Einsatz der Extended Markup Language (XML)

Bei dem in den letzten Jahren sich ausbreitenden Hype rund um die Extended Markup Language (XML) erscheint es uns angebracht, ein paar Gedanken zur sinnvollen Verwendung dieses Formats zu Papier zu bringen.

Wir halten hier fest, worum es sich bei XML handelt und wofür es grundsätzlich gedacht ist. Ergänzend geben wir eine Zusammenfassung der wichtigsten Vor- und Nachteile von XML, wie sie sich für uns in der Software-Entwicklung darstellen. Wenn Sie das Material gelesen haben, werden Sie wissen, wofür XML geeignet ist und in welchen Fällen sich sein Einsatz wirklich lohnt.

1. Die Basics

XML ist ein plattformunabhängiges Format zur Repräsentation von Daten. Es ist keine Programmiersprache und schon gar kein Kommunikationsprotokoll.

XML ist im Grunde nichts anderes als eine Vereinbarung, wie Daten strukturiert in Textform zusammengestellt werden. In dieser Vereinbarung liegt auch die große Stärke von XML begründet – der Datenaustausch zwischen unterschiedlichen Systemplattformen. Das ist insofern nützlich, als moderne Enterprise-Lösungen stark von Heterogenität geprägt sind. Oft müssen viele verschiedenartige Systeme mit unterschiedlichen Betriebssystemen in einem größeren Verbund zusammenarbeiten. Um in einer solchen Landschaft Daten von einer Systemplattform auf eine andere zu transportieren, ist XML oft und zu Recht das Mittel der Wahl.

Dabei wird der Programmierer in der Regel von einem XML-Parser unterstützt, mit dem die zu übertragenden Daten in das XML-Format gebracht bzw. wieder ausgelesen werden. Diese Parser (z.B. Xerces oder Microsoft MSXML) stehen im Grunde auf jeder Plattform zur Verfügung. Die Parser bringen dem Programmierer den Vorteil, dass er sich nicht um die syntaktische Richtigkeit der Daten zu kümmern braucht. Für unerfahrene Programmierer kann diese Erleichterung einen Stolperstein bedeuten - die hochgradige Automatisierung durch den Parser verleitet dazu, auf die Prüfung der semantischen Korrektheit der Daten zu vergessen.

Das XML-Format kann auch an spezielle Anwendungen angepasst werden. Das geschieht durch die Metasprachen DTD oder XML-Schema, welche die in einem gegebenen Fall zulässigen Inhalte einer XML-Datei beschreiben. In der Praxis wird oft darauf verzichtet, da das Erlernen der Metasprachen (speziell von XML-Schema) großen Aufwand bedeutet.

XML hält noch einen weiteren Bonus bereit: Mit XSLT, einer besonderen Transformationssprache, können XML-Dateien ineinander übergeführt werden. Das ist zum Beispiel dann von Nutzen, wenn die Daten in einer XML-Datei nicht in dem Format enthalten sind, wie sie von einer Applikation erwartet werden. Um sie entsprechend aufzubereiten, kann XSLT verwendet werden.

2. Die Vorteile von XML

● *Selbstbeschreibend*

XML-Daten beschreiben sich selbst. Wenn man ein XML-File liest, kann man zumeist (d.h. wenn die Namen der Tags sinnvoll gewählt sind) erraten, welche Daten welche Bedeutung haben. Das ist insbesondere während der Entwicklung einer Applikation ein großer Vorteil, solange noch keine vollständigen Dokumentationen existieren.

● *Abwärtskompatibel*

XML ist eine hierarchische Tag-language und erbt somit die Eigenschaft aller Tag-languages, dass sie abwärtskompatibel sind. Das heißt, dass Programme, die eine bestimmte Version eines XML-Files verarbeiten können, auch neuere Versionen parsen können, die zusätzliche Daten aufweisen - sie ignorieren die zusätzlichen Daten dann einfach.

● *Validierung durch Parser*

XML-Daten lassen sich auf sehr detaillierte Art und Weise datengetrieben validieren. "Datengetrieben" heißt, dass man in einer Metasprache, nämlich DTD oder XML-Schema beschreibt, welchen Bedingungen die XML-Daten genügen müssen und dann den XML-Parser diese Validierung übernehmen lässt - normalerweise muss man dazu keine weitere Codezeile schreiben. Dabei können sowohl die Struktur der Tags als auch Werte vielfältig überprüft werden.

● *Entkopplung von Daten und verarbeitendem Code*

Es gibt fertige XML-Parser, etwa Xerces oder MSXML von Microsoft. Es muss somit, um XML-Daten zu verarbeiten, kein Code geschrieben werden, der mit dem Parsen der Daten direkt zu tun hat - erfahrungsgemäß ein besonders fehlerträchtiger Code. Durch die auf diese Art erzielte Entkopplung von Datenformat und verarbeitendem Code ist es leicht, Änderungen des Datenformats im Code zu berücksichtigen (mit Änderung des Datenformats ist hier nicht gemeint von XML nach Nicht-XML sondern die Art bzw. Namen und den Inhalt der Tags). Der Code ist somit flexibel und robust.

● *Plattformunabhängigkeit*

XML ist plattformunabhängig. XML ist ebenso unabhängig von Codepages wie vom byte ordering der speziellen Maschine. Ein XML-File etwa, das auf einer Big Endian Maschine mit EBCDIC Codepage erzeugt wurde, lässt sich auf einer Little Endian / ASCII Maschine problemlos wieder einlesen.

