

Universität Fribourg
Institut für Informatik
1700 Fribourg
Prof. Dr. Andreas Meier

Seminararbeit

Einsatz des DOM für den Datenaustausch zwischen XML Dokumenten und RDB

Fabio Patocchi
Hallerstrasse 25
3012 Bern

22. Oktober 2002

Inhaltsverzeichnis

1	Einleitung	3
1.1	Problemstellung	3
1.2	Zielsetzung	4
1.3	Vorgehensweise	4
2	Grundlagen	5
2.1	eXtensible Markup Language (XML)	5
2.1.1	Einordnung und Bedeutung	5
2.1.2	Grundlagen, Schema und Schnittstellen	6
2.2	Document Object Model (DOM)	10
2.2.1	Das Konzept	10
2.2.2	DOM Architekturen	11
2.3	Mapping Konzepte für relationale Datenbanken	12
2.3.1	Mapping von Attributen	13
2.3.2	Mapping von Werten	17
2.3.3	Ein alternatives Verfahren: Der 2-Phasen Mapping Ansatz	18
3	Der Einsatz vom DOM am Beispiel von AIDA PC	21
3.1	Allgemeines	21
3.1.1	Die relationale Datenbank AIDA PC	21
3.1.2	Anforderungen und Einschränkungen	21
3.2	Konzeption des Datenaustausches	22
3.3	Implementierung	25

3.3.1 Tabellen und Views	25
3.3.2 Prozeduren	28
4 Schlussfolgerungen	30
Abbildungsverzeichnis	31
Literaturverzeichnis	32

1 Einleitung

Diese Arbeit befasst sich mit der Ausarbeitung eines Konzeptes für den Informationsaustausch zwischen verteilten relationalen Datenbanken mit Hilfe der eXtensible Markup Language (XML). Gegenstand und Anwendungsbereich der Arbeit ist die Koordination von Projektinformationen zwischen humanitären Organisationen basierend auf dem XML Schema IDML. Die Arbeit beschäftigt sich im speziellen mit der Konzeption und Implementierung einer Importfunktionalität für AIDA PC, einer relationalen Datenbank, die Informationen über verschiedene Organisationen und deren Projekte im humanitären Bereich verwaltet, speichert und bereitstellt.

1.1 Problemstellung

Kommunikation und Informationsaustausch ist die Basis jeder Zusammenarbeit zwischen verschiedenen Akteuren. Akteure können sowohl Menschen wie auch Maschinen oder Systeme sein. Im Zentrum der Betrachtung stehen bei uns computergestützte Systeme [HUESE 02]. Eine computergestützte automatisierte Gestaltung von Informationsaustauschprozessen zwischen dezentralisierten, heterogenen Systemen erfordert eine einheitliche Strukturierung der Informationen und eine Kommunikationsplattform als Datenautobahn für den Datentransport. Die Herausforderung liegt nun darin, einerseits eine gemeinsame Sprache zu finden, die die Kommunikation zwischen den Informationssystemen gewährleistet, und andererseits den verschiedenen Systemen die gemeinsame Sprache beizubringen, d.h. die Sprache im System zu implementieren. Die für die Bedürfnisse im humanitären Bereich ausgerichtete Sprache konnte mit Hilfe von XML

standardisiert bzw. definiert werden. Es resultierte das XML Schema IDML (International Development Markup Language) [HUESE 02]. Ein Problem, das sich nun bei XML stellt, ist die Frage nach der Art der Datenspeicherung. Ein Lösungsansatz besteht in der Speicherung der Informationen in Datenbanken. An dieser Stelle soll gezeigt werden, wie der Datenaustausch zwischen einer relationalen Datenbanken und XML Dokumenten dargestellt werden kann.

1.2 Zielsetzung

Die Arbeit verfolgt die Zielsetzung, eine Einführung in die Metasprache¹ XML zu geben, die Beschreibung sowie den Einsatz der Applikationsschnittstelle DOM (Document Object Model) aufzuzeigen sowie verschiedene Mapping Ansätze für XML-Dokumenten in relationale Datenbanken (RDB) kennenlernen.

Im praktischen Teil der Arbeit wird dann der Datenaustausch zwischen XML-Dokumenten, die auf dem IDML Schema basieren, und der AIDA PC Datenbank modelliert und technisch implementiert werden.

1.3 Vorgehensweise

Im ersten Teil liefert die Arbeit eine theoretische Abhandlung zu XML und führt den Leser in die standardisierte Schnittstelle DOM ein. Der zweite Teil wird ein Konzept und die Implementierung für den Informationsaustausch zwischen XML Dokumenten und AIDA PC aufzeigen.

¹Sprache, die verwendet werden kann, um andere Sprachen wie IDML zu definieren. vgl. S.112 in [XMLHND 00]

2 Grundlagen

2.1 eXtensible Markup Language (XML)

2.1.1 Einordnung und Bedeutung

XML heisst eXtensible Markup Language und stellt eine einfache und sehr flexible Textformatierungssprache dar, die als Standard des World Wide Web Consortium (W3C) basierend auf SGML (Structured Generalized Markup Language) entwickelt wurde. XML sowie HTML (Hypertext Markup Language) sind Auszeichnungssprachen (Markup). Während HTML aber auf einer festen Struktur von Markierungszeichen (engl.:tag) aufbaut, können mit XML frei wählbare Tags semantisch beschrieben werden. Aufgrund dieser Eigenschaft wird XML auch als strukturbeschreibende Metasprache bezeichnet [IS 01]. Eine weitere Unterscheidung der beiden Formen lässt sich in ihrem Hauptzweck erkennen. HTML wurde konzipiert, um der Darstellung von Daten zu genügen, XML konzentriert sich hingegen auf die Datenübertragung. Dieser Grundanspruch und die Flexibilität verleihen XML insbesondere bei Datenaustauschprozessen zwischen heterogenen Systemen eine wichtige Rolle. Zusammenfassend lassen sich folgende wichtige Vorteile von XML gegenüber HTML auflisten:

- Universellere Methode zur Datenbeschreibung
- Möglichkeit der Datenmanipulation
- Freie Definition von Tags
- Universelle Methode zur Datendarstellung

- Verweise auf mehrere Quellen

Aufgrund der gerade erwähnten Vorteile gewinnt XML immer mehr an Bedeutung. Dies zeigt sich vor allem bei der breiten Anwendung von Web Applikationen. In den folgenden Kapiteln sollen nun die wichtigsten Elemente von XML kurz beschrieben werden.

2.1.2 Grundlagen, Schema und Schnittstellen

Grundlagen

XML Dokumente bestehen aus einem oder mehreren Elementen. Elemente werden von einem Anfangs-tag und einem Schluss-tag gekennzeichnet. Als Beispiel (vgl. Abbildung 2.1) könnte ein Element `activity` folgendermassen definiert werden `<activity>Bau Wasserreservat</activity>`. Daraus wird erkennbar, dass die Daten von den tags `<activity>` und `</activity>` umgeben sind. Die semantische Bedeutung des Elementes wird vom Namen des tags festgelegt. In unserem Fall entspricht der Bau eines Wasserreservats einem Projekt bzw. einer Aktivität. Elemente können in einem Dokument beliebig tief verschachtelt werden, wobei sie abgesehen vom Wurzelement (engl.: root element) direkt nur ein übergeordnetes Element (parent) aufweisen dürfen. Diese Form der Strukturierung entspricht der Baumstruktur in der hierarchischen Modellierung [SLASH 02].

Eine weitere Möglichkeit, Daten einzubinden, bietet die Definition von Attributen. Attribute haben die Eigenschaft, dass sie direkt Elementen zugeteilt werden, aber keine Unterkategorien erlauben. Sie haben für das zugehörige Element den Charakter von Zusatzinformationen. In der Abbildung stellen *origin*, *dbKey* und *date* die Attribute für das Element `<activity>` dar.

