der Menge der möglichen Änderungen ab, aber tendenziell ändern sich die Abbildungsvorschriften öfter, als man es sich am Anfang denkt, und eine gegen kleine Änderungen robuste Implementierung hat viele Vorteile.

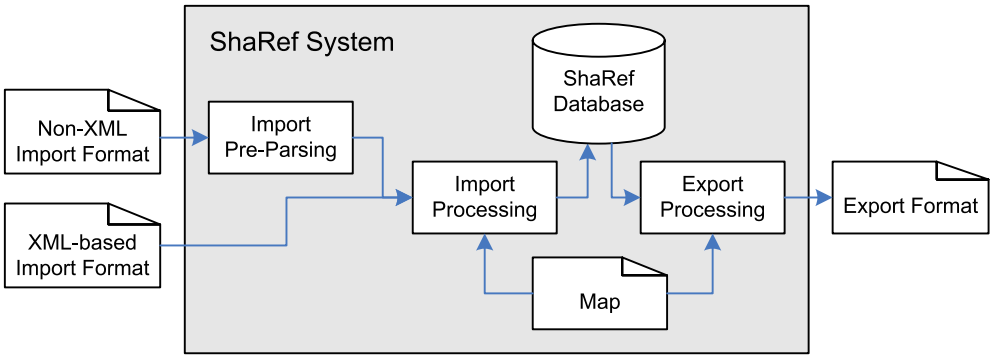

Abbildung 2: Import/Export in ShaRef/bibconvert

Mapping Tables

In einem XML-zentrierten Projekt (ShaRef, http://dret.net/projects/sharef/) wenden wir eine solche robuste Implementierung an, die auf dem Web mit Abbildungen zwischen verschiedenen Formaten für Literaturreferenzen (z.B. EndNote und BibTeX) unter http://dret.net/bibconvert/ ausprobiert werden kann. Der Anwendungsfall ist hier nicht besondern interessant, aber interessant ist das Design des Import/Export Prozesses, der wie in Abbildung 2 gezeigt aus einfachen Import und Export Modulen besteht (geschrieben in XSLT 2.0), die die Abbildungsinformation jeweils aus einem Map Dokument beziehen, das die Abbildungsinformation in deklarativer Weise als ein XML Dokument definiert.

Unter http://dret.net/bibconvert/map lässt sich die Map anschauen, und diese Seite ist auch deswegen etwas langsamer im Aufbau, weil die XML Map per XSLT in eine HTML-Seite übersetzt wird. Wird ein Fehler in den Abbildungsvorschriften entdeckt, so wird er in der Map korrigiert und ist sofort wirksam, da die Map bei jedem Import/Export benutzt wird. Zudem muss bei einem Fehler der XSLT Code nicht unbedingt geändert werden,

jedoch trifft dies nur zu, wenn der Fehler ein reines Mapping und keine Verarbeitung betrifft. Dies spiegelt die oben angesprochene Unterscheidung wider aus rein deklarativer Map, und zusätzlichem Code, der diejenigen Abbildungsvorschriften implementiert, die nicht reine Mappings sind. Als Beispiel sollen die folgenden Mappings dienen, die ein kleiner Ausschnitt aus der gesamten Map sind (wer Interesse an der gesamten Map hat, kann sie unter http://dret.net/bibconvert/xslt/bibmap.xml herunterladen):

```
<mapping importComplete="yes" exportComplete="yes">
        <external format="endnote7" field="TERTIARY_TITLE" refType="3"/>
        <sharef field="title" type="sharef:seriesTitle"/>
</mapping>
<mapping importComplete="no" exportComplete="no">
        <external format="bibtex" field="day"/>
        <external format="bibtex" field="month"/>
        <external format="bibtex" field="year"/>
        <sharef field="date"/>
</mapping>
```