3. Die Nachteile von XML

● *Hoher Lernaufwand für Metasprachen*

Die Datenbeschreibungssprache XML-Schema ist kompliziert und schwer zu erlernen (im Gegensatz zu XML selbst). Das einfachere DTD ist im Auslaufen begriffen und sollte laut W3C durch XML-Schema ersetzt werden - wir empfehlen dennoch durchaus DTD zu verwenden, wenn man in kurzer Zeit ein gutes Schema für Validierungen erstellen möchte.

● *Keine Komprimierung der Daten*

XML-Daten sind schlecht komprimiert: da die Daten im Textformat gesendet werden, sind sie größer als nötig. In Fällen, bei denen die Kommunikationsbandbreite gering ist, erscheint es angeraten, die XML-Daten zu komprimieren (etwa vermittels LZW-Algorithmus) bevor sie gesendet werden. Vor ihrer Verarbeitung müssen sie dann natürlich wieder dekomprimiert werden.

● *Nicht alle Zeichen können Daten sein*

Gewisse Textdaten (insbesondere solche, die viele Sonderzeichen beinhalten, wie etwa '<', und '>') müssen relativ umständlich in so genannten CDATA-Sections codiert werden. Dies wird mit Sicherheit dann nötig, wenn man zum Beispiel Programmcode in XML-Daten einbetten möchte.

● *Parser muss verwendet werden*

Man MUSS in so gut wie allen Fällen einen XML-Parser verwenden, da das eigene Codieren eines XML-Parsers aufgrund der umfangreichen Möglichkeiten nicht angeraten scheint - das bedeutet wiederum, dass man sich eines relativ schwergewichtigen Mechanismus bedienen muss. Das ist überall dort ein Nachteil, wo CPU- und Speicher-Ressourcen rar sind (wie etwa bei Palmtops, PDAs, Handys, etc...)

● *Verarmung der Filetypen*

Es hat sich eingebürgert, XML-Files als einen Datentyp anzusehen und mit der Extension .XML abzuspeichern. Das führt zu einer Verarmung von Filetypen - da nun alle Files, die XML-Daten beinhalten, gleich welchen Inhalts, .XML heißen, kann nur mehr eine Applikation damit assoziiert werden (typischerweise ein XML-Viewer), anstelle einer Applikation, welche die Daten nicht nur auf syntaktischer, sondern auch auf semantischer Ebene versteht und anzeigt. Das ist kein prinzipieller Nachteil von XML an und für sich, sondern eine missverständliche Verwendung. Es sollten weiterhin die Daten nach ihrem Inhalt und nicht nach ihrer Syntax klassifiziert werden und mit File-Extensions versehen werden. (Beispiel: ein File, das Daten für eine CAD-Applikation beinhaltet, sollte zum Beispiel die Extension .CAD bekommen und nicht .XML, auch wenn die Daten im XML-Format gespeichert werden)

4. Empfehlungen

XML hat sich insbesondere als Datenformat zwischen heterogenen Systemen verschiedener Hersteller bewährt. Da erfahrungsgemäß die Kommunikation und Dokumentation sowie schnell veränderliche Rahmenbedingungen in diesen Fällen das Hauptproblem sind, besticht XML hier voll und ganz, XML-Daten dokumentieren sich selbst und auf XML aufbauende Lösungen zeichnen sich durch große Flexibilität und Robustheit aus. Darüber hinaus ist die Plattformunabhängigkeit in diesem Fall das wichtigste Argument.

Als Fileformat ist XML hingegen mit Vorsicht zu genießen – es vergrößert die Datenmenge stark und kann verschiedene Formen von Daten (z.B. Programmcode) nur umständlich abspeichern. Insbesondere sollte man XML nicht aus dem einzigen Grund wählen, dass es abwärtskompatibel ist. Das ist eine Anforderung, die sich mit jedem anderen Tag-basierten Datenformat (etwa mit einem selbst entwickelten und an die Erfordernisse der Applikation angepassten Datenformat) genauso erfüllen lässt.

Vor allem sollte auch die Möglichkeit, binär abzuspeichern, genutzt werden. Wir empfehlen das teils aus Gründen der Datengröße, teils aus Gründen der Datensicherheit. Denn generell möchte man die Benutzer einer Applikation daran hindern, Daten in den Files einer Applikation selbst zu ändern (etwa in einem Editor). XML verführt leider geradezu dazu.

Ergänzend zu diesem Text steht Ihnen auf der InfraSoft Website ein Glossar zur Verfügung, in dem Sie die meisten der hier verwendeten Begriffe finden. Über das aktuelle Angebot an weiteren, kostenlosen Fachbeiträgen zur Softwareentwicklung informieren Sie sich bitte unter www.infrasoft.at/service.

Harald Nowak
Wien, im August 2002

Der Autor ist Mitarbeiter der InfraSoft, einem Unternehmen, das auf komplexe Softwareentwicklungen spezialisiert ist. Die Experten der InfraSoft haben langjährige Erfahrungen in der Entwicklung und verfügen über fundierte Kenntnisse in Design, Analyse, Realisierung, Test und Projektmanagement.
Für **individuelle Beratungen** zur Entwicklung von Softwarelösungen und die Bereitstellung von **Realisierungsteams** wenden Sie sich bitte an info@infrasoft.at.