Johannes Kepler Universität Linz

Institut für Bioinformatik
Abteilung für Informationssysteme

Entwicklung eines Prototyps zur Integration von relationalen Datenbanken und XML Schema

Diplomarbeit zur Erlangung des akademischen Grades:
Magister rerum socialium oeconomicarumque (Mag. rer. soc. oec.)

in der Studienrichtung Wirtschaftsinformatik
eingereicht bei a. Univ.-Prof. Mag. Dr. Werner Retschitzegger

Christian Ortner
9956842 / 175

Linz, 24. März 2005

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Linz, 24.03.2005

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich während meines Studiums begleitet und mir geholfen haben, mein Studium letztendlich erfolgreich abzuschließen.
Zu Beginn möchte ich mich recht herzlich bei Elisabeth Kapsammer und Werner Retschitzegger bedanken, die mich tatkräftig unterstützt und mir stets über die eine oder andere Hürde geholfen haben.
Vielen Dank auch an meine Studienkollegen und meine Freunde.
Ganz besonders bedanken möchte ich mich bei meiner Familie, die immer für mich da waren und mir es dadurch ermöglichten, diesen wichtigen Abschnitt meines Lebens erfolgreich zu meistern.

Kurzfassung

Eine Vielzahl von heute gängigen Applikationen verwenden relationale Datenbanksysteme (RDBS), um Daten persistent speichern, verarbeiten und verwalten zu können. Immer häufiger kommt beim Datenaustausch, sowohl zwischen Applikationen als auch zwischen Benutzern und Applikationen, XML (eXtensible Markup Language) zum Einsatz. XML ist eine erweiterbare Auszeichnungssprache, die unter anderem ein Quasi-Standard für den elektronischen Datenaustausch ist. Mit dem zunehmenden Volumen an ausgetauschten Daten in Form von XML wird der Wunsch nach einer an die veränderten Bedürfnisse angepassten Speichermöglichkeit immer deutlicher. Um die Vorteile beider Technologien nutzen zu können, ist die Integration von XML und RDBS nötig. Eine Möglichkeit diese beiden Technologien zu integrieren besteht darin, Schemata von XML-Dokumenten auf Schemata von relationalen Datenbanken (RDB) abzubilden. Auf dieser Möglichkeit basiert auch X-Ray, ein generischer Ansatz zur Integration von XML und relationalen Datenbanken auf Basis eines Metaschemas, das selbst in einem RDBS gespeichert ist. Im Rahmen des erweiterten Ansatzes X-Rayxs wurde ein Metaschema entwickelt, mit welchem relationale Schemata auf Schemata abgebildet werden können, die mit der Sprache XML Schema definiert wurden.

Ziel dieser Diplomarbeit war es, auf Basis dieses Metaschemas, welches das notwendige Abbildungswissen enthält, Algorithmen zu entwickeln, mit denen Daten aus XML-Dokumenten in ein oder mehrerer RDBS eingefügt bzw. Daten aus RDBS in XML-Dokumente ausgelesen werden können, sowie einen Prototyp von X-Rayxs zu implementieren.

Abstract

Nowadays many applications use relational database systems (RDBS) to process, manage and persistently store data. At the same time, XML (eXtensible Markup Language) is used more and more often for datatransfer between applications as well as for datatransfer between users and applications. XML is an extensible markup language that is rapidly emerging as the de-facto standard for digital datatransfer. In order to benefit from both technologies, an integration of XML and RDBS is necessary. One approach to achieve an integration is to map the schemata of XML documents to schemata of relational databases. This approach is the base of X-Ray, a generic approach to map XML to relational databases based on a metaschema, stored in a RDBS.

By developing a metaschema for mapping schemata, defined in XML Schema, to schemata of RDBS, the X-Ray approach has been extended to X-Rayxs. The main objective of this diploma thesis was to develop algorithms to import data from XML documents into one or more RDBS and to export data from an RDBS to an XML document, based on the newly developed metaschema. Furthermore, a prototype of X-Rayxs had to be implemented.

Inhaltsverzeichnis

1. Einleitung 9

1.1. Motivation und Zielsetzung 9

1.2. Aufbau der Diplomarbeit 10

2. Vergleich zwischen DTD und XML Schema 12

2.1. Schemasprachen 12

2.1.1. Document Typ Definition (DTD) 13

2.1.2. XML Schema (XSD) 14

2.2. Die wesentlichen Vorteile von XML Schema 16

3. X-Ray 18

3.1. Designziele von X-Ray 19

3.1.1. Art der Schemata 19

3.1.2. Darstellung der Abbildungsinformationen 19

3.1.3. Bindung an Schemata 19

3.1.4. Abbildungskardinalität 20

3.1.5. Zugriffsmöglichkeit 20

3.1.6. Zugriffssprache 20

3.1.7. Zugriffsziel 20

3.2. Arbeitsphasen von X-Ray 20

3.2.1. Initialisierungsphase 21

3.2.2. Laufzeitphase 21

3.3. Architektur von X-Ray 21

3.4. Abbildungsmuster zwischen XML-Schema und DB-Schema 23

3.4.1. Grundlegende Abbildungsarten 23

3.4.2. Sinnvolle Abbildungsstrategien 26

3.5. Metaschema im Detail 30

3.5.1. DB-Schema Komponente 31

3.5.2. XMLDTD-Komponente 31

3.5.3. XMLDBSchemaMapping-Komponente 33

4. Funktionalität des Prototyps 34
4.1. Unterstützte XML Schema Konzepte 34
4.2. Nicht unterstützte XML Schema Konzepte 45
4.3. Anwendungsfälle 47
4.4. Benutzerschnittstelle 53

5. Architektur 56
5.1. Interne Pakete 57
5.2. Externe Pakete 59

6. Laufzeitphase des Prototyps X-Rayxs 60
6.1. Login 62
6.2. Laden des Metaschemas 63
6.3. Export 67
6.4. Export-Algorithmus 69
6.4.1. ET_0 72
6.4.2. $ET\_R_{direkt/indirekt}$ 72
6.4.3. ET_A_{direkt} 73
6.4.4. $ET\_A_{indirekt}$ 74
6.4.5. $A\_A_{direkt/indirekt}$ 77
6.4.6. Elemente mit abstrakten komplexen Datentypen 77
6.5. Abfragen mittels XQuery 79
6.6. Import 83
6.7. Import Algorithmus 85

7. Ausblick 89

Abbildungsverzeichnis 91
Tabellenverzeichnis 93
Quellenverzeichnis 94
Anhang A: Beispiel XML-Dokument mit Schemadateien 99
Anhang B: Relationen des Metaschemas mit Beispieldaten 106
Anhang C: UML Diagramme 119
Anhang D: Installationsanleitung / Benutzerhandbuch 123

1. Einleitung

1.1. Motivation und Zielsetzung

Die Realisierung des reibungslosen und schnellen Datenaustauschs zwischen oft nur lose gekoppelten Systemen ist eine wesentliche Herausforderung bei der Entwicklung von Applikationen. In diesem Zusammenhang gewinnt XML als Format für den Datenaustausch immer mehr an Bedeutung. Gleichzeitig werden große Datenmengen in zumeist relationalen Datenbanken gehalten, deren Struktur nicht dazu geeignet ist, um Daten in Form von XML zu speichern. Eine Schlüsselaufgabe besteht deshalb in der Entwicklung von Konzepten und Techniken, mit deren Hilfe Daten aus herkömmlichen relationalen Datenbanken in XML abgebildet und umgekehrt, XML-Daten in relationalen Datenbanken abgelegt werden können.

Es gibt zwar einige kommerzielle Anbieter von XML-Datenbanken und RDBS, jedoch existieren nur wenige Applikationen, die es, unter Gewährleistung der Autonomien der Schemata auf Seite von XML sowie auf Seiten des RDBS, ermöglichen, XML-Daten in ein RDBS unter Verwendung eines von Benutzern erstellten und erweiterbaren Abbildungswissens zu integrieren [KAPP04].

An der Abteilung für Informationssysteme der Johannes-Kepler-Universität wurde das X-Ray Konzept entwickelt, dass die Basis für diese Arbeit darstellt. X-Ray ist ein generischer Ansatz zur Integration von XML und relationalen Datenbanken auf Basis eines Metaschemas zum Speichern des Abbildungswissens. Das Metaschema selbst ist ebenfalls in einem RDBS gespeichert. Bei X-Ray ist es jedoch nur möglich, XML-Dokumente, die mittels DTDs (Document Type Definition) beschrieben werden, auf ein RDBS abzubilden. Ziel der Diplomarbeit [KRAU05] ist es, X-Ray dahingehend zu erweitern, dass auch XML-Dokumente, die mit XML Schema [W3C05] beschrieben werden, verwendet werden können. Der Grund dafür ist, dass XML Schema, im Gegensatz zu DTDs, immer häufiger verwendet wird, um Strukturen von XML-Dokumenten zu beschreiben. Diese erweiterte Variante von X-Ray trägt bezeichnenderweise den Namen X-Rayxs.

Der Terminus *XML Schema* wird im Zuge dieser Arbeit für die gleichnamige Schemasprache des W3C [W3C05] verwendet. *XML-Schema* hingegen wird als Bezeichnung für die Schemadatei eines XML-Dokumentes, egal ob auf DTDs oder XML Schema basierend, gebraucht.
Während das Ziel der Diplomarbeit [KRAU05] ist, ein Metaschema für X-Rayxs zu entwickeln, ist das Ziel dieser Diplomarbeit entsprechende Algorithmen für eine prototypische Umsetzung des X-Rayxs Konzepts zu entwickeln. Da der Prototyp ausschließlich die Laufzeitphase, also den Import, Export und Abfragen mit XQuery, von X-Rayxs umfaßt, wurden drei Demonstrationsbeispiele hinzugefügt. Das Befüllen, des Metaschemas mit diesen Beispieldaten ersetzt die Initialisierungsphase, die für die Laufzeitphase Voraussetzung ist. Somit ist ein Prototyp entstanden, der die Validität des X-Rayxs-Konzepts nachweist.

1.2. Aufbau der Diplomarbeit

Zu Beginn der Diplomarbeit werden die beiden Möglichkeiten, die Struktur eines XML-Dokumentes zu beschreiben, gegenüber gestellt. Dabei wird zuerst DTD und im Anschluss XML Schema vorgestellt. Am Schluss des Kapitels werden die Vorteile von XML Schema gegenüber DTD dezidiert herausgehoben.