Um den Datenaustausch fehlerfrei durchführen zu können, müssen XML-Dokumente die folgenden zwei Anforderungen erfüllen. Sie müssen einerseits gewissen XML-Spezifikationen genügen, d.h. syntaktisch korrekt sein (engl.: well-formed; eine nähere Beschreibung dafür findet sich bei [W3C 00]), andererseits den Verweis auf eine vordefinierte Struktur (Document Type Definition oder XML Schema) enthalten, die die Markierungszeichen bzw. tags des Dokumentes beschreibt. Letzteres ermöglicht schliesslich die Validierung

```

<activity origin="1", dbKey="1", date="2002-1-1">
    <id>1</id>
    <title>Bau Wasserreservat</title>
    <location>Nairobi</location>
    <orgInvolved>
        <org dbKey="134", origin="89">
            <startDate date="2003-1-1">
            <endDate date="2004-3-31">
        </orgInvolved>
    </activity>
<activity origin="2", dbKey="2", date="2001-10-10">
    <id>2</id>
    <title>Einrichtung einer Schule</title>
    <location>Nairobi</location>
</activity>

```

Abbildung 2.1: Beispiel eines IDML Dokumentes

des XML-Dokumentes (engl.: valid).

DTD/XML Schema

Die DTD (Document Type Definition) definiert alle Elemente und Attribute in XML-Dokumenten und stellt deren logischen Bezug innerhalb des Dokumentes dar. Es werden drei mögliche Assoziierungsformen unterschieden:

- keine DTD: Die Dokumente können nur nach ihrer Wohlgeformtheit (well-formed), aber nicht nach der Validität (valid) überprüft werden.
- interne DTD: Das DTD ist im XML-Dokument selbst integriert und wird in einem

DOCTYPE Bereich deklariert.

- externe DTD: Das XML-Dokument verweist im DOCTYPE Bereich auf ein externes DTD-Dokument (*.dtd). Das DTD-Dokument beinhaltet die Beschreibung der Elemente und Attribute.

Eine Veranschaulichung der Syntax anhand von Beispielen kann in [W3SCHOOL 02] gefunden werden.

Das XML Schema (XSD) wurde am 2. Mai 2001 als "Recommendation" von W3C herausgegeben und dient als Standard zur Beschreibung von Struktur, Inhalt und Semantik von XML-Dokumenten. Im Gegensatz zu DTD, das über eine begrenzte Ausdruckstärke verfügt, erfüllt XSD die Anforderung nach einer differenzierten Gestaltung von Typisierungsformen sowie der Einschränkung von Wertebereichen. Um Konflikte bei der Namensgebung der Elemente zu vermeiden, können Namensräume deklariert werden, die über eine einmalige URI (Uniform Resource Identifier) voneinander unterschieden werden.

Als Beispiel zeigt die Abbildung 2.2 aus [HUESE 02] die wichtigsten Elemente des IDML Schemas, das einen offenen XML Standard zur Strukturierung von Informationen für den internationalen humanitären Bereich darstellt. Diese Elemente weisen allgemeine Projektinformationen auf, die von [BELLANET 02] als *AIDA core metadata* bezeichnet und unter den teilnehmenden Organisationen ausgetauscht werden. Das IDML Schema unterscheidet weiter zwischen den Kardinalitäten "multiple" (0 oder mehrere), "optional" (0 oder 1) und "required" (genau 1). Eine graphische Darstellung des gesamten IDML Schemas (Version 0.91) kann im Anhang von [HUESE 02] gefunden werden.

Schnittstellen

Unter einer Schnittstelle versteht man ein Programm, das eine Beziehung oder Verknüpfung zwischen zwei verschiedenen Programmen herstellt. Um XML-Dokumente in Programmen leicht weiterzuverarbeiten, mussten also Schnittstellen geschaffen werden, die einem festgelegten Standard genügen. Das W3C hat dafür die Schnittstelle DOM

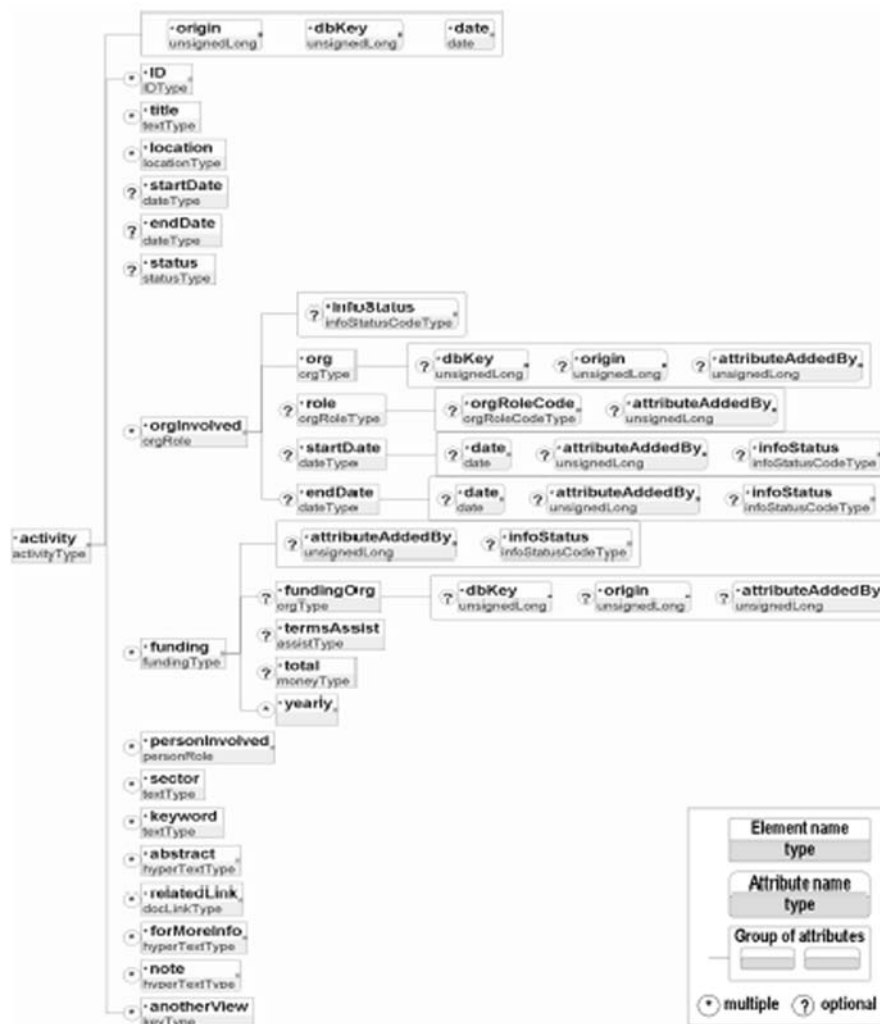


Abbildung 2.2: Grafische Übersicht vom IDML Schema

(Document Object Model) modelliert. In einer offenen Diskussionsgruppe von Benutzern und Entwicklern wurde als ereignisorientierte Alternative zu DOM SAX (Simple API for XML) entwickelt. Beide sind plattformunabhängig und bieten einen Standard für den Zugriff auf XML-Dokumente. Der wesentliche Unterschied der beiden Schnittstellen liegt in der Art und Weise des Zugriffs auf Daten. Während beim SAX lediglich der Lesezugriff unterstützt wird, ist mit DOM ebenfalls der Schreibzugriff auf XML-Dokumente realisiert. Weitere Unterschiede zwischen beiden Schnittstellen können in [SCHAE 02] nachgelesen werden. Im nächsten Kapitel beschäftigen wir uns intensiver mit DOM,

später wird dann der Einsatz von DOM anhand eines Beispiels veranschaulicht.