Dieses Fragment aus der Map (die einem sichtbar einfachen Schema folgt) definiert im ersten Element ein Mapping, das definiert, dass ein EndNote 7 Element `TERTIARY_TITLE` abgebildet wird auf ein ShaRef Element `title` mit dem Attribut `type="sharef:seriesTitle"`, dass diese Abbildungsvorschrift komplett ist und von daher rein aus der Map gesteuert wird. Der XSLT Code macht in diesem Fall nicht mehr, als bei der Abbildung von EndNote 7 auf ShaRef (im Import oder Export) die definierte Abbildung vorzunehmen. Stellt sich die Abbildung als falsch heraus, muss nur die Map geändert werden, und die Abbildung wird entsprechend anders durchgeführt.

Der zweite Teil ist ein wenig komplizierter. Es wird definiert, dass die BibTeX Elemente `day`, `month` und `year` auf das ShaRef Element `date` abgebildet werden (wie in Abbildung 1 gezeigt werden nicht-XML Formate wie BibTeX zunächst in eine XML-Repräsentation gebracht, so dass sich aus Sicht der Map auch BibTeX als XML behandeln lässt). Da diese Abbildung aber nicht komplett ist, sondern zusätzlicher Code implementiert werden muss, der die Abbildung in der oben beschriebenen Weise vornimmt, ist die Abbildung als nicht komplett gekennzeichnet. Der XSLT Code muss diese Abbildung also speziell behandeln, ist eine solche Behandlung nicht implementiert, kann anhand des `importComplete/exportComplete="no"` im XSLT festgestellt werden, dass aus Sicht der Map die Abbildung nicht komplett ist, und es kann eine Warnung erzeugt werden.

Das hier beschriebene Vorgehen ist zwar sehr flexibel, weil wirklich ausschliesslich die Map geändert werden muss, wenn sich etwas einfaches an einer Abbildungsvorschrift ändert, aber es ist auch nicht sehr effizient, weil der XSLT Code bei jeder Abbildung die entsprechende Regel in der Map suchen muss. Durch die Verwendung von XSLT `key`s lässt sich die Performance zwar erheblich verbessern, aber nach wie vor ist der Code ein wenig umständlich, weil er sehr allgemein gehalten ist. Eine Variante, die ebenfalls möglich wäre, wäre das Generieren von XSLT Code aus der Map, so dass bei jeder Änderung der Map neuer Code generiert werden muss. Dies ist der Ansatz, den kommerzielle Mapping Tools wählen.

Mapping Tools

Dass XML viel zum Datenaustausch eingesetzt wird und deswegen auch viel zwischen verschiedenen Formaten konvertiert werden muss, und dass XSLT das zwar kann, aber für viele zu umständlich oder zu wenig intuitiv ist, haben natürlich auch Anbieter kommerzieller Software bemerkt. Mit Altovas MapForce und Progress Softwares Stylus Studio haben zwei Anbieter mit graphisch der ansprechend daherkommenden Tools positioniert, die das gleiche Ziel mit der gleichen Vorgehensweise erreichen wollen. In beiden Fällen kann man mit einfachen graphischen Mitteln Schemas aufeinander abbilden, indem man im einfachsten Fall Linien zwischen Elementen und Attributen zieht, oder diese einfachen Abbildungen durch das Einfügen von weiteren Verarbeitungsschritten ergänzen kann. Nach Abschluss der Abbildungen kann Code generiert werden, der diese

Abbildung dann implementiert, im Fall von MapForce nicht nur XML-orientierter Code (XSLT/XQuery), sondern auch Code in anderen Sprachen.