Im anschließenden Kapitel 3 wird X-Ray, das die konzeptionelle Basis für X-Rayxs bildet, vorgestellt. Nach den Designzielen und den beiden Phasen von X-Ray wird die Architektur dieses Konzeptes beleuchtet. Zum besseren Verständnis werden die Abbildungsmuster, welche beschreiben, wie Elemente von XML-Dokumenten abhängig von ihrer Ausprägung, auf ein RDBS abgebildet werden können, erläutert. Der Abschluss des Kapitels widmet sich dem Metaschema, in welchem das Abbildungswissen gespeichert wird. Hier finden auch die zuvor vorgestellten Abbildungsmuster ihre praktische Anwendung.

Die Kapitel vier bis sechs beleuchten den im Zuge dieser Diplomarbeit entwickelten Prototyp X-Rayxs. Der Prototyp ist eine Umsetzung des Metaschemas von X-Rayxs, das im Detail in der Diplomarbeit [KRAU05] beschrieben ist, und darauf aufbauender Algorithmen für das Importieren, Exportieren und Abfragen mit X-Query.

In Kapitel 4 wird die Funktionalität des Prototyps betrachtet. Dazu werden die in X-Rayxs berücksichtigten bzw. nicht berücksichtigten XML Schema Konzepte erläutert. Die Anwendungsfälle für den Prototyp sollen dem Leser die möglichen Ablaufszenarien näher bringen. Abschließend wird in diesem Kapitel ein Überblick der Benutzerschnittstellen des Prototyps gegeben.

Kapitel 5 behandelt die Architektur des Prototyps. Die Java-Klassen des Prototyps werden dort gemäß ihrer Funktionalitäten und Aufgaben in Pakete zusammengefasst und erläutert.

Im Kapitel 6 wird die Laufzeitphase des Prototyps im Detail betrachtet. Es wird der Export von relationalen Daten in XML-Dokumente, das Abfragen von relationalen Daten mittels XQuery und der Import von XML-Dokumenten in das RDBS erläutert. Dabei werden sowohl die internen Abläufe, als auch die Sicht des Benutzers, berücksichtigt. Beispiele unterstützen dabei die Ausführungen.

2. Vergleich zwischen DTD und XML Schema

Die eXtensible Markup Language (XML) wurde 1998 in erster Version vom World Wide Web Consortium (W3C) als Empfehlung verabschiedet [W3C05]. XML ist eine Metasprache, die es ermöglicht, anhand von Auszeichnungen, so genannten Tags, neue Markup-Sprachen zu definieren. Auf diese Weise definierte Sprachen werden XML-Anwendungen genannt [SCHN04].

2.1. Schemasprachen

„Der Zweck eines Schemas ist es, eine Klasse von XML-Dokumenten zu definieren. Deshalb wird häufig der Begriff “Instanzdokument” oder kurz “Instanz” verwendet, um ein Dokument zu beschreiben, das einem bestimmten Schema entspricht.“ [W3C04C]

Die ursprüngliche XML 1.0 Empfehlung definierte nur eine Art der Beschreibung von Inhaltsmodellen eines XML Instanzdokumentes: die Document Type Definition (DTD) [W3C04A]. Eine DTD wird verwendet, um in einem XML-Dokument zulässige Elemente und Attribute zu beschreiben und ihre Beziehungen untereinander auszudrücken. DTD’s waren somit die erste Möglichkeit, um XML-Anwendungen zu beschreiben.

Bald merkte man allerdings, dass man zum Beschreiben von XML-Anwendungen ein flexibleres Modell benötigte, das u. a. Datentypen berücksichtigt. 1999 begann das W3C an der neuen XML Schema Definitions-Sprache (XSD) zu arbeiten, die heute als endgültige Empfehlung vom 2. Mai 2001 (2. Auflage 28. Okt. 2004) erhältlich ist [W3C04B]. XML Schema baut auf den Erfahrungen mit DTD’s und älteren Sprachen auf, und ist die derzeit flexibelste und leistungsfähigste Möglichkeit um XML-Anwendungen zu beschreiben.

Im Folgenden werden zunächst sowohl DTD’s als auch XML Schema kurz beschreiben. Anschließend daran werden die wichtigsten Vorteile von XML Schema im Vergleich zu DTD’s diskutiert.

2.1.1. Document Typ Definition (DTD)

DTDs dienen zur Definition der erlaubten Elemente, Attribute und ihrer Zusammensetzung für Instanzen, den XML-Dokumenten. Die DTD legt damit die in der Instanz zu verwendenden Tags fest und bestimmt, in welcher Reihenfolge sie auftreten und wie sie verschachtelt werden dürfen. Außerdem werden die bei einem Element erlaubten Attribute und die Art der Werte festgelegt [GALI04].
Liegt eine DTD vor, kann ein konkretes XML-Dokument an Hand der zugeordneten DTD auf Gültigkeit geprüft werden. Die Verwendung von DTDs ist aber nicht vorgeschrieben. Fehlt eine DTD, kann ein XML-Dokument nur auf Wohlgeformtheit geprüft werden. Im Folgenden werden die Stärken und Schwächen von DTD's aufgelistet und es wird ein Beispiel für eine DTD zur Verwaltung von Unterkünften gezeigt (vgl. Abbildung 1).

- *Stärken von DTDs:*
  - Kompakte Syntax
  - Schnelle Verarbeitung möglich
  - Handliche Größe
- *Schwächen von DTDs:*
  - DTD's verwenden keine XML-Syntax
  - DTD's unterstützen keine Namensräume
  - Datentypen sind begrenzt
  - Keine Vererbung möglich
  - Eingeschränktes Identifizierungs- und Referenzierungskonzept

```
 …
<!ELEMENT accommodations (accommodation*)>
<!ELEMENT accommodation (name, address, email*, phone+,
     acceptsCreditCard?, facilities, sauna, pool*, description?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (street, village, country)>
 …

<!ATTLIST accommodation id      CDATA      #REQUIRED
     state                CDATA      #FIXED "Austria"
     kind                 (hotel | motel) "hotel">
     …
```

Abbildung 1 - Beispiel für DTD

2.1.2. XML Schema (XSD)

Ein XML Schema beschreibt selbstdefinierte Datentypen und die Struktur von XML Dokumenten. Als Basis-Datentypen stehen die in “XML Schema Part 2: Datatypes Second Edition” [W3C04D] definierten Typen zur Verfügung. Hierzu zählen z. B. Integer – Zahlen ohne Nachkommastellen, Float – Zahlen mit Nachkommastellen und Strings – Zeichenketten mit fester oder variabler Länge etc. (siehe Abbildung 21). XML Schema erlaubt darüber hinaus auch die Definition eigener Datentypen. In der Strukturdefinition wird festgelegt in welcher Art und Weise (Reihenfolge, Anzahl, etc.) die Elemente und Attribute in den Instanzen auftreten dürfen. Somit ist es möglich z. B. einen Datentyp „Hotel“ zu definieren, der wiederum aus mehreren Zimmern besteht, die selbst wieder durch einen eigenen Datentyp definiert werden, und jeweils eine bestimmte Anzahl an Betten usw. aufweisen. Im Gegensatz zu DTD’s entsprechen die Möglichkeiten von XML Schema den heutigen Anforderungen von Datenbanken und Programmiersprachen. Die Bestandteile des Schemas werden wiederum in XML-Syntax beschrieben. [GALI04]
Liegt ein Schema vor, kann ein konkretes XML-Dokument mit Hilfe des zugeordneten XML Schemas auf Gültigkeit geprüft werden. Die Verwendung von XML Schemata ist aber, geanuso wie die Verwendung von DTD’s, nicht vorgeschrieben. Fehlt die Zuordnung zu einem XML Schema, kann ein XML-Dokument nur auf Wohlgeformtheit geprüft werden.
Im Anschluss werden die Stärken und Schwächen von XML Schema aufgelistet und es wird ein Beispiel für ein XML Schema zur Verwaltung von Unterkünften gezeigt (vgl. Abbildung 2).

- *Stärken von XML Schema*
  - Schemadefinitionen sind XML-Dokumente
  - Unterstützung von Namensräumen
  - Viele vordefinierte Datentypen (Datum, Währung, Integer, etc.)
  - Definition eigener Datentypen
  - Vererbung von Typdefinitionen
  - Flexibles Identifizierungs- und Referenzierungskonzept

- *Schwächen von XML Schema*
  - Syntax von XML Schema sehr umfangreich
  - XML Schema Dokumente sehr umfangreich

```
<schema targetNamespace="AccommodationSchema"
elementFormDefault="qualified" xmlns:ac="AccommodationSchema"
xmlns:sp="xray_supervision.xsd"
xmlns="http://www.w3.org/2001/XMLSchema">
...
<complexType name="accommodationType">
   <sequence>
      <element ref="ac:name"/>
      <element name="address" type="ac:addressType"/>
      <element name="manager" type="ac:managerType"/>
      <element name="supervisor" type="su:supervisionType"/>
      <element name="test" type="ac:mangerType"/>
      <element name="email" type="string" minOccurs="0"
           maxOccurs="unbounded"/>
      <element name="phone" type="ac:phoneType"
           maxOccurs="unbounded"/>
      <element name="acceptsCreditCards" minOccurs="0">
           <complexType/>
      </element>
      <element name="facilities">
           <complexType/>
      </element>
      <element name="sauna" type="ac:saunaType"/>
      <element name="pool" minOccurs="0" maxOccurs="unbounded">
           <complexType/>
      </element>
      <element name="description" type="ac:descriptionType"
           minOccurs="0"/>
      <element name="room" type="ac:roomType"
           maxOccurs="unbounded"/>
   </sequence>
   <attribute name="id" type="positiveInteger" use="required"/>
   <attribute name="state" type="string" fixed="Austria"/>
</complexType>
...
```

Abbildung 2 - Beispiel für XML Schema (XSD)

2.2. Die wesentlichen Vorteile von XML Schema

Unterstützung von Namensräumen

Namensräume dienen zur eindeutigen Identifizierung von Typdefinitionen und Deklarationen von Elementen in einem Schema. Jeder Namensraum ist durch einen URI eindeutig beschrieben. Alle Namen in einem Schema gehören zu einem bestimmten Namensraum. Sie erlauben uns also zwischen verschieden Vokabularen zu unterscheiden [W3C04C]. Dies geschieht mit Hilfe von Definitionen verschiedener Namensräume und deren Zuweisung zu jeweils einem Präfix im <*schema*>-Element eines XML Schemas (siehe Abbildung 2).

Beispielsweise kann nun zwischen einem <*Titel*>-Element, das zum Typ „Buch“ gehört, und einem gleichnamigen Element, das allerdings dem Typ „Person“ zuggehörig ist, unterschieden werden.

Definition eigener Datentypen

XML Schema verfügt bereits über eine Vielzahl von vordefinierten Typen wie z.B. String, Integer, Float, Boolean, etc. Auf diesen Basistypen aufbauend können weitere, eigene Typen definiert werden, wobei zwischen einfachen und komplexen Typen unterschieden wird.