2.2 Document Object Model (DOM)

2.2.1 Das Konzept

Das DOM beschreibt ein Konzept zur Manipulation von XML Dokumenten und erlaubt damit den Datenaustausch zwischen Programmen und XML Dokumenten. Das XML Dokument, das aus einer Textdarstellung besteht, wird zuerst in eine Reihe von begrifflichen Objekten umgewandelt. Dieser Vorgang wird *Parsing* genannt. Dabei untersucht der DOM Parser das XML Dokument nach syntaktischen Fehlern und validiert das XML Dokument gegen sein Schema. Nach erfolgreicher Validierung generiert der DOM Parser einen DOM Baum (engl.: Tree), welcher aus sog. DOM Objekten besteht und das gesamte XML Dokument in hierarchischer Form wiedergibt. Ausgehend vom DOM Baum können Programme auf Daten zugreifen, sie im DOM Baum direkt verändern und daraus XML Dokumente aktualisieren bzw. neue Dokumente erstellen. In Anlehnung an [DOMKONZ 02] zeigt die Abbildung 2.3 das DOM Konzept für die Microsoft Implementierung (MSXML).

Der objektnahe Bezug von DOM lässt sich aufgrund der folgenden zwei Eigenschaften erkennen: Dokumente werden unter Verwendung von Objekten modelliert und das Modell gibt nicht nur die Struktur des Dokumentes wider, sondern beschreibt ebenfalls das Verhalten des Dokumentes und seiner Objekte. Das DOM als Objektmodell identifiziert somit

- Schnittstellen und Objekte, die zur Darstellung und Manipulation des Dokumentes verwendet werden
- die semantische Bedeutung dieser Schnittstellen und Objekte (inkl. der Attribute und Verhaltensmethoden)
- die Beziehungen zwischen den Schnittstellen und Objekten wie auch ihre Koordination.

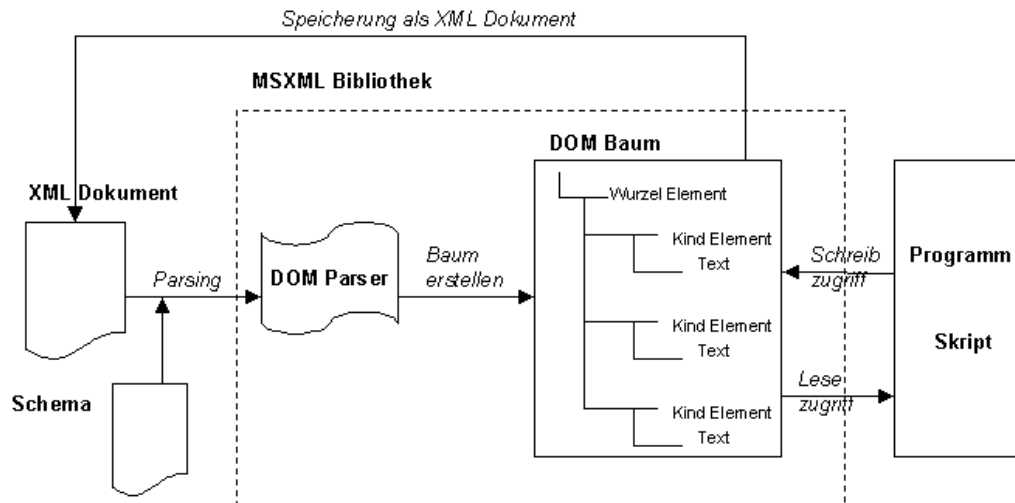


Abbildung 2.3: Konzept für die DOM Implementierung von MSXML (Microsoft XML)

2.2.2 DOM Architekturen

Die DOM Architektur setzt sich aus mehreren Modulen zusammen, die sich an verschiedenen Umgebungen (z.B. XML, HTML, Cascading Style Sheets (CSS)...) orientieren. Das *DOM Core* repräsentiert die Baumstruktur des Dokumentes. Es ermöglicht dem Benutzer, sich durch den Baum frei zu bewegen, die Baum-Objekte anzusprechen und zu manipulieren. Eine Erweiterung der CORE Plattform auf XML 1.0 wurde mit *DOM XML* realisiert. Hier haben XML-spezifische Funktionen (Prozessanweisungen, CDATA und Elemente) Berücksichtigung gefunden. *DOM HTML* und *DOM CSS* definieren Funktionssammlungen, mit denen HTML Dokumente bzw. CSS Vorlagen manipuliert werden können. Benutzerschnittstellen, HTML Objekte oder Manipulationen von XML Bäume können Ereignisse auslösen. Ihre Integration ist im Modul des *DOM Events* realisiert. Die Module DOM Load and Save (Operationen zur Kontrolle von Lade- und Speicherprozessen), DOM Validation (Validierung eines veränderten DOM-Baumes) und DOM XPath (Generierung von Abfragen mit XPath 1.0 auf DOM-Bäume) sind zur Zeit noch in der Entwicklungsphase.

Im Laufe der Zeit hat sich DOM ständig weiterentwickelt und neue Umgebungen

integriert. Tabelle 2.2.2 zeigt die wichtigsten Entwicklungsstadien von DOM bis zum heutigen Zeitpunkt auf.

<i>Level</i>	<i>Weiterentwicklungen</i>	<i>W3C-Status</i>
0	Funktionen gemäss Netscape 3.0/Internet Explorer 3.0	keine Spezifik.
1	XML 1.0, HTML 4.0	seit Oktober 1998
2	XML-Namensräume, CSS, Ereignisunterstützung	seit November 2000
3	Validierung, Laden/Speichern von Dokumenten Erweiterungen der Schnittstellenereignisse (user interface events)	in Entwicklung

Tabelle 2.1: Entwicklungsstadien der DOM Architekturen im Zeitverlauf

2.3 Mapping Konzepte für relationale Datenbanken

Wir haben im Kapitel 2.1.1 festgestellt, dass XML im Vergleich zu HTML breitere Einsatzmöglichkeiten bietet und der Grundansatz im Transfer von Daten liegt.

Bei der Datenübertragung aus einem XML-Dokument in eine relationale Datenbank (RDB) ist es oft sinnvoll, gewisse Informationselemente bzw. -kategorien unberücksichtigt zu lassen. Es liegt nahe, dass Daten wie Dokumentinformationen (z.B. Dokumentname, DTD), die physische Struktur des Dokumentes (z.B. Elementdefinition) oder die logische Struktur (z.B. Reihenfolge der Attributwerte) in der Datenbank keine Verwendung finden. Beim Datentransfer von einer Datenbank in ein XML-Dokument andererseits werden aus ähnlichen Überlegungen Elementdefinitionen fehlen, und die Reihenfolge der Attributwerte sich aus der Datenbankausgabe ergeben.

Eine grundsätzliche Anforderung für Datentransfer-Software wird die Berücksichtigung der hierarchischen Ordnung sein, d.h. welche Elemente/Attribute welchen Gruppen angehören. Als Beispiel kann hierfür der Fall genannt werden, bei dem Informationen zu einem Projekt im humanitärem Bereich von einer RDB in eine andere transferiert werden sollen. In diesem Fall wird weder eine Rolle spielen, ob die Projektnummer oder das Datum des Projektbeginns zuerst oder danach in der Datenbank gespeichert wird,

noch ob der Name der Organisation als CDATA Bereich, Element oder Attribut im XML-Dokument dargestellt wird. Es wird einzig von Bedeutung sein, ob alle relevanten Daten aus der ersten Datenbank in die zweite einfließen.