Die Mapping Tools bieten eine Abstraktion an, die das Arbeiten mit Abbildungen vereinfacht. Auf der anderen Seite zahlt man den Preis jeder Abstraktion: Will man etwas machen, das im Konzept des Mapping Tools nicht vorgesehen ist, hat man Probleme. Nicht optimal geeignet für diese Tools sind Abbildungen mit viel Kontext (unterschiedliche Abbildung eines Elementes abhängig vom Kontext), und Abbildungen die Werte aus verschiedenen Knoten zusammenführen sollen. Natürlich kann man die entsprechenden Aspekte im generierten Code ergänzen, doch dann verliert man den unmittelbaren Zusammenhang zwischen dem vom Tool verwalteten Mapping und der tatsächlichen Abbildung. Ein weiterer Schwachpunkt der Tools ist die Frage der Umkehrbarkeit der Abbildungen, also ob man die gleichen Abbildungsvorschriften für Abbildungen in beiden Richtungen verwenden kann, was oft eine sehr nützliche Eigenschaft wäre. Der Nutzen der Tools soll hier keinesfalls in Frage gestellt werden, man sollte sich nur vor der Verwendung eines solchen Tools fragen, ob es das richtige Tool für das Problem ist.

Für das ShaRef Projekt wurde der Nutzen einer sauber getrennten Map (die nicht alle Informationen enthält, die darin enthaltenen aber in abstrakter und umkehrbarer Form) und separatem XSLT Code höher beurteilt als die Benutzerfreundlichkeit der graphisch orientierten Tools, und mit der oben erwähnten HTML-Darstellung der Map kann die darin enthaltene Information recht übersichtlich präsentiert werden.

Schemas semantisch beschreiben

Nachdem jetzt recht ausführlich beschrieben wurde, wie nach einer erfolgten semantischen Abbildung die Implementierung folgen muss (manuell oder halbautomatisch mit einem Mapping Tool), soll abschliessend noch das Thema angegangen werden, ob sich das Feststellen der semantischen Interoperabilität nicht auch automatisieren liesse. Zunächst einmal könnte man vermuten, dass einem XML Schema mit seinen Datentypen hier helfen könnte, doch dem ist leider nicht so. Die Datentypen von XML Schema sind einfach Wertmengen, sie haben keinerlei darüberhinausgehende Bedeutung. Für eine Applikation ist aber essentiell, ob ein Datum das Start- oder das Enddatum einer Aktivität ist, oder ob eine als positive Zahl definierte Entfernung in Kilometern oder in Meilen angegeben wird. Man kann also sehen, dass die Bedeutung von Daten in einem Schema weit mehr ist als nur der XML Schema Datentyp.

Was gefordert ist, ist die Beschreibung des mit einem Datentyp verbundenen Konzepts, also dass es sich bei einem Datum um eine bestimmte Art von Datum handelt. Sobald man im Bereich der Web-Technologien von Beschreibungen von Konzepten redet, fällt das Stichwort *Resource Description Framework (RDF)*, und dies ist tatsächlich auch in diesem Fall die angemessene Technologie. Mit der RDF-Beschreibung von Typen in XML Schemas (seien es Simple oder Complex Types) könnte recht einfach erreicht werden, dass durch das RDF-annotierte Schema klar ersichtlich ist, dass Instanzen dieses Typs ein bestimmtes Konzept beschreiben. Es wäre dann sogar gleichgültig, was für einen Element- oder Attributnamen die Instanz benutzen würde, dies wäre z.B. auch sehr praktisch für mehrsprachig geführte Schemas.

Nach diesem kurzen Exkurs in die Welt des Möglichen nun jedoch zurück auf den harten Boden der Tatsachen: Auch wenn RDF die semantische Beschreibung von XML Schemas technisch ermöglicht, so ist dies momentan noch im Bereich der Forschung anzusiedeln und wird in der Praxis nicht eingesetzt. Aus diesem Grund ist im Bereich der semantischen Interoperabilität von XML Schemas nach wie vor Handarbeit nötig, man muss die XML Schemas verstehen, und erst, wenn man ausgehend von diesem Verständnis eine Menge von Abbildungsvorschriften festgelegt hat, kann man sich daran machen, diese wie beschrieben manuell oder mit Unterstützung eine Mapping Tools zu implementieren.