Ein einfacher Typ beinhaltet weder Attribute noch Kindelemente. Ein Element dieses Typs wird als atomares Element bezeichnet. Mit Hilfe des <*restriction*>-, <*list*>- und <*union*>-Elements und den so genannten Facetten ist es jedoch möglich, einfache Typen auf die eigenen Bedürfnisse abzustimmen.

Wesentlich mehr Möglichkeiten eigene Datentypen zu definieren, bieten die komplexen Datentypen. Mit ihnen ist es möglich Elemente und Attribute in nahezu jeder beliebigen Struktur zu kombinieren.

Vererbung

Ähnlich wie in objektorientierten Programmiersprachen ist es bei XML Schema möglich, Typen von anderen Typen abzuleiten. XML Schema verfügt dabei über einen Ur-Typ, oder auch Basistyp genannt (siehe Abbildung 21), von dem alle einfachen und komplexen Typen abgeleitet werden. Eine Ausnahme ergibt sich durch die Angabe eines anderen Basistypen durch das *<… source …>*-Attribut.
Dabei ist es möglich bestehende Typdefinitionen zu erweitern oder einzuschränken. XML Schema unterstütz darüber hinaus das Konzept der abstrakten Typen, welche in Instanzen nicht direkt genutzt werden können. Abstrakte Typen stehen nur anderen Typdefinitionen zur Verfügung, um sie zu erweitern oder einzuschränken. Umgekehrt ist es möglich, Typdefinition als „final“ zu deklarieren, um zu verhindern, dass weitere Typen von diesem Typ abgeleitet werden.

3. X-Ray

In diesem Abschnitt der Diplomarbeit soll das Konzept und die Architektur von X-Ray erklärt werden. Sämtliche Informationen und Daten in diesem Kapitel der Diplomarbeit wurden [KAPP04] entnommen.
Es gibt mehrere Ansätze und Realisierungen zum Speichern und Verwalten von Daten aus XML-Dokumenten in relationale Datenbanken. Jedoch haben diese Ansätze allesamt unterschiedliche Schwerpunkte. Ein Überblick hierzu wird in [KAPP04] gegeben.
X-Ray wurde entwickelt, um Daten aus XML-Dokumenten mit Hilfe eines Metaschemas in Relationen einer oder mehrerer relationalen Datenbanken zu speichern und auch wieder auslesen zu können, um die gespeicherten Daten in wohlgeformte und gültige XML-Dokumente zu exportieren.
Ein wichtiges Ziel von X-Ray ist die Flexibilität bezüglich der abzubildenden Schemata. Hierfür wurde der so genannte „benutzerdefinierte“ Ansatz gewählt, welcher die Autonomie der Schemata der XML-Dateien (in Form von DTDs) sowie die Schemata des RDBS (Relationales Datenbanksystem) gewährleistet. Die Schemainformationen der XML-Dokumente sowie jene der Relationen des RDBS werden unabhängig voneinander gespeichert und verwaltet. Die Daten, welche die Informationen zur Abbildung des Schemas der XML-Dateien auf das relationale Schema enthalten, werden im sogenannten „Mappingschema“ gespeichert.
Das Schema eines XML-Dokumentes wird im Folgenden als XML-Schema bezeichnet, unabhängig davon, ob es sich bei der Schemasprache um DTDs oder um XML Schema [W3C05] handelt.

3.1. Designziele von X-Ray

3.1.1. Art der Schemata

Bei X-Ray wird wie erwähnt vom „benutzerdefinierten“ Ansatz ausgegangen. Das Gegenteil dazu ist der „abgeleitete“ Ansatz, bei welchem das Datenbankschema vom XML-Schema oder vice versa nach vordefinierten Regeln abgeleitet wird. Beim „benutzerdefinierten“ Ansatz können sowohl Datenbankschema, als auch das Schema eines XML-Dokumentes (in Form einer DTD) unabhängig voneinander entwickelt werden. Dies erlaubt eine höhere Flexibilität. Das Abbilden zwischen den beiden Schemata kann dann jedoch nicht automatisiert nach vordefinierten Regeln ablaufen, sondern wird vom Benutzer von X-Ray durchgeführt.

3.1.2. Darstellung der Abbildungsinformationen

Um Abbildungstransparenz zu erreichen, müssen die Abbildungsinformationen (*mapping knowledge*) in irgendeiner Form gespeichert und verwaltet werden. Das Einbinden der Abbildungsinformationen in den Applikationscode ist nachteilig, da bei Änderungen immer der Programmcode der Applikation geändert werden müsste. Da ein wichtiges Designziel von X-Ray neben einem transparenten Mapping (Abbilden eines Schemas auf ein anderes) eine einfache Wartbarkeit ist, werden sämtliche Metadaten, das sind Informationen über die Schemata und die Abbildung zwischen diesen, in einem Metaschema verwaltet, welches in einem RDBS gespeichert wird.

3.1.3. Bindung an Schemata

Bei X-Ray wird die „lose“ Bindung der Abbildungsinformationen an das XML-Schema bzw. DB-Schema realisiert. Dies bedeutet, dass die Abbildungsinformationen unabhängig von den Schemata gespeichert werden können, was wesentlich zur Flexibilität beiträgt.

3.1.4. Abbildungskardinalität

Ein weiteres wichtiges Designziel von X-Ray ist die Unterstützung von „multiplen Schemata“. Eine Abbildung (*Mapping*) ist genau für ein XML-Schema und ein DB-Schema gültig. Jedoch kann es mehrere Abbildungen zwischen einem XML-Schema und beliebig vielen DB-Schemata sowie vice versa geben, so auch z.B. mehrere Abbildungen eines XML-Schemas auf verschiedene DB-Schemata eines RDBS.

3.1.5. Zugriffsmöglichkeit

X-Ray unterstützt den „*unified approach*“. Dies bedeutet, dass man mit X-Ray nicht nur Daten aus XML-Dokumenten speichern, sondern auch Daten aus einer DB in XML-Format ausgeben kann.

3.1.6. Zugriffssprache

Für den Zugriff auf Daten die in der DB gespeichert sind, gibt es den XML-zentrierten sowie den DB-zentrierten Ansatz für Abfragesprachen. X-Ray unterstützt den XML-zentrierten Ansatz und verwendet daher eine XML-basierte Abfragesprache. (z.B. X-RAYQL oder XQuery)

3.1.7. Zugriffsziel

Beim Zugriff auf X-Ray wird eine virtuelle XML-Sicht (*View*) der in einer DB gespeicherten Daten zur Verfügung gestellt. Der Benutzer benötigt daher z.B. beim Erstellen von Abfragen kein Wissen über das DB-Schema.

3.2. Arbeitsphasen von X-Ray

X-Ray lässt sich bzgl. seiner Verwendung in zwei Hauptphasen unterteilen:

- Initialisierungsphase
- Laufzeitphase

Diese beiden Phasen sind von einander abhängig, d.h. zuerst ist die Initialisierungsphase zu durchlaufen und danach ist die Laufzeitphase von X-Ray verfügbar.

3.2.1. Initialisierungsphase

Mit X-Ray können Daten nur verwaltet werden, wenn die entsprechenden Metadaten der XML-Dokumente, der RDBS sowie die Abbildungsdaten, gespeichert in einem Metaschema, vorhanden sind. In der Initialisierungsphase können Schemainformationen von XML-Dokumenten ebenso wie von RDBS importiert werden. Mittels des im Abschnitt ***Architektur von X-Ray*** beschriebenen *„Mapping Knowledge Editors“* kann die Abbildung eines XML-Schemas auf das Schema einer RDBS (deren Relationen) vorgenommen werden. Die dabei verwendeten Abbildungsinformationen werden als Abbildungsdaten in dem Metaschema gespeichert.

Ein Metaschema speichert nach der Initialisierungsphase die Schemadaten von XML-Dokumenten, die Schemadaten der verwendeten Relationen eines RDBS und letztendlich die zugehörigen, vom Benutzer erstellten, Abbildungsdaten (*Mappingdaten*). Diese Phase sollte nur dem X-Ray-Administrator zugängig sein.

3.2.2. Laufzeitphase

Das Importieren von Daten aus XML-Dokumenten, das Exportieren von Daten in ein XML-Dokument, sowie das Ausführen von Abfragen auf gespeicherte Daten erfolgt in der Laufzeitphase von X-Ray.

In dieser Phase kann ein beliebiges Metaschema aus dem im Abschnitt ***Architektur von X-Ray*** beschriebenen „*Metaschema-Repository*“ geladen und anschließend verwendet werden.

Des Weiteren ist es in dieser Phase möglich, Abfragen auf die gespeicherten Daten zu erstellen und bei Bedarf das Ergebnis zu exportieren. Diese Phase ist dem (der) X-Ray-Benutzer(in) zugänglich und stellt den Standardbetrieb von X-Ray dar.

3.3. Architektur von X-Ray

Die statische X-Ray-Architektur, welche in Abbildung 3 dargestellt ist, setzt sich aus folgenden Komponenten, die nachfolgend kurz beschrieben werden, zusammen.

Metaschema:
Den Kern von X-Ray bildet das zuvor erwähnte Metaschema. In einem Metaschema werden die Schemainformationen von XML-Dokumenten, der Relationen des (der) RDBS, sowie die entsprechenden Abbildungsdaten verwaltet.

Decomposer:
Um Daten aus einem XML-Dokument in die Datenbank zu importieren, wird ein XML-Dokument mittels des Decomposers „zerlegt“ und die Daten mittels der XML-Schemainformationen und der gespeicherten Abbildungsdaten in die entsprechenden Relationen des RDBS gespeichert.

Composer:
Zu exportierende Daten werden mit Hilfe der gespeicherten Abbildungsdaten aus den Relationen ausgelesen und entsprechend den Daten des gespeicherten XML-Schemas im „Composer“ zu einem XML-Dokument zusammengestellt.

Mapping Knowledge Editor:
Diese Komponente von X-Ray dient zum Erstellen von Abbildungsdaten. Mit Hilfe dieser Komponente kann die Abbildung eines XML-Schemas auf ein Schema eines RDBS vorgenommen werden. Diese Komponente kommt in der Initialisierungsphase zum Einsatz.

Domain DB:
Die Daten aus XML-Dokumenten werden persistent in einem RDBS gespeichert. X-Ray ermöglicht es, ein XML-Schema auf verschiedene Schemata eines RDBS abzubilden. Die *DomainDB* repräsentiert hierbei ein solches RDBS. In solch einer *DomainDB* sind die Daten eines oder mehrerer XML-Dokumente gespeichert.

Metaschema-Repository:
Im *Repository* werden sämtliche Daten des Metaschemas persistent gespeichert. Beim Start von X-Ray wird das Metaschema aus dem „*Metaschema-Repository*“ geladen und X-Ray damit initialisiert. Somit steht das Metaschema während der Laufzeit von X-Ray zur Verfügung. Sämtliche Metadaten werden bei X-Ray persistent in einem RDBS gespeichert.