Im folgenden betrachten wir den Datentransfer aus einem XML-Dokument in eine RDB. Ausgangspunkt ist der XML-Baum, der den Aufbau eines XML-Dokumentes beschreibt. Ein Mapping Schema legt fest, welche Tabellen in der Datenbank erstellt werden sollten, welche Indizes zu bilden sind und in welchen Tabellen die Elemente, Attribute und die Werte gespeichert werden sollen. [FLOR 99] unterscheidet zwei Arten von Dimensionen: Die Repräsentierung von *attributes* (Attribute) und die von *values* (Werte). Zu jeder Dimension beschreibt er mehrere Ansätze, die in den folgenden zwei Kapiteln besprochen werden.

2.3.1 Mapping von Attributen

Edge Ansatz

Der Edge Ansatz (auf deutsch: Rand) wird als einfachstes Mapping Schema bezeichnet. Hier werden alle Attribute in einer einzigen Tabelle gespeichert. Die Struktur der Edge Tabelle kann wie folgt beschrieben werden:

$$Edge(source, ordinal, name, flag, target) \quad (2.1)$$

Die Edge Tabelle besteht somit aus der Identifikation des Quellobjektes (*source*), dem Zielobjekt für das Attribut (*target*), seinem Namen (*name*), einem sog. *flag*, das die Attributausprägung als Referenz auf ein *oid* (object identifier) oder als Wert definiert, und schliesslich einer Ordinalnummer (*ordinal*), die zur Festlegung der Reihenfolge verwendet wird. In Anlehnung an Abbildung 2.1 auf Seite 7 lässt sich folgende Edge Tabelle für unser Beispiel aufstellen. Der Primärschlüssel wird von $\{source, ordinal\}$ gebildet. Für die Indizierung schlagen die Autoren eine Kombination von $\{source\}$ und $\{name, target\}$ vor. Damit soll sowohl die Vorwärtssuche (z.B. Suche alle Attribute der *source* 2) wie auch die Rückwärtssuche (z.B. Suche alle Aktivitäten, die von der Organisation 134 geführt werden) gewährleistet werden können. Eine Variante des Edge Ansatzes wäre,

<i>source</i>	<i>ordinal</i>	<i>name</i>	<i>target</i>
1	1	date	2002-1-1
1	2	id	1
1	3	title	Bau Wasserreservat
1	4	orgInvolved	134
2	1	date	2001-10-10
2	2	id	2

Tabelle 2.2: Bsp: Edge Tabelle

die Attributnamen in einer einzigen gesonderten Tabelle aufzuführen. Diese Massnahme würde zwar zu einer Reduktion der Datenbankgrösse führen, andererseits aber die Abfrageprozessleistung stark verringern.

Attribut Ansatz

Beim Attribut Ansatz wird vorgeschlagen, alle Attribute mit gleichem Namen in gesonderten Tabellen zu speichern. Dieser Ansatz entspricht konzeptionell einer horizontalen Partitionierung des Edge Ansatzes, bei dem der Partitionsname durch den *name* gegeben wird (vgl. hierzu die Struktur 2.1). Folgende Struktur kann somit abgeleitet werden.

$$A_{name}(source, ordinal, flag, target) \quad (2.2)$$

Wie beim Edge Ansatz wird der Primärschlüssel auf die Felder $\{source, ordinal\}$ gesetzt. Bezüglich der Bedeutung der einzelnen Felder kann auf den Edge Ansatz verwiesen werden. Tabelle 2.3 stellt für unser XML Beispiel die Attribute in den Tabellen *location* und *orgInvolved* dar.

Universaltabelle

Ein dritter Ansatz beschreibt die Speicherung aller Attribute eines XML Dokumentes in eine Universaltabelle. Sie ist dadurch gekennzeichnet, dass für jedes Attribut des XML

$A_{location}$			$A_{orgInvolved}$		
<i>source</i>	<i>ordinal</i>	<i>target</i>	<i>source</i>	<i>ordinal</i>	<i>target</i>
1	1	Nairobi	1	1	134
2	1	Nairobi	2	1	168
3	1	Johannesburg	2	2	172

Tabelle 2.3: Bsp: Zwei Attribute Tabellen

<i>source</i>	...	<i>ord_{id}</i>	<i>targ_{id}</i>	<i>ord_{locat}</i>	<i>targ_{locat}</i>	...	<i>ord_{orgI}</i>	<i>targ_{orgI}</i>
1	...	2	1	1	Nairobi	...	1	134
2	...	2	1	Null	Null	...	2	168
2	...	2	2	1	Nairobi	...	3	172
2	...	2	2	2	Johannesburg	...	Null	Null
3	...	2	3	1	Johannesburg	...	1	153

Tabelle 2.4: Bsp: Universaltabelle

Dokumentes separate Spalten erstellt werden. Konzeptionell entspricht dieser Ansatz einem *outer join* (vgl. hierzu [IBM 01]) aller Attribute Tabellen zueinander. Hier kann ihre Struktur wie folgt dargestellt werden:

$$Universal(source, ordinal_1, flag_1, target_1, \dots, ordinal_k, flag_k, target_k) \quad (2.3)$$

In der Tabelle 2.4 wird ein Beispiel einer Universaltabelle aufgezeigt. Auffallend hierbei sind die nicht unwesentlich hohen Anteile an leeren Feldern, die auf eine hohe Redundanzdichte schliessen lassen. In Anlehnung an den Attribut Ansatz sollten hier die Indizes auf die *source* wie auch auf allen *target* Spalten gesetzt werden.

Normalized Universal Ansatz (UnivNorm)

Als Variante zur Universaltabelle wird die UnivNorm vorgestellt. Diese Variante unterscheidet sich zur oben dargestellten Universaltabelle darin, dass die mehrwertigen Attribute in separaten *Overflow* Tabellen gespeichert werden.

$UniNorm(source, ordinal_1, flag_1, target_1, \dots, ordinal_k, flag_k, target_k)$

$Overflow_1(source, ordinal, flag, target), \dots, Overflow_k(source, ordinal, flag, target)$
(2.4)

<i>UniNorm</i>								
<i>source</i>	...	<i>ord_{id}</i>	<i>flag_{id}</i>	<i>targ_{id}</i>	...	<i>ord_{orgI}</i>	<i>flag_{orgI}</i>	<i>targ_{orgI}</i>
1	...	2	-	1	...	1	-	134
2	...	2	-	2	...	2	m	-
3	...	2	-	3	...	1	-	153
4	...	2	-	4	...	1	Null	Null

<i>Overflow_{orgI}</i>		
<i>source</i>	ord	target
2	2	168
2	3	172

Tabelle 2.5: Bsp: UniNorm und Overflow Tabelle

Die Struktur in 2.4 wird nun anhand eines Beispieles (vgl. Tabelle 2.5) beschrieben. Wiederum setzt sich der Primärschlüssel der UniNorm Tabelle aus dem *source* zusammen, während die Tupel in den Overflow Tabellen vom $\{source, ordinal\}$ identifiziert werden. Das *flag* Feld gibt nun folgende Hinweise:

- Objekt mit einwertigem Attribut: Das *flag* macht Aussagen über eine Referenz auf ein anderes Objekt oder über einen Wert.
- Objekt mit mehrwertigen Attributen: In der UniNorm Tabelle wird im *flag* Feld ein "m" vermerkt, und alle Attributwerte werden in der Overflow Tabelle aufgeführt.
- Objekt enthält das Attribut nicht: Das *flag* wird auf Null gesetzt.

Das Beispiel weist auf den Zusammenhang zwischen der *flag_{orgI}* Spalte in der UniNorm Tabelle und der Overflow Tabelle von *orgInv* hin und zeigt die oben erwähnten drei Ausprägungsformen für das *flag* Feld auf. Ähnlich wie bei der Universaltablelle sollten hier die Indizes in sämtlichen Tabellen (UniNorm und Overflows) für die Spalten *source* und *target* gesetzt werden.

2.3.2 Mapping von Werten

getrennte Wertetabellen

Eine erste Möglichkeit Werte zu speichern, ist, sie in datentypenspezifische Tabellen (Wertetabelle) zu hinterlegen. In diesem Fall werden bspw. alle alphanumerischen und numerischen Werte sowie Datumsformate getrennt nach ihren Datentypen in verschiedenen Tabellen verwaltet. Die Struktur einer solchen Wertetabelle sieht wie folgt aus:

$$V_{type}(vid, value) \tag{2.5}$$

Entscheidend an dieser Struktur ist der Schlüssel *vid* (value identifier), der den eindeutigen Verweis auf die Objekttablelle liefert.

<i>source</i>	<i>ordinal</i>	<i>name</i>	<i>flag</i>	<i>target</i>	<i>V_{int}</i>		<i>V_{date}</i>	
1	1	date	date	<i>v1</i>	<i>vid</i>	value	<i>vid</i>	value
1	2	id	int	<i>v2</i>	v2	1	v1	1.1.2002
1	3	title	string	<i>v3</i>	v4	134	v5	10.10.2001
1	4	orgInvolved	int	<i>v4</i>	v6	2		
2	1	date	date	<i>v5</i>				
2	2	id	int	<i>v6</i>				

Tabelle 2.6: Bsp.: Edge Tabelle mit separaten Wertetabellen

Ausgehend vom Edge Ansatz erkennt man im Beispiel der Tabelle 2.6, dass die Werte

in Abhängigkeit der Ausprägungen aus der *flag* Spalte bzw. ihrer Datentypen in verschiedenen Wertetabellen (V_{int} oder V_{date}) gespeichert werden und durch Vergleich der *target* Werte mit den *vids* den Objekten ($\{source, ordinal\}$) eindeutig zugewiesen werden. An dieser Stelle sollte vollständigshalber noch angefügt werden, dass, obwohl dieses Beispiel es nicht vorsieht, Referenzen im *flag* auch weiterhin auf ein oid gemacht werden können. So wäre es denkbar, Tupel mit den Ausprägungen "ref" im *flag* und der oid-Nummer im *target* zu kennzeichnen. Indiziert sollen jeweils die *vids* und *value* Spalten von allen Wertetabellen.

Inlining

Eine naheliegende Variante zu den separaten Wertetabellen stellt das "Inlining" dar. Hier werden die Werte und Attribute in der gleichen Tabelle gespeichert. In Anlehnung an den Edge Ansatz stellt diese Variante konzeptionell einen *outer join* zwischen der *Edge* Tabelle und den *Value* Tabellen dar, was die Bildung von Spalten für jeden Datentyp zur Folge hat. Das Beispiel in Tabelle 2.7 veranschaulicht diese Methode anhand des Attribut Ansatzes (vgl. Tabelle 2.3). Man erkennt, dass das *flag* zwar nicht mehr verwendet wird, aber dafür eine hohe Zahl an Null-Werten in Kauf genommen werden muss. Zur Indizierung der Tabellen sollten ergänzend zur *source* und *target* alle *val* Spalten eingesetzt werden.

Sowohl das Inlining wie auch die Methode der getrennten Wertetabellen sind nicht nur für die in den Beispielen dargestellten Ansätze (Edge resp. Attribut Ansatz) vorgesehen, sondern können für alle bisher diskutierten Ansätze (vgl. Kapitel 2.3.1) angewendet werden.

2.3.3 Ein alternatives Verfahren: Der 2-Phasen Mapping Ansatz

Ein alternatives Verfahren zu den vorangegangenen Ansätzen stellt der 2-Phasen Mapping Ansatz dar. Diese Strategie verfolgt in der ersten Phase die Speicherung der XML-Daten als normalisierte Baumstruktur (engl.: *raw tree data*) in der RDB, ohne Mapping Regeln anwenden zu müssen. Während der zweiten Phase werden schliesslich vordefiniert-

<i>A_{location}</i>				
<i>source</i>	<i>ord</i>	<i>val_{int}</i>	<i>val_{string}</i>	<i>target</i>
1	1	Null	Nairobi	Null
2	1	Null	Nairobi	Null
3	1	Null	Johannesburg	Null

<i>A_{orgInvolved}</i>				
<i>source</i>	<i>ord</i>	<i>val_{int}</i>	<i>val_{string}</i>	<i>target</i>
1	1	134	Null	Null
2	2	168	Null	Null
2	3	172	Null	Null

Tabelle 2.7: Bsp.: Attribut Tabellen mit Inlining

te Regeln (z.B. Prozeduren) für den Datenaufbereitungs- und -speicherungsprozess (z.B. SQL Transformationsprozesse) angewendet. Der grosse Vorteil dieses Verfahrens liegt darin, dass vorerst eine universelle Speicherung von "well-formed" XML-Dokumenten in der RDB gewährleistet wird und damit, im Gegensatz zu vollautomatisierten Verfahren, eine Speicherung der Daten vom Transformationsprozess kontrolliert durchgeführt wird.

In Abbildung 2.4 werden die Tabellenstrukturen zur Darstellung des *raw tree* in Anlehnung an [DAYEN 02] gezeigt: Die Implementierung dieses Ansatzes wird im Rahmen des AIDA PC Projektes im Kapitel 3.3 vorgestellt.

Knoten-Baumstruktur:

NodeDef	Feld	Wert	Beschreibung
	(NodeID	NOT NULL	Identifikation des Knotens
	DimID	NOT NULL	Baumkategorie: XML
	Name	NOT NULL	Name des Knotens
	Value	NULL	Wert des Knotens
	ValueType	NULL	Datentyp des Wertes (z.B. string, date, integer)
	Power	NULL	Baumstufe bzw. Baumhierarchie
	IsRoot	NOT NULL	Ist der Knoten Wurzelement?
	Description	NULL	DOM Knotentyp
	Color)	NULL	informelles Datenobjekt

Vater-Kinder Beziehung:

NodeLinks	Feld	Wert	Beschreibung
	(ParentID	NOT NULL	Identifikation des übergeordneten Knotens (Vater)
	ChildID)	NOT NULL	Identifikation des untergeordneten Knotens (Kind)

Abbildung 2.4: Raw tree Struktur

3 Der Einsatz vom DOM am Beispiel von AIDA PC

3.1 Allgemeines

3.1.1 Die relationale Datenbank AIDA PC

AIDA PC ist eine relationale Datenbank, die auf Microsoft Access 2000 entwickelt wurde, um einfache Projektinformationen von humanitären Aktivitäten zwischen den involvierten Organisationen auszutauschen. Oft verfügen diese Gruppen weder über technisches Wissen noch über eine adäquate technische Infrastruktur, die ihnen den Zugang zu neuen Technologien eröffnet. [HUESE 02] nennt diese Gruppen *low-tech organizations*. AIDA PC dient dem Zweck, diese Organisationen bei Informationsaustauschprozessen mit leicht und kostengünstig implementierbaren Hilfsmitteln und ohne XML Kenntnisse zu unterstützen. Die breite Nutzung von AIDA PC fördert somit den Informationsaustausch zwischen den Gruppen und erhöht die Nutzung und Akzeptanz von XML-Technologien.

3.1.2 Anforderungen und Einschränkungen

Der Einsatz von AIDA PC kann nur bei Erfüllung einer minimalen Anzahl von Software Anforderungen und unter Berücksichtigung gewisser Einschränkungen erfolgreich gewährleistet werden. Im folgenden erfolgt eine Auflistung der technischen Anforderungen und Einschränkungen:

Technische Anforderungen für den Einsatz von AIDA PC

- Betriebssystem Windows 98/2000/NT/XP
- Microsoft Access 2000
- Microsoft XML Core Services 4.0 SP1; Genaue Installationsbeschreibung kann unter [MSXML40 02] abgerufen werden.

Einschränkungen für AIDA PC und den Datenimport von XML-Dokumenten

- stand-alone version (Einzelplatzversion): die Daten werden auf der lokalen Festplatte gespeichert. Der Mehrbenutzerbetrieb über das Netzwerk wird somit nicht unterstützt.
- Der XML-Datenimport basiert auf Definitionen vom IDML Schema V. 0.91.
- Beim Datenimport wird von einem aktualisierten Stammdatenbestand ausgegangen. Diese Einschränkung hat zur Folge, dass Stammdaten, die in AIDA PC noch nicht aktualisiert wurden, d.h. nicht enthalten sind, beim Datenimport nicht berücksichtigt werden.
- Bei der Projektsynchronisation wird keine Aktualisierungsfunktion unterstützt. Mit der Option "Replace" beschränkt sich die aktuelle Version lediglich auf das Löschen und Neuanfügen des Projektes. Die Funktion "Replace" führt für Projekte, die in AIDA PC anderen Projekten zugeordnet sind, zu relationalen Integritätsverletzungen und somit nicht zur gewünschten Datenaktualisierung.

3.2 Konzeption des Datenaustausches

Im folgenden Kapitel wird ein Konzept für den Datenaustausch zwischen IDML Dokumenten und der AIDA PC Datenbank vorgestellt. Als Grundlage des hier dargestellten

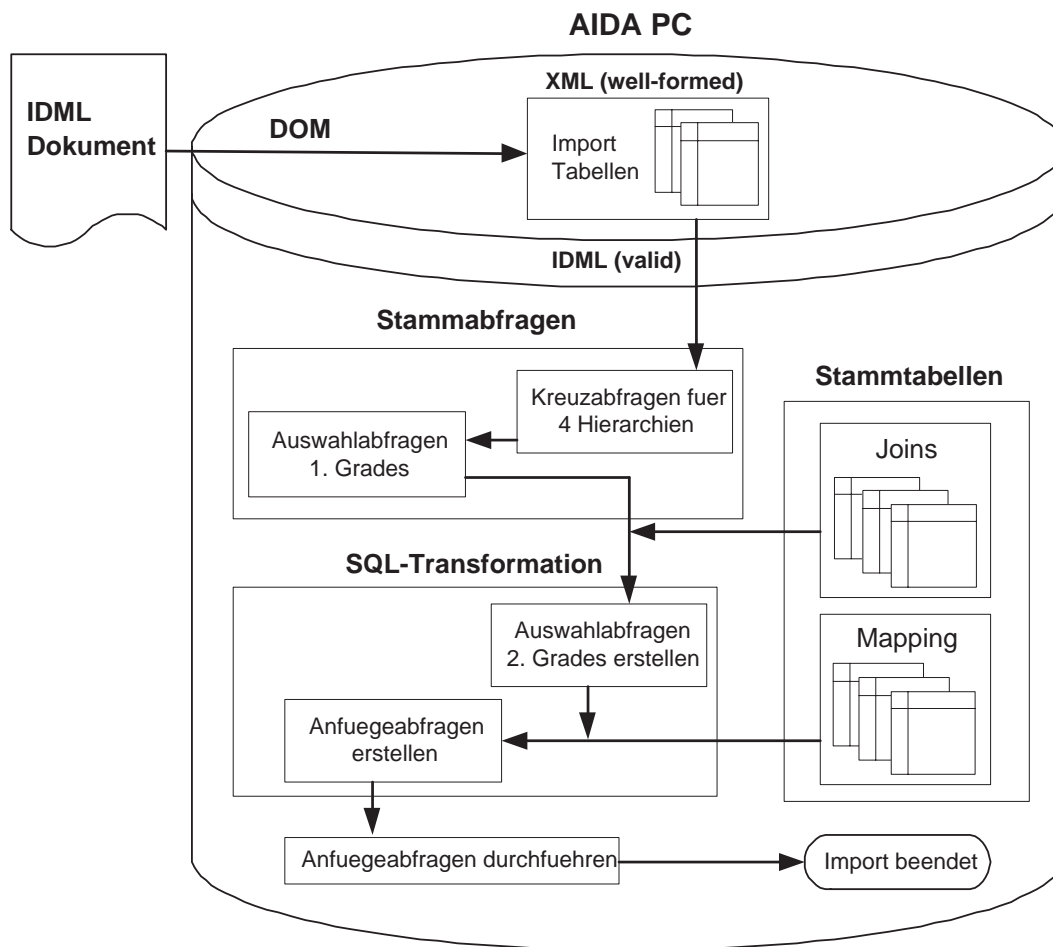


Abbildung 3.1: Konzept für den Import von IDML Dokumenten in AIDA PC

Prozesses dient das in Kapitel 2.3.3 beschriebene Konzept des 2-Phasen Mapping Ansatzes. Der Datenaustausch kann hier in mehrere Phasen, die im folgenden erläutert werden, weiter unterteilt werden. Abbildung 3.1 zeigt grafisch den Prozess des Datenimportes auf.

1. Phase: Die Speicherung der Baumstruktur

Der Ausgangspunkt ist ein IDML-Dokument. Damit das XML-Dokument von der Datenbank interpretiert und gelesen werden kann, muss eine Schnittstelle eingesetzt werden. Der Einsatz von DOM hat sich wegen seiner guten Implementierung für die Microsoft Access Datenbank und seiner flexiblen Einsatzmöglichkeiten als sinnvoll erwiesen. Damit

nun mit DOM auf das XML-Dokument zugegriffen werden kann, muss es zuerst geparkt werden. Damit wird sichergestellt, dass es sich beim Dokument um ein validiertes IDML-Dokument handelt. Weitere Prozeduren zur Validierung werden somit überflüssig. An dieser Stelle sollte darauf hingewiesen werden, dass die erste Phase des 2-Phasen Mappings nur die Bedingung eines "well-formed" Dokuments verlangt. In diesem Fall müsste aber die Validierung bis zur Phase des Transformationsprozesses verschoben werden. Die aus dem DOM-Baum eingelesenen Daten werden nun in den beiden Importtabellen gespeichert. In der ersten werden die Entitäten der Knoten-Baumstruktur, in der zweiten die Vater-Kind Beziehungen gespeichert.

2. Phase: Stammabfragen

Die Stammabfragen stellen fix programmierte Abfragen dar, die auf die beiden Importtabellen zugreifen und die IDML Struktur aufweisen. Wir unterscheiden zwei Arten von Stammabfragen: die Kreuzabfragen (cross queries) und die Auswahlabfragen 1. Grades. Die Kreuzabfragen stellen alle aus dem XML-Dokument stammenden Elemente und Attribute für jede Baumhierarchiestufe mit den zugehörigen Werten zusammen. Insgesamt sind für das vierstufige IDML Schema somit vier Kreuzabfragen definiert. Die Auswahlabfragen 1. Grades stellen erstmals eine fix definierte Annäherung der Kreuztabellenprojektionen an die Struktur der Datenbankobjekte dar. Dabei wird jeder Zieltabelle (z.B. activity, abstract, funding, etc.) eine vordefinierte Auswahlabfrage mit allen Feldern, die den Datenaustausch berühren, gegenübergestellt.

3. Phase: SQL-Transformation 1. Phase

In der ersten Phase des SQL-Transformationsprozesses werden den Auswahlabfragen 1. Grades die sog. *Lookup*-Tabellen angehängt. Die *Lookups* haben den Charakter von Stammdaten und stellen Entitäten in der Datenbank dar (z.B. Tabelle LocationLookup). Grundsätzlich liesse sich auch hier das Aktualisierungsproblem der Stammdatenausprägungen bei verteilten Datenbanksystemen erwähnen. Diese Problematik wird insofern ausgeklammert, als hier von fixen Stammdatenbeständen ausgegangen wird (vgl. 3.1.2

auf Seite 21). Die Definitionen für die Anbindung der *lookups* werden in zwei Tabellen, den sog. *joins*, gespeichert. Die SQL-Transformationsphase wird prozedural gesteuert, greift auf die *joins* zu und führt damit die Anbindung der *lookups* durch, was schliesslich zu den Auswahlabfragen 2. Grades führt.

4. Phase: SQL-Transformation 2. Phase

Auch die zweite Phase der SQL-Transformation wird prozedural gesteuert. Dabei werden die Felder aus den Auswahlabfragen 2. Grades ihren Zieltabellen und -feldern eindeutig zugeordnet. Die Definitionen der Zuordnungen von Tabellen und Feldern befinden sich in der sog. *mapping*-Tabelle. Das Produkt dieser Transformationsphase sind Anfügeabfragen, die nach deren einfacher Ausführung den Import beenden.

3.3 Implementierung

Im Rahmen der Implementierung werden einerseits die Strukturen der Tabellen sowie die fix definierten Abfragen und andererseits die Prozesse anhand der einzelnen Prozeduren besprochen.

3.3.1 Tabellen und Views

Die Importtabellen

Es werden 2 Typen von Importtabellen unterschieden vgl. Kapitel 3.2:

- *tmp_imp_IDML*:
Hier werden die Entitäten der Knoten-Baumstruktur gespeichert.
- *tmp_imp_IDML_Relations*:
Sie stellt die Vater-Kind Beziehungen des Dokumentes dar.

Die Stammabfragen

Die vier Kreuztabellen sind mit den Namen *xml_nodes_dim_i* versehen. *i* gibt die Baumhierarchie der Kreuzabfragen von eins bis vier wieder. Die Zeilen der Kreuztabellen beschreiben die Ausprägungen der Elemente für die entsprechende Baumhierarchie. Ihre Spalten stellen ihre Attribute sowie die Elemente der nächstfolgenden Baumhierarchie dar. Die *parent_id* Spalte weist auf das zugehörige übergeordnete Objekt hin. So erkennt man für die Kreuzabfrage *xml_nodes_dim_1*, dass die Zeilen die Ausprägungen vom Wurzelement *activity* abbilden und die Spalten sich aus den Attributen von *activity* (d.h. *origin*, *dbKey* und *date*) und den Elementen der zweiten Baumstruktur (d.h. den Core Elementen) zusammensetzen.

Es gibt eine ganze Reihe von Auswahlabfragen 1. Grades. Charakteristisch für sie ist ihre eindeutige Zuordnung zu einer Zieltabelle in AIDA PC (z.B. *idml_imp_activity* Abfrage ist der *activity* Tabelle zugeordnet). Die Namen der Auswahlabfragen sowie ihre Zuordnungen können in der Tabelle *Import_Entities* mit dem Typvermerk "Import" eingesehen werden. So erkennt man in Abbildung 3.2, dass als Zieltabelle für *oid* "1" bzw. "*idml_imp_activity*" die *oid* "50" also die "activity" Tabelle in der "Import_ID" Spalte angegeben ist. Das Ziel der SQL-Transformation ist es, die Auswahlabfrage 1. Grades so zu transformieren, dass die Abfrage als Anfügeabfrage ausgeführt wird und die zugeordnete AIDA PC-Zieltabelle mit den XML-Daten ergänzt.

Die Stammtabellen

Sämtliche Verknüpfungen mit Stammdatentabellen werden in den *Joins* definiert. Die *IDML_Join* Tabelle definiert alle Verknüpfungsobjekte, die aus Verweisen auf Stammdatentabellen ("LookupTable_ID") und Verknüpfungsfelder ("Lookup_field_ID") sowie aus der Verknüpfungsart ("Join_Type") bestehen.

Die Verknüpfungsverweise auf die Auswahlabfragen 1. Grades befinden sich schliesslich in der Tabelle *IDML_Lookup_Join*. Darin werden den Auswahlabfragen 1. Grades ("Query_ID") die oben besprochenen Verknüpfungsobjekte (Join_ID) und das zu verknüpfende Feld ("Fld_ID") aus der Auswahlabfrage zugeordnet. In Abbildung 3.3 lässt

Import_ Entities:

ID	Name	Type	Import_ ID
1	idml_imp_activity	Import	50
2	idml_imp_id	Import	51
3	idml_imp_title	Import	52
4	idml_imp_location	Import	53
.			
50	activity	Table	
51	ID	Table	

Abbildung 3.2: Ausschnitt aus der Tabelle *Import_ Entities*: Auswahlabfragen 1. Grades

sich leicht erkennen, dass an die Auswahlabfrage *idml_imp_abstract* vier Stammtabellen bei der SQL-Transformation angehängt werden, d.h. dem Feld "translatedBy" das Verknüpfungsobjekt "2", "lang" "5", "ID" "14" und dem "infoStatus" Feld schliesslich die *oid* "1".

Die *Import_ Mapping* Tabelle bietet die schematische Grundlage zur Überführung der transformierten Auswahlabfragen (2. Grades) in Anfügeabfragen. Hiermit wird das eigentliche "Mapping", sprich die definitive Zuordnung der importierten Felder auf die AIDA PC Felder, abgebildet. Abbildung 3.4 zeigt eine Auswahl von Ausprägungen aus dieser Tabelle. Die Struktur der Tabelle wird durch die Spalten *Import_ ID*, *XML_ Tab_ ID*, *XML_ Field_ ID* und *AIDA_ Field_ ID* definiert. Aus der *Import_ ID* Spalte erkennt man den Namen der Auswahlabfrage 1. Grades, die mittlerweile in den 2. Grad überführt wurde. Die Kombination aus *XML_ Tab_ ID* und *XML_ Field_ ID* kennzeichnet eindeutig die Quelle, aus der der Zielwert entnommen werden soll. Dieser Wert wird schliesslich in das AIDA Zielfeld *AIDA_ Field_ ID* transferiert. Man beachte, dass die AIDA PC-Zieltabelle bereits in der Tabelle *Import_ Entities* der *Import_ ID* eindeutig zugeordnet wurde (vgl. Abschnitt "die Stammabfragen"), so dass hier eine weitere Nennung überflüssig ist. Besonders auffallend am Beispiel ist das Fehlen der *XML_ Field_ ID*-Ausprägung für das *AIDA_ Field_ ID* "harmBy". Die Vermutung liegt hier nahe, dass das "harmBy" Feld aus AIDA PC sich nicht einem IDML Element oder Attribut zuordnen lässt.

IDML_ Lookup_ Join:

Query_ ID	Join_ ID	Fld_ ID
idml_imp_abstract	2	translatedBy
idml_imp_abstract	5	lang
idml_imp_abstract	14	ID
idml_imp_abstract	1	infoStatus
idml_imp_activity	14	ID
idml_imp_activity	3	origin
idml_imp_anotherView	14	ID
idml_imp_anotherView	1	infoStatus
idml_imp_duration	1	infoStatus
idml_imp_duration	14	ID

Abbildung 3.3: Ausschnitt aus der Tabelle IDML_ Lookup_ Join

3.3.2 Prozeduren

Der Überblick in Abbildung 3.5 stellt die wichtigsten Prozeduren zusammen, die für den Datenimport verwendet wurden.

Import_ Mapping:

Import_ ID	XML_ Tab_ ID	XML_ Field_ ID	AIDA_ Field_ ID
idml_imp_activity	idml_imp_activity	date	last_update
idml_imp_activity	idml_imp_activity	origin	origin_activity_dbKey
idml_imp_activity	tmp_imp_activityID	ID	dbKey
idml_imp_id	idml_imp_id	assigningOrg	assigningOrg
idml_imp_id	idml_imp_id		harmBy
idml_imp_id	idml_imp_id	uniqID	uniqID
idml_imp_id	infoStatusLookup	IDML_infoStatus_refKey	infoStatus_dbKey

Abbildung 3.4: Ausschnitt aus der Tabelle Import_ Mapping

Nr.	Prozedur	Beschreibung
1	Import_XML	Startet den Importvorgang
2	loadXMLFile	Parsen, DOM-Baum erstellen und Daten in die Tabellen einlesen
3	Synchronization	Führt Checkroutinen zur Festlegung der Importoptionen pro Projekt durch (Add, Replace, No Import) Formular "Synchronisation" wird geladen.
4	DataImport	Die Transformation wird in Abhängigkeit der Importoption vorbereitet.
5	Actualizing_AIDA	Transformationsprozess durchführen.
6	WriteLogfile	Erstellung des Logfile für den erfolgten Datenimport.

Abbildung 3.5: Überblick der Prozeduren/Funktionen für den Datenimport

4 Schlussfolgerungen

Anhand des Beispiels von AIDA PC haben wir gesehen, dass sich der flexible Einsatz von DOM für den Datenaustausch zwischen XML-Dokumenten und relationalen Datenbanken gut bewährt. Insbesondere ist sein Einsatz dann sinnvoll, wenn ganze Baumstrukturen in relationale Datenbanken abgebildet werden sollen oder Schreib- und Lesezugriff gefordert werden.

Im Gegensatz zu den vollautomatischen Mapping Ansätzen, bei denen eine kontrollierte Steuerung erschwert ist, bietet der 2-Phasen Mapping Ansatz beim Datenaustausch grosse Gestaltungsfreiheiten. So können in der ersten Phase beliebige "well-formed" XML-Schemas in der Datenbank gespeichert und in der nächsten Phase die Daten prozedural nach Belieben transformiert werden.

Die Steuerung des Importprozesses wird mit steigender Dimensionenzahl des XML-Dokumentes anspruchsvoller. Besondere Anforderungen werden hier bei der Zuweisung von Fremdschlüsselwerten gestellt, die während des Datenimportvorganges automatisch generiert werden.

Die Synchronisationsfunktion mit ihren Optionen "Add", "Not Import" und "Replace" stellt eine grosse Herausforderung dar. Insbesondere die "Replace"-Funktionalität konnte aus diesem Grund nur als unausgereifte Version implementiert werden.

Abbildungsverzeichnis

2.1	Beispiel eines IDML Dokumentes	7
2.2	Grafische Übersicht vom IDML Schema	9
2.3	Konzept für die DOM Implementierung von MSXML (Microsoft XML) .	11
2.4	Raw tree Struktur	20
3.1	Konzept für den Import von IDML Dokumenten in AIDA PC	23
3.2	Ausschnitt aus der Tabelle Import_ Entities: Auswahlabfragen 1. Grades	27
3.3	Ausschnitt aus der Tabelle IDML_ Lookup_ Join	28
3.4	Ausschnitt aus der Tabelle Import_ Mapping	29
3.5	Überblick der Prozeduren/Funktionen für den Datenimport	29

Literaturverzeichnis

[AIDA PC 02] Huesemann, S., Patocchi, F. (2002): *AiDA PC*

abrufbar unter: http://iiufpc06.unifr.ch/huesemann/diss/Aida_PC

Zugriffsdatum: 11.03.2002

[BELLANET 02] Bellanet (2002): *AiDA Accessible Information on Development Activities*

abrufbar unter: <http://www.developmentgateway.org/aida>

Zugriffsdatum: 06.07.2002

[DAYEN 02] Dayen, I. (2002): *Storing XML in Relational Databases*

abrufbar unter: <http://xml.com/lpt/a/2001/06/20/databases.html>

Zugriffsdatum: 15.04.2002

[DOMACT 02] W3C (2002): *Document Object Model Activity Statement*

abrufbar unter: <http://www.w3.org/DOM/Activity>

Zugriffsdatum: 03.10.2002

[DOMFAQ 02] W3C (2002): *Document Object Model FAQ*

abrufbar unter: <http://www.w3.org/DOM/faq>

Zugriffsdatum: 11.03.2002

[DOMKONZ 02] Microsoft (2002): *DOM and MSXML*

abrufbar unter: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/htm/dom_concepts_2lkd.asp

Zugriffsdatum: 04.07.2002

- [DOMOBJ 02] Microsoft (2002): *XML DOM Objects/Interfaces*
abrufbar unter: http://msdn.microsoft.com/library/en-us/xmlsdk/htm/xml_obj_overview_20ab.asp
Zugriffsdatum: 04.07.2002
- [DOMREF 02] Microsoft (2002): *DOM Reference*
abrufbar unter: http://msdn.microsoft.com/library/en-us/xmlsdk/htm/xml_mth_ac_44c3.asp
Zugriffsdatum: 11.03.2002
- [FLOR 99] Florescu, D., Kossmann D. (1999): *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database*
abrufbar unter: <ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-3680.pdf>
Zugriffsdatum: 13.08.2002
- [HUESE 02] Huesemann, S. (2002): *Information Exchange Between Humanitarian Organizations: Using the XML Schema IDML*. Journal of the Association for Information Systems, Vol. 3 (2002), S. 1-26.
- [IBM 01] Purcell, T. (2001): *Meet the Experts: Terry Purcell on Outer Joins*
abrufbar unter: <http://www7b.boulder.ibm.com/dmdd/library/techarticle/purcell/0112purcell.html>
Zugriffsdatum: 10.08.2002
- [IS 01] Huesemann, S. (2001): *Einführung in XML*, Vorlesungsunterlagen der Vorlesung Electronic Business und Informationsmanagement an der Universität Fribourg, 2001
abrufbar unter: <http://www2-iiuf.unifr.ch/is>
Zugriffsdatum: 15.04.2002
- [MSXML40 02] Microsoft (2002): *Microsoft XML Core Services 4.0 SP1*
abrufbar unter: <http://msdn.microsoft.com/downloads/default.asp?url=>

/downloads/sample.asp?url=/msdn-files/027/001/766/msdncompositedoc.xml

Zugriffsdatum: 12.04.2002

[SCHAE 02] Schädler, T. (2002): *Datenaustausch mit XML - Entwicklung eines generischen Vermittlungs-Werkzeuges zwischen XML Schemas und Datenbanken*, Diplomarbeit, Universität Fribourg, Schweiz, 2002

abrufbar unter: <http://www.e-freak.ch>

Zugriffsdatum: 20.10.2002

[SLASH 02] *With XML, is the Time Right for Hierarchical DBs?*

abrufbar unter: <http://slashdot.org/askslashdot/01/11/15/1943223.shtml>

Zugriffsdatum: 10.07.2002

[W3C 00] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E. (2000): *Extensible Markup Language (XML) 1.0 (Second Edition)*

abrufbar unter: <http://www.w3.org/TR/2000/REC-xml-20001006>

Zugriffsdatum: 11.03.2002

[W3SCHOOL 02] *Introduction to DTD*

abrufbar unter: http://www.w3schools.com/dtd/dtd_intro.asp

Zugriffsdatum: 10.07.2002

[XMLHND 00] Goldfarb, C. F., Prescod P. (2000): *Das XML-Handbuch* Addison-Wesley, München