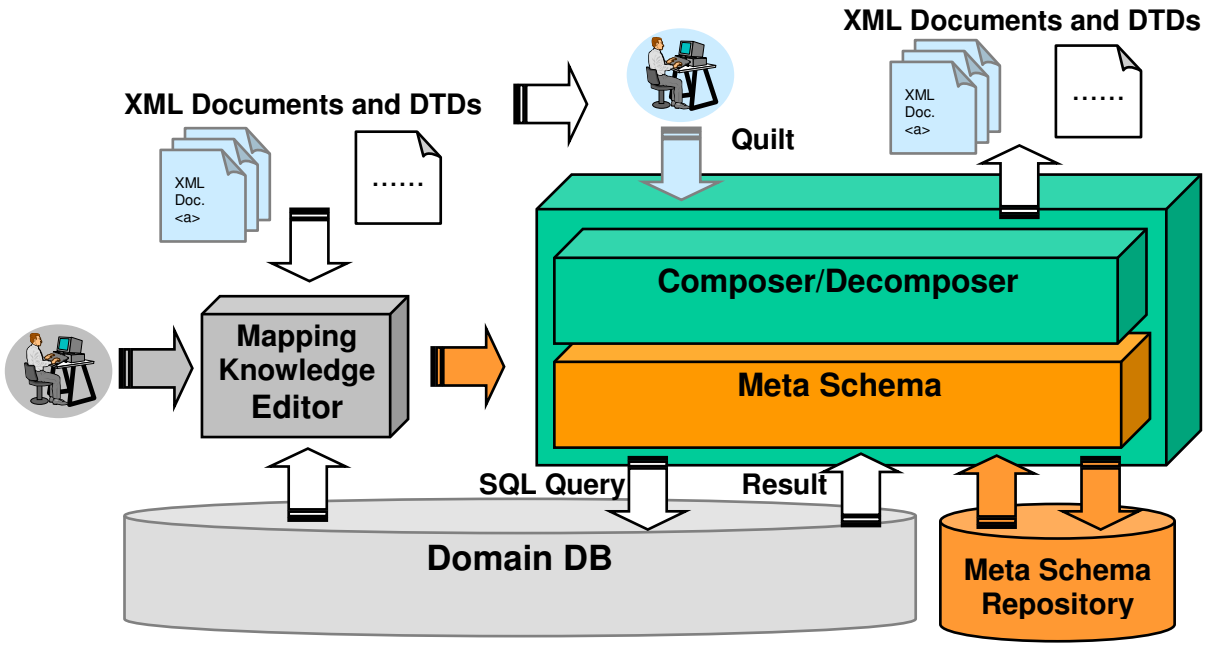


Abbildung 3 - Architektur von X-Ray

3.4. Abbildungsmuster zwischen XML-Schema und DB-Schema

Um ein XML-Schema auf ein DB-Schema abzubilden, wurden allgemeine Abbildungsmuster bzw. grundlegende Abbildungsregeln entwickelt.

In [KAPP04] werden die Unterschiede zwischen den Konzepten von XML sowie RDBS genauer analysiert. Auf Basis dieser Analyse der Konzepte konnten sinnvolle Abbildungsmuster (*mapping pattern*), die wiederum die Basis für das Metaschema darstellen, entwickelt werden.

3.4.1. Grundlegende Abbildungsarten

Der einfachste Weg wäre, die XML-Elemente direkt auf Tupel von Relationen der RDBS abzubilden. Dabei werden Elementtypen auf Relationen und die Attribute der Elemente auf die Attribute der entsprechenden Relation abgebildet. Dies ist jedoch aufgrund der Heterogenität zwischen einem XML-Schema und einem DB-Schema nicht immer möglich bzw. sinnvoll. Zusätzlich ergeben sich erhebliche Nachteile

bezüglich Performanz und Fragmentierung des RDBS bei tief geschachtelten XML-Elementen.

Unter Berücksichtigung der Strukturen der beiden Schemata wurden für X-Ray drei grundlegende Abbildungsarten entwickelt.

Diese drei Abbildungsarten sind:

- ET_R: Ein Elementtyp (ET) wird auf eine Relation der DB abgebildet. Mehrere Elementtypen können auf eine so genannten Basisrelation (*base relation*) abgebildet werden
- ET_A: Ein Elementtyp wird auf ein Attribut einer Relation der DB abgebildet, wobei die Relation des Attributes die Basisrelation des Elementtyps darstellt. Mehrere Elementtypen können auf die Attribute einer Basisrelation abgebildet werden
- A_A: Das Attribut eines Elementtyps wird auf ein Attribut jener Relation abgebildet, welche die Basisrelation des Elementtyps, zu dem das XML-Attribut gehört, repräsentiert.

In Abbildung 4 werden diese drei Abbildungsarten beispielhaft dargestellt.

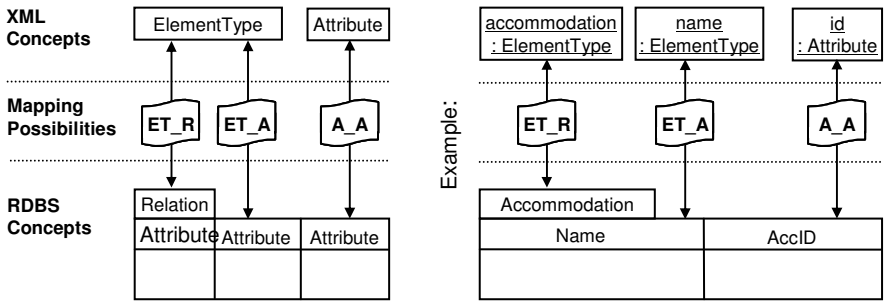


Abbildung 4 - Grundlegende Abbildungsmuster zwischen XML und DB

Es ist sinnvoll, dass ET_R in Verbindung mit ET_A und/oder A_A verwendet wird. Des Weiteren ist es nicht zwingend notwendig, Abbildungsmuster für alle Elementtypen bzw. deren Attribute, sowie für alle Relationen und deren Attribute zu verwenden. Dies ist z.B. bei Surrogatschlüsseln von Relationen der DB, welche lediglich Fremdschlüssel für Beziehungen zwischen einzelnen Relationen darstellen, der Fall. Ein Beispiel auf XML Seite ist ein leerer Elementtyp, der exakt nur einmal vorkommt.

Das Konzept der Basisrelation kann noch verfeinert werden. Man betrachtet das erste der Vorgänger-Elementtypen des abzubildenden Elementtyps, das auf eine

Relation oder ein Attribut abgebildet wird. Dessen Basisrelation bildet nun die Eltern-Basisrelation (*parent base relation*) des abzubildenden Elementtyps und ist zugleich ein Kandidat als dessen Basisrelation. Ist keiner der Vorgänger-Elementtypen abgebildet worden und besitzt daher keine Basisrelation, kann eine beliebige Relation als Basisrelation verwendet werden. Dies wird in Abbildung 5 verdeutlicht.

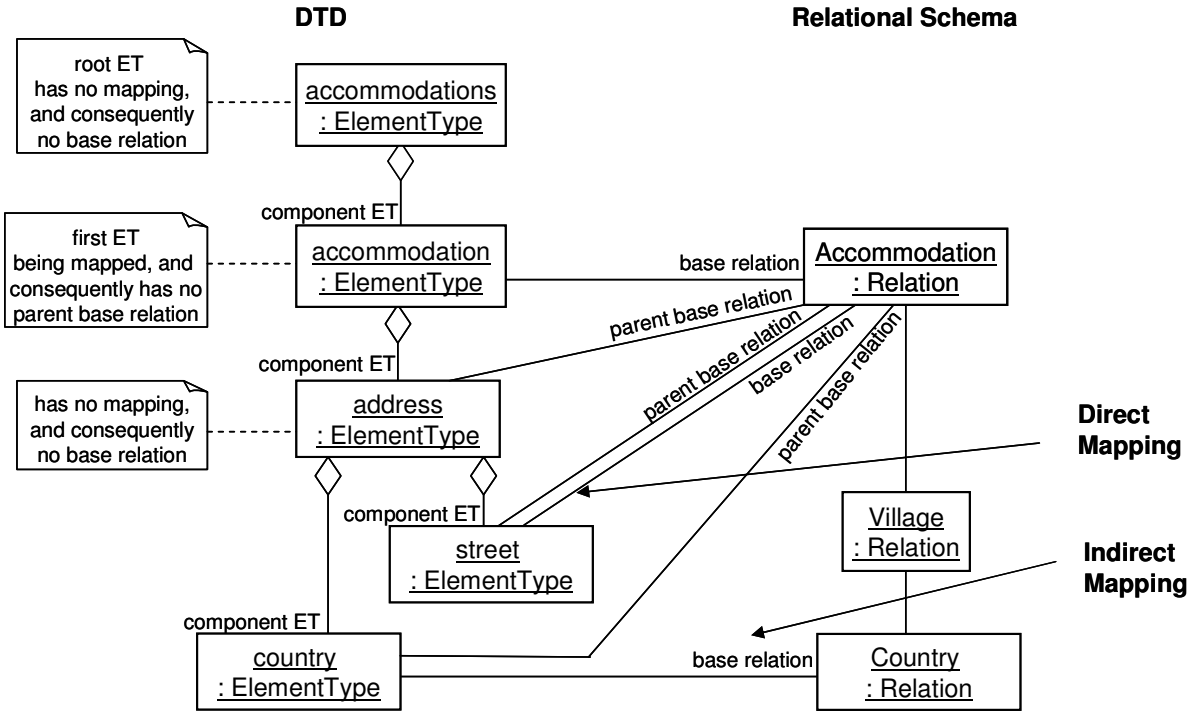


Abbildung 5 - Verfeinerte Abbildungskonzepte

Aus den Konzepten der Basisrelation sowie Eltern-Basisrelation wird das Konzept des direkten und indirekten Abbildens (*direct/indirect mapping*) abgeleitet.

Soll ein XML-Attribut abgebildet werden, so wird zuerst dessen Elementtyp betrachtet. Hat dieser jedoch keine Basisrelation, so wird nach dem zuvor erwähnten Ablauf vorgegangen, um eine Basisrelation und/oder Eltern-Basisrelation zu eruieren. Die Eltern-Basisrelation stellt dann zugleich die Basisrelation dar, wenn ein Elementtyp bzw. ein Attribut des Elementtyps auf diese Relation oder eine ihrer Attribute abgebildet werden kann. In diesem Fall spricht man vom direkten Abbilden (*direct mapping*). In Abbildung 5 wird der Elementtyp „*street*“ direkt auf ein Attribut seiner Eltern-Basisrelation „*Accommodation*“ abgebildet.

Wird hingegen eine geeignete Basisrelation gefunden, die über Fremdschlüsselbeziehungen von der Eltern-Basisrelation aus erreicht werden kann, so spricht man von indirektem Abbilden (*indirect mapping*).

In Abbildung 5 wird dies durch den Elementtyp „*country*“, welcher indirekt auf die Basisrelation „*Country*“ (mit Eltern-Basisrelation „*Accommodation*“) abgebildet wird, dargestellt. Indirektes Abbilden ist sinnvoll, wenn ein Elementtyp oder Attribut aufgrund von Normalisierung der Relationen in eine eigene Relation ausgelagert werden soll.
Somit ergeben sich letztendlich die sechs Abbildungsarten $ET\_R_{direkt/indirekt}$, $ET\_A_{direkt/indirekt}$ und $A\_A_{direkt/indirekt}$.

3.4.2. Sinnvolle Abbildungsstrategien

Bisher wurden bei den Abbildungsstrategien die verschiedenen Arten von XML-Elementtypen sowie XML-Attributtypen noch nicht berücksichtigt. Für das Entwickeln sinnvoller Abbildungsstrategien ist dies jedoch unerlässlich. Die resultierenden Abbildungsmuster sollen einerseits den Abbildungsprozess auf der syntaktischen Ebene vereinfachen und andererseits Abbildungen vermeiden, bei denen es zu syntaktischen Konflikten kommt. Dieser Ansatz unterscheidet sich wesentlich von [KAPP04].
Die bestimmenden Faktoren können wie folgt unterteilt werden:

- Eigenschaften von XML-Elementtypen
- Eigenschaften von XML-Attributtypen

Diese Eigenschaften werden nun näher beleuchtet.

3.4.2.1. Eigenschaften von XML-Elementtypen

Die Eigenschaften von XML-Elementtypen lassen sich in drei Dimensionen unterteilen:

1. Art des Elementtyps
2. Existenz von Attributen
3. Kardinalität

In Abbildung 6 werden diese drei entscheidenden Eigenschaften mitsamt ihren Ausprägungen dargestellt.

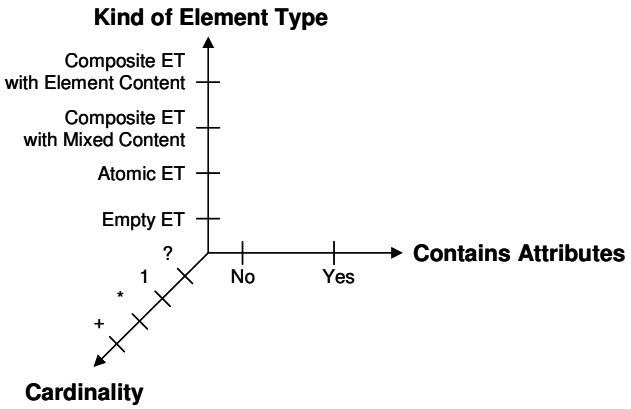


Abbildung 6 - Charakteristische Eigenschaften von XML-Elementtypen

Abhängig von der Kombination der Ausprägung dieser drei Eigenschaften lassen sich sinnvolle Abbildungsstrategien ableiten. Weist jedoch ein Elementtyp das Inhaltskonzept *ANY* auf, ist keine Abbildungsstrategie im Vorhinein bestimmbar. Eine Übersicht über die Abbildungsstrategien wird in Abbildung 7 gegeben.

Kind of element type	Contains attributes	Cardinality	Reasonable mapping
Composite ET with element content	No influence	No influence	ET_$R_{direct/indirect}$; No mapping
Atomic ET	No influence	?, 1	ET_$A_{direct/indirect}$
Atomic ET	No influence	+, *	ET_$A_{indirect}$
Empty ET	No	1	No mapping
Empty ET	Yes	1	ET_$R_{direct/indirect}$; No mapping
Empty ET	No influence	?	ET_A_{direct}
Empty ET	No influence	*, +	ET_$A_{indirect}$
Composite ET with mixed content	No influence	No influence	ET_$A_{indirect}$

Abbildung 7 - Sinnvolle Abbildungsstrategien für XML-Elememttypen

Im Folgenden werden die einzelnen XML-Elementtypen näher betrachtet, um die Abbildungsstrategien verständlicher zu machen.

Zusammengesetzter ET mit Element-Inhalt (Composite ET with element content):
Mit Elementen dieses Typs sind keine zu speichernden Daten verbunden, woraufhin die einzig sinnvolle Abbildungsart ET_R ist. Ob direkt oder indirekt abgebildet wird, ist davon abhängig, ob dieses Element auf die Eltern-Basisrelation abgebildet werden kann oder nicht (siehe Abschnitt ***Grundlegende Abbildungsarten***). Es käme keinem Informationsverlust gleich, wenn es kein Abbildungsmuster für dieses Element gäbe, da es keine Daten enthält, die in der DB gespeichert werden.

Atomarer ET (atomic ET):
Bei Elementen dieses Typs ist ausschließlich die Kardinalität für das sinnvolle Abbilden ausschlaggebend. Da diese Elemente immer einen Wert enthalten, muss dieser Wert in einem Attribut einer Relation gespeichert werden, was logischerweise zur ET_A – Abbildungsart führt. Die Unterscheidung zwischen direktem und indirektem Abbilden ist abhängig von der Kardinalität. Ist diese als „1“ oder „?“ angeben, ist ET_A_{direkt} sinnvoll. Wird das Element jedoch nicht auf ein Attribut der Eltern-Basisrelation abgebildet, bleibt nur $ET\_A_{indirekt}$ als Möglichkeit. $ET\_A_{indirekt}$ ist auch bei Kardinalität „*“ und „+“ die einzig sinnvoll Variante.

Leerer Elementtyp (empty ET):
Hat ein Element dieses Typs die Kardinalität „1“, egal ob es Attribute enthält oder nicht, ist ein Abbilden unnötig, da es ja nur exakt einmal vorkommt und keinen Wert enthält. Enthält ein leerer Elementtyp jedoch Attribute, ist $ET\_R_{direkt/indirekt}$ sinnvoll, da die Basisrelation auch als Basisrelation für die XML-Attribut dienen kann.
Hat ein leerer Elementtyp eine andere Kardinalität als „1“, ist es egal, ob er Attribute enthält, weshalb ET_A die einzig sinnvolle Abbildungsart ist. Ob direkt oder indirekt abgebildet wird, ist nur von der jeweiligen Kardinalität abhängig.

Zusammengesetzter ET mit gemischtem Inhalt (Composite ET with mixed content):
Die Abbildungsart dieses Elementtyps ist gänzlich unabhängig von dessen Kardinalität und ob er Attribute enthält oder nicht.
Da auf der Instanzebene mehrere Daten in einem Element vorkommen können, ist jedoch nur $ET\_A_{indirekt}$ sinnvoll.

3.4.2.2. Eigenschaften von XML-Attributtypen

Die Eigenschaften von XML-Attributen lassen sich in zwei Dimensionen unterteilen:

1. Multiplizität des XML-Attributes
2. Standarddeklaration des XML-Attributes

In Abbildung 8 werden diese zwei entscheidenden Eigenschaften mitsamt ihren Ausprägungen dargestellt.

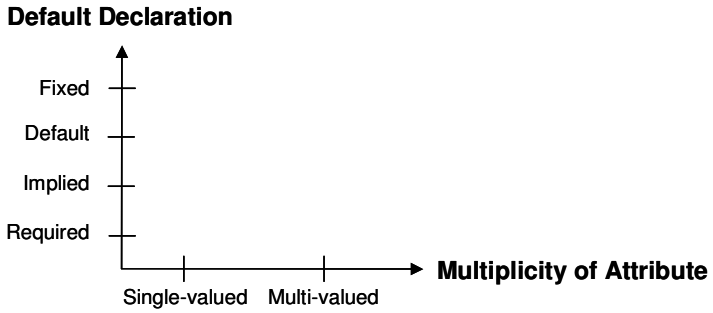


Abbildung 8 - Charakteristische Eigenschaften von XML-Attributen

Abhängig von der Kombination der Ausprägung dieser beiden Eigenschaften lassen sich sinnvolle Abbildungsstrategien ableiten. Eine Übersicht über die Abbildungsstrategien wird in Abbildung 9 gegeben.

Multiplicity of XML attribute	Default declaration	Reasonable mapping
No influence	#FIXED	No mapping
Single-valued	#REQUIRED, #IMPLIED, Default Value	$A\_A_{direct/indirect}$
Multi-valued	#REQUIRED, #IMPLIED, Default Value	$A\_A_{indirect}$

Abbildung 9 - Sinnvolle Abbildungsstrategien für XML-Attribute

Für Attribute mit der Standarddeklaration „FIXED“ ist kein Mapping nötig, da der Fixwert im Abbildungswissen gespeichert werden kann.
Bei den Attributen, die eine andere Standarddeklaration als „FIXED“ aufweisen, ist nur die Multiplizität (m*ultiplicty*) des XML-Attributes ausschlaggebend, ob ein direktes oder indirektes Abbilden auf ein Attribut einer Relation der DB verwendet werden soll. Bei einmal vorkommenden Attributwerten kann entweder A_A_{direkt} verwendet werden, um den Wert in einem Attribut einer Relation der DB zu speichern oder auch $A\_A_{indirekt}$ auf Grund der Normalisierung einer Relation. Für mehrfaches Vorkommen von Attributwerten kann nur $A\_A_{indirekt}$ verwendet werden.

3.5. Metaschema im Detail

Das Kernstück von X-Ray, dessen Architektur im Abschnitt ***Architektur von X-Ray*** beleuchtet wurde, stellt das Metaschema dar. Es ist der Schlüsselmechanismus für die Generizität von X-Ray um DTDs auf relationale Schemata und vice versa abzubilden. Die Heterogenitäten zwischen den Datenmodellen und denen der Schemata, welche in [KAPP04] genauer erläutert werden, sowie die zuvor beschriebenen Abbildungsstrategien bilden die Basis für das Metaschema.
Das Metaschema besteht, wie in Abbildung 10 ersichtlich, aus drei Komponenten:

- *DBSchema*
- *XMLDTD*
- *XMLDBSchemaMapping*

Die *DBSchema*-Komponente ist zuständig für das Speichern der Informationen über die Schemata der Relationen, welche auf DTDs abgebildet werden sollen, um in ihnen gespeicherte Daten in XML-Dokumente exportieren zu können.
Die *XMLDTD*-Komponente speichert analog dazu die Schemainformationen der XML-Dokumente, nämlich der entsprechenden DTDs.
Die *XMLDBSchemaMapping*-Komponente überbrückt die Heterogenitäten der Datenmodelle und der Schemata und ermöglicht dadurch ein korrektes Abbilden.

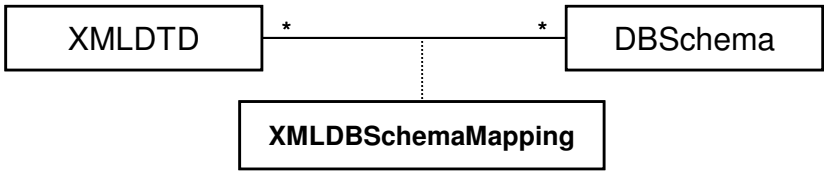


Abbildung 10 - Komponenten des X-Ray Metaschemas

Bei X-Ray wird das Abbildungswissen (*mapping knowledge*) auf zwei Arten verwaltet. Gespeichert wird das Abbildungswissen in einer relationalen Datenbank und es ist als objektorientierte Repräsentation im Arbeitsspeicher verfügbar.
Das Abbildungswissen wird zum Start einer X-Ray-Session aus dem RDBS in eine objektorientierte Repräsentation geladen und steht dann für das effiziente Zusammenstellen sowie Zerlegen von XML-Dokumenten zur Laufzeit der X-Ray-Session zur Verfügung. Im Folgenden werden die drei Komponenten näher betrachtet.

3.5.1. DB-Schema Komponente

Es muss nicht das gesamte relationale Schema gespeichert werden, sondern nur jene Relationen und Attribute, die relevant für das Abbilden auf DTDs sind. Dazu gehören auch jene Relationen die z.B. lediglich als Verbindungsrelationen zwischen zwei Basisrelationen dienen. Wie in Abbildung 11 ersichtlich, besteht die *DBSchema*-Komponente aus mindestens einer *DBRelation* mit zumindest einem *DBAttribute*, die beide zum *DBConcept* generalisiert werden.

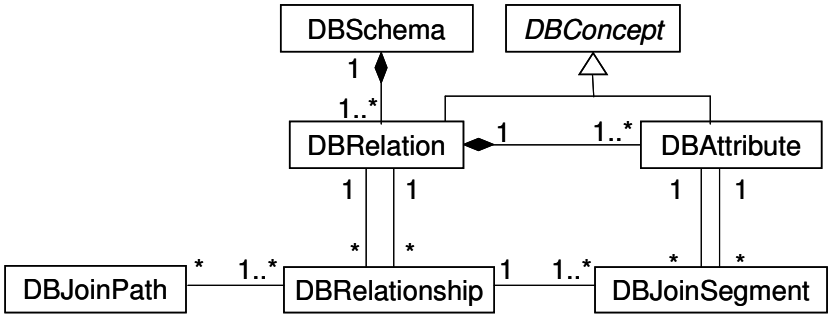


Abbildung 11 - DB-Schema Komponente des Metaschemas

Ein *DBAttribute* speichert einen Wert und zusätzlich, ob es sich um einen Primärschlüssel oder auch nur Teil des Primärschlüssels handelt. Mittels Beziehungen (*DBRelationship*) werden zwei Relationen verbunden, wobei diese Beziehungen Teil einer oder mehrerer Verbindungsstücke (*DBJoinSegment*) darstellen können. Die Attribute dieser Verbindungsstücke sind Primär- und Fremdschlüssel zweier verbundener Relationen. Setzt sich der Primärschlüssel aus mehreren Attributen zusammen, dann besteht eine Beziehung aus mehreren Verbindungsstücken. Sind Teile eines XML-Dokumentes auf verschiedene weiter verzweigte Relationen verteilt, muss die Information über entsprechende Verbindungspfade (*DBJoinPath*) gespeichert werden. Ein Verbindungspfad setzt sich aus mindestens einem oder mehreren Beziehungen (*DBRelationship*) zusammen.

3.5.2. XMLDTD-Komponente

Analog zur *DBSchema*-Komponente muss in der *XMLSchema*-Komponente nicht die gesamte DTD gespeichert werden, sondern nur jene Informationen, die zur Abbildung notwendig sind. Entsprechend dem Metawissen muss eine DTD einen Elementtyp (*XMLElemType*) haben, der die Wurzel (*root*) repräsentiert. Abbildung 13 ist die Verfeinerung der in Abbildung 12 dargestellten XMLDTD-Komponente

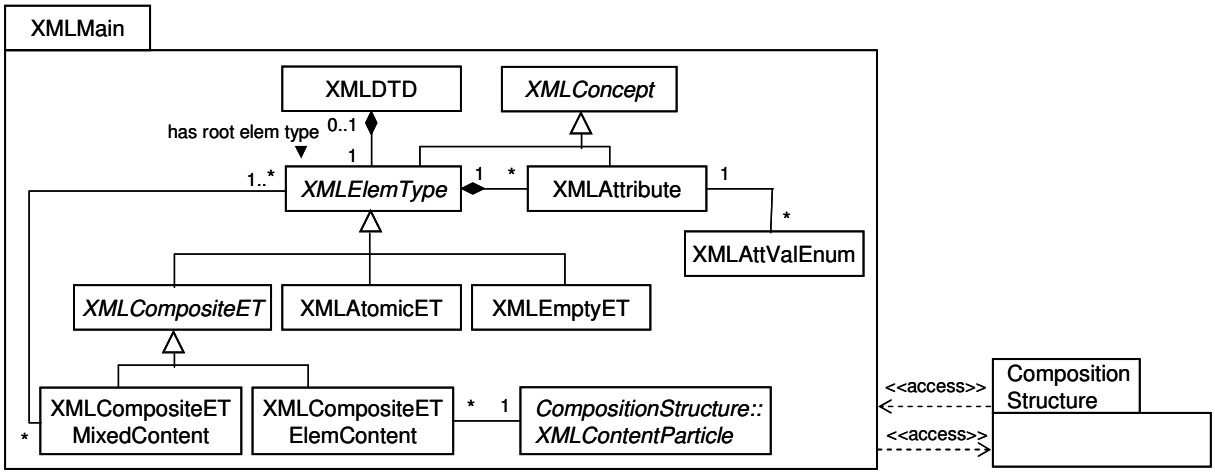


Abbildung 12 - XMLDTD-Komponente des Metaschemas

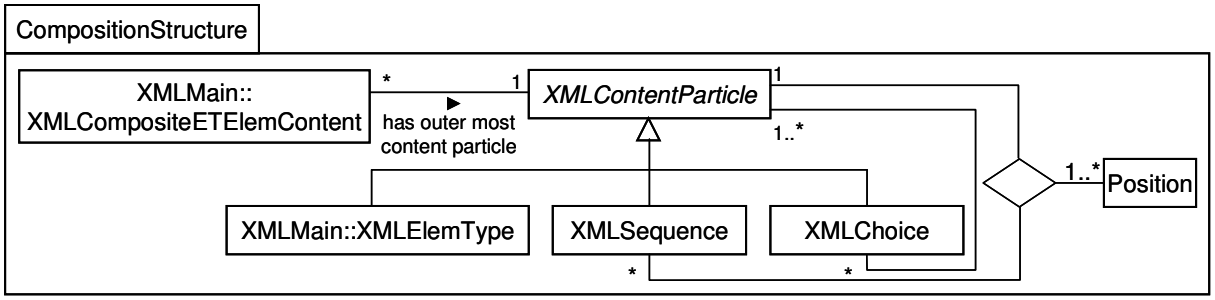


Abbildung 13 - Verfeinerung des zusammengesetzten ET

CompositeStructure dargestellt.

Der Wert sowie die Standarddeklaration von XML-Attributen, die einem Elementtypen angehören werden in *XMLAttributes* gespeichert. *XMLElemType* und *XMLAttribute* werden zu *XMLConcept* generalisiert.

Die möglichen Werte für Enumerations-Attribute werden in *XMLAttVAlEnum* gespeichert. *XMLElemType* wird weiter unterteilt in die verschiedenen Ausprägungsarten, die im Abschnitt ***Eigenschaften von XML-Elementen*** näher ausgeführt sind. Die verschachtelte Struktur eines Elementtyps wird mittels des Paketes *CompositeStructure* beschrieben. Für Elementtypen, die im Zuge einer *XMLChoice* oder *XMLSequence* vorkommen, wird die Kardinalität berücksichtigt und bei Elementtypen einer *XMLSequence* zusätzlich die Position des Elementtyps in der *Sequence*. Zusätzlich können mittels *XMLContentParticle* beliebige Kombinationen von *Sequence* und *Choice* beschrieben werden.

3.5.3. XMLDBSchemaMapping-Komponente

Das Abbildungswissen wird durch die Assoziationen zwischen Objektklassen der *XMLDTD*-Komponente und jenen der *DBSchema*-Komponente repräsentiert. In Abbildung 14 werden diese Assoziationen durch fette Linien dargestellt.

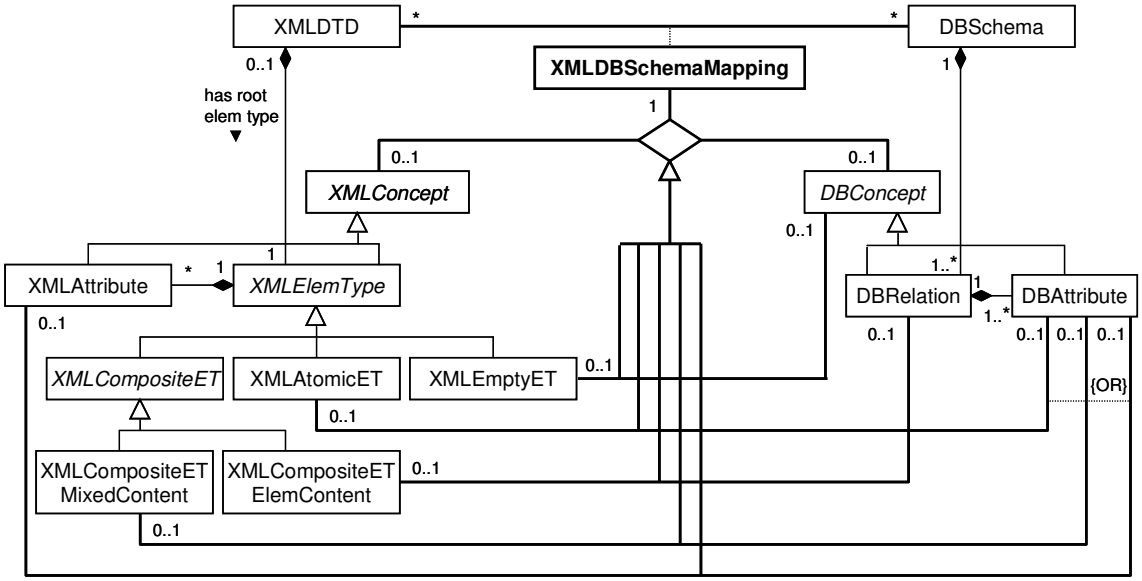


Abbildung 14 - Ausschnitt aus dem Metaschema (XMLDBSchemaMapping-Komponente)

Durch die ternäre Beziehung zwischen *XMLConcept*, *DBConcept* und *XMLDBSchemaMapping* wird ein Hauptziel von X-Ray, nämlich die Abbildungen zwischen mehreren DTDs und Datenbankschemata zu speichern, realisiert. Wie ein Elementtyp tatsächlich abgebildet wird, hängt, wie im Abschnitt ***Eigenschaften von XML-Elementen*** diskutiert, von dessen Art und Aufbau ab. Daher werden bis auf zwei Ausnahmen alle Blätter der *XMLElemType*-Hierarchie auf *DBAttribute* abgebildet. Dies sind der *XMLEmptyET* und der *XMLCompositeETElemContent*. Im Besonderen ist das Abbilden von *XMLEmptyET* nicht genau vordefinierbar, da es abhängig davon ist, ob diese Art von Elementtyp Attribute enthält oder nicht sowie von dessen Kardinalität. Das Abbilden der XML-Attribute erfolgt gemäß der im Abschnitt ***Eigenschaften von XML-Attributen*** vorgestellten Abbildungsarten.

4. Funktionalität des Prototyps

Im vorhergehenden Kapitel wurde der Aufbau, die Abbildungskonzepte und das Metaschema von X-Ray vorgestellt. Wie in dem Kapitel beschrieben, werden DTDs als Basis zum Abbilden von XML-Dokumenten auf ein RDBS verwendet. Im Abschnitt **Future work** von [KAPP04] wird vorgeschlagen, X-Ray dahingehend zu erweitern, um auch das Abbilden von XML-Dokumenten auf Basis von XML Schema zu realisieren. Diese erweiterte Variante von X-Ray trägt die Bezeichnung X-Rayxs, und wurde im Rahmen der Diplomarbeit [KRAU05] entwickelt.
Der Grund hierfür ist, dass es immer mehr zum Standard wird, XML Schema zum Beschreiben der Struktur von XML-Dokumenten zu verwenden. Die Vorteile (und auch Nachteile) von XML Schema gegenüber DTD werden im Kapitel **Vergleich zwischen DTD und XML Schema** näher beleuchtet.
Der Terminus XML Schema wird für das Schemakonzept von [W3C05] verwendet.
Da jedoch XML Schema ungleich umfangreicher im Vergleich zu DTD ist, konnten nicht alle Konzepte von XML Schema [W3C05] im Zuge der Diplomarbeit berücksichtigt werden. Ein Überblick über die berücksichtigten und explizit nicht berücksichtigten Konzepte von XML Schema sowie eine genauere Beschreibung der Konzepte wird in den Abschnitten **Unterstützte XML Schema Konzepte** bzw**. Nicht unterstützte XML Schema Konzepte** gegeben.
Es mussten alle drei Komponenten des Metaschemas überarbeitet bzw. neu entwickelt werden, um XML Schema zu unterstützen. Die Beschreibung des Metaschemas mitsamt seiner Komponenten wird in Diplomarbeit [KRAU05] gegeben.

4.1. Unterstützte XML Schema Konzepte

Im folgenden Abschnitt werden die in X-Rayxs berücksichtigten Konzepte von XML Schema erläutert. Der Aufbau orientiert sich dabei an der „XML Schema - Structures Quick Reference Card“ [DVIN03A], in der sich auch die genaue Syntax zur Umsetzung der Konzepte befindet. Da die Berücksichtigung aller Konzepte von XML Schema, wie bereits erwähnt, den Rahmen der Diplomarbeit sprengen würde, mussten Konzepte selektiert werden die in X-Rayxs umgesetzt werden sollten. Die Konzepte wurden dahingehend ausgewählt, um einen größtmöglichen Teil der in

XML Schema vorhandene Konzepte umzusetzen, der nötig ist, um Standard-XML-Anwendungen behandeln zu können. Darüber hinaus wurde bei der Auswahl darauf geachtet, dass die im Kapitel **„Vergleich zwischen DTD und XML Schema“** vorgestellten Vorteile von XML Schema ebenso enthalten sind.
Die Paragraphen neben den Konzeptbezeichnungen beziehen sich auf das nicht normative Dokument „XML Schema Teil 0: Einführung“ [W3C04C]. Diese Einführung beschreibt die Sprachkonstrukte anhand zahlreicher Beispiele, ergänzt durch etliche Verweise auf die normativen Texte.

1. Atomare Elemente §2.3

Ein Element auf Basis eines vordefinierten Datentyps (simple predefined data type) wird als atomares Element bezeichnet. Es besteht somit aus einem unteilbaren Wert. Beispielsweise besteht zwar ein String aus Zeichen, diese werden jedoch als Einheit betrachtet.

```
<element name="email" type="string"/>
```

Abbildung 15 - Beispiel Atomares Element

2. Komplexe Datentypen §2.2

Eine komplexe Typdefinition wird verwendet, um ein Inhaltsmodell, das Elemente und/oder Attribute enthält, zu spezifizieren. Wenn komplexe Typen einen Namen besitzen, können sie mehrfach zur Deklaration von Elementen verwendet werden, auch in externen Schemadokumenten. Ansonst können anonyme Typdefinitionen innerhalb einer Elementdeklaration erfolgen.

Mit Hilfe des komplexen Datentyps können folgende Inhalte definiert werden:

- Einfacher Inhalt (simple content)
- Komplexer Inhalt (complex content) in den Varianten:
  - Nur-Element Inhalt
  - Gemischter Inhalt (in X-Rayxs nicht unterstützt; siehe dazu Abschnitt **Komplexe Datentypen mit gemischtem Inhalt**)
  - Leerer Inhalt (siehe dazu Abschnitt **Leere Elemente**)

<table>
<tr><td rowspan="3">Simple Type</td><td colspan="3">Complex Type</td></tr>
<tr><td rowspan="2">Simple Content</td><td colspan="2">Complex Content</td></tr>
<tr><td>Element only</td><td>mixed</td><td>empty</td></tr>
</table>

Abbildung 16 - Übersicht Datentypen

Abbildung 16 soll den Zusammenhang der möglichen Datentypen verdeutlichen. Bei den komplexen Datentypen mit Nur-Element Inhalt dürfen Elemente, die mit diesem Datentyp deklariert werden, keinen einfachen eigenen Text beinhalten. Im Gegensatz dazu ist dies bei gemischten Inhalten möglich. Komplexe Datentypen mit einfachem Inhalt werden dazu verwendet, um einfache vordefinierte Datentypen mit Attributen zu versehen bzw. um diese erweitern oder einschränken zu können.

Das Inhaltsmodel eines komplexen Datentyps beschreibt die Ordnung und Struktur der im Typ enthaltenen Elemente. Es setzt sich aus Modellgruppen (*sequence*, *choice*, *all*), Elementen und wildcards (in X-Rayxs nicht unterstützt) zusammen.

Die Inhaltsmodelle von benannten komplexen Typen können außerdem mit Hilfe der Vererbungsmechanismen erweitert (*extension*) oder eingeschränkt (*restriction*) werden.

Im folgenden Beispiel (vgl. Abbildung 17) wird ein benannter komplexer Typ „address“ definiert. Die Definition enthält in diesem Fall drei weitere Elementdefinitionen und eine Attributbeschreibung, wobei „village“ wiederum aus einem global definierten komplexen Datentyp besteht.

Die Reihenfolge der Elemente ist durch das *<sequence>*-Element festgelegt.

```
<complexType name="address">
     <sequence>
          <element name="street" type="string"/>
          <element name="village"
               type="villageType"/>
          <element name="country" type="string"/>
     </sequence>
     <attribute name="postalCode" type="string"
                use="required"/>
</complexType>
```

Abbildung 17 - Beispiel komplexer Datentyp

3. Leere Elemente §2.5.3

Ein leeres Element speichert keinen Wert zwischen den Element-Tags, die Information liegt im Namen des Tags bzw. in den möglichen Attributen. Durch das Vorhandensein bzw. Nichtvorhandensein eines leeren Elements in der XML-Instanz entsteht ein weiterer Informationsgehalt des Elements. Ein leeres Element darf keine Kindelemente beinhalten. Das leere Element ist also ein komplexer Datentyp, dessen leerer Inhalt die Einschränkung von „anyType“ auf die leere Menge ist.

```
<element name="pool" minOccurs="0" maxOccurs="unbounded">
     <complexType/>
</element>
```

Abbildung 18 - Beispiel leeres Element

4. Vererbung §4

Ähnlich wie in objektorientierten Programmiersprachen ist es auch in XML Schema möglich, dass Typen von anderen Typen abgeleitet werden.
Zur Unterstützung der Wiederverwendbarkeit und Erhöhung der Strukturierung des Entwurfs definiert XML Schema ein Vererbungskonstrukt zur Bildung neuer komplexer Typen auf der Basis bereits bestehender. Diese müssen allerdings so genannte „benannte“ komplexe Typen sein, um auf sie verweisen zu können.

Zwei verschiedene Ableitungsarten werden angeboten:

- Ableitung durch Einschränkung (*restriction*) - der erbende Subtyp beschreibt eine engere Definition des Supertypen
- Ableitung durch Erweiterung (*extension*) - der erbende Subtyp erweitert die Definition des Supertypen

XML Schema unterstützt darüber hinaus das Konzept der abstrakten Typen. Abstrakte Typen können in keinem XML-Instanzdokument vorkommen. Sie dienen nur anderen Typdefinitionen, um den abstrakten Typ einzuschränken oder zu erweitern. Umgekehrt kann eine Typdefinition als endgültig (*final*) deklariert werden, um zu verhindern, dass weitere Typen von diesem Typ abgeleitet werden.
Des Weiteren lassen sich auch vordefinierte Datentypen zu neuen Datentypen ableiten, um zum Beispiel den Wertebereich des „decimal“ Typs auf Zahlen mit zwei Nachkommastellen für einen neuen Typ „Währung“ einzuschränken [KENN00] (Benutzerdefinierte einfache Typen (simple data type) werden in XRayxs nicht unterstützt).
Im folgenden Beispiel erbt „standardRoomType“ von „roomType“ und erweitert diesen um die Elemente „shower“ und „tv“.

```
<complexType name="standardRoomType">
   <complexContent>
      <extension base="act:roomType">
            <sequence>
                  <element name="tv" type="int"
                        minOccurs="0" maxOccurs="2"/>
                  <element name="shower" minOccurs="0">
                        <complexType/>
                  </element>
            </sequence>
      </extension>
   </complexContent>
</complexType>
```

Abbildung 19 - Beispiel Vererbung

5. Identifizierung und Referenzierung §5

Die XML Schema Identifizierungen und Referenzierungen werden verwendet, um eindeutige Werte, Schlüssel oder Verweise auf Schlüssel festzulegen. Dies wird durch die Elemente *unique*, *key* und *keyref* gewährleistet.

Der Unterschied zwischen *key* und *unique* besteht darin, dass das mit *key* spezifizierte Element oder Attribut bzw. Kombinationen von Elementen und Attributen vorkommen muss und eindeutig ist, *unique* besagt, dass das Element, falls es vorkommt, eindeutig sein muss. *Keyref* entspricht im Wesentlichen dem Fremdschlüsselkonzept in RDBS (Referenzielle Integrität).

Der Wert eines *<keyref>*-Elements muss also mit einem Wert (Schlüssel) an anderer Stelle im Dokument, welcher durch das *<key>*-Element beschrieben wird, übereinstimmen.

Das XML Schema Schlüsselkonzept kann auf alle Elemente und Attribute angewandt werden, unabhängig von deren Typ. Der Bereich, in dem Eindeutigkeit gelten soll, kann festgelegt werden. Eindeutigkeit kann auch für Kombinationen aus mehreren Elementen und Attributen festgelegt werden.

Die Identifizierungen und Referenzierungen werden durch ein *<selector>*-Element und einem oder mehreren *<field>*-Elementen beschrieben. Das *<selector>*-Element enthält einen XPath Ausdruck, der eine Menge von Elementen referenziert. Diese stellen den Bereich dar, für den Eindeutigkeit gewährleistet wird. Das *<field>*-Element enthält einen XPath Ausdruck, der die Attribute bzw. Elemente identifiziert, die einen eindeutigen Wert besitzen sollen [ECKS04].

Im folgenden Beispiel ist der Schlüssel das Attribut „id“ vom Element „accommodation“. Des Weiteren wird durch das *<keyref>-Element* festgelegt, dass der Wert des Elements „winner“ bereits als Schlüssen („id“ von „accommodation“) vorhanden sein muss.

```
<key name="AccKey">
      <selector xpath="accommodation"/>
      <field xpath="@id"/>
</key>
<keyref name="AwardWinnerRef" refer="AccKey">
      <selector xpath="accAwards/award"/>
      <field xpath="winner"/>
</keyref>
```

Abbildung 20 - Beispiel Identifizierungen und Referenzierungen

6. Vordefinierte einfache Datentypen §2.3

XML Schema verfügt insgesamt über 44 vordefinierte einfache Datentypen *(simple predefined datatypes)*. Abbildung 21 zeigt eine hierarchische Gliederung dieser Typen. Eine weitere Übersicht der Datentypen und deren Wertebereiche befindet sich in „XML Schema - Data Types Quick Reference“ [DVIN03B]. Neue einfache Typen können durch Ableiten erzeugt werden. Dabei wird durch Einschränken des Wertebereichs eines bestehenden einfachen vordefinierten Datentyps ein neuer Datentyp erzeugt (Einfache Typen (simple data type) werden in X-Rayxs nicht unterstützt, somit auch nicht das Ableiten von vordefinierten einfachen Datentypen).

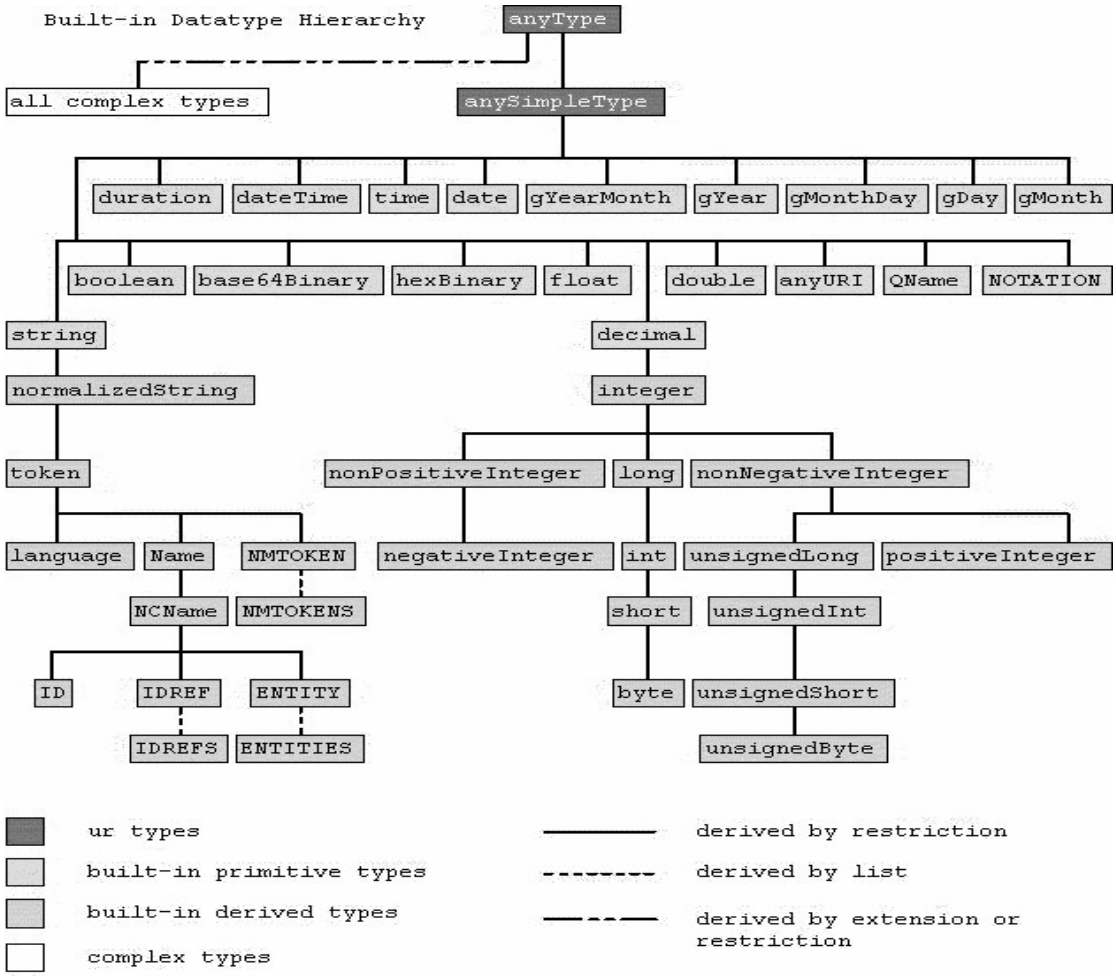


Abbildung 21 - XML Schema Datentyp Hierarchie [W3C04D]

7. Lokale bzw. globale Elemente, Attribute und Datentypen §2.2.2 / §3

In XML Schema besteht die Möglichkeit Datentypen, Elemente und Attribute global oder lokal zu definieren.

Global bedeutet, dass sie direkt unterhalb des *<schema>*-Elements spezifiziert werden. Diese Elemente, Attribute und Datentypen sind im gesamten Schema sichtbar und lassen sich von beliebigen Stellen aus über ihren Namen referenzieren. Für Elemente, Attribute und Datentypen lokaler Definitionen gilt das nicht. Sie lassen sich ausschließlich an der Position benutzen, an der sie definiert werden. Im Gegensatz zu Datentypen müssen Elementen und Attributen immer Namen gegeben werden, unabhängig davon, ob sie global oder lokal definiert sind. Diese Namen werden im XML-Dokument für die Auszeichnung benutzt. Durch die Namensgebung können Typdefinitionen bzw. Element- und Attribut-Deklarationen mehrfach verwendet werden

In folgendem Beispiel ist das Element “name” global deklariert und kann somit referenziert werden. Das Element „accommodation“ hingegen wird in einem lokal definierten Datentyp deklariert und referenziert selbst wieder einen global definierten Datentyp „accommodationType“.

```
<schema targetNamespace="http://www.ifs.jku.at/
      XRay/AccommodationSchema" …>

<element name="name" type="string"/>
<element name="accommodations">
      <complexType>
            <sequence>
                  <element name="accommodation"
                        type="accommodationType" minOccurs="0"
                        maxOccurs="unbounded"/>
                  <element name="accAwards"
                        type="accAwardsType"/>
            </sequence>
      </complexType>
      …
<complexType name="accommodationType">
      <sequence>
            <element ref="name"/>
            <element name="address" type="addressType"/>
      …
```

Abbildung 22 - Beispiel lokale bzw. globale Elemente, Attribute und Datentypen

8. Namensräume §3

Um die Wiederverwendbarkeit und die Kombinationsmöglichkeiten von Elementen, Attributen und Typdefinitionen von XML Schemata zu erhöhen, müssen diese über eindeutige Bezeichnungen verfügen, um Verwechslungen zu verhindern. Um dies zu erreichen, gibt es so genannte Namensräume (*namespaces*). Nur durch die Vergabe von Namensräumen können die benutzten Elemente, Attribute und Typen überhaupt identifiziert werden.

Um nicht immer den gesamten Namensraum vor ein Element oder Attribut zu schreiben, lassen sich beliebige Präfixe vergeben, die als abkürzende Schreibweise gedacht sind. Derartige Abkürzungen lassen sich durch den Präfix-Zusatz am *<xmlns>*-Element realisieren.

Mit *„xmlns:xs="http://www.w3.org/2001/XMLSchema"* lässt sich beispielsweise der Namensraum für die XML Schema Schemadefinition durch *xs:* abkürzen.

Der aktuelle Namensraum, für den die Elemente eines Schemas geschrieben werden, kann durch die Verwendung des *<targetNamespace>*-Elements im *<schema>*-Element für alle Elemente gesetzt werden.
Des Weiteren gibt es einen Standardnamensraum, er ist überall gültig, wo sonst keine besondere Kennzeichnung durch ein Namensraumpräfix mit zugeordnetem Namensraum-URI existiert. Der URI (Uniform Resource Identifier) für den Standardnamensraum wird durch das Attribut *xmlns=" Standard-Namensraum-URI"* gesetzt.
Das Attribut *elementFormDefault* hat ebenfalls Auswirkungen auf die Namensräume. Es kann einen der beiden Werte *qualified* oder *unqualified* annehmen. Der Wert *unqualified* bewirkt, dass in Instanzen nur globale Elemente durch einen Namensraum Präfix qualifiziert werden dürfen. Lokale Elemente werden implizit über das globale Element qualifiziert, in das sie eingebettet sind. Ist der Wert von *elementFormDefault* dagegen auf *qualified* gesetzt, so müssen alle Elemente in XML-Instanzen qualifiziert werden. Die explizite Angabe des Namensraum Präfixes in jedem einzelnen Element kann aber durch die Deklaration eines Standardnamensraumes umgangen werden [HOLZ04]. X-Rayxs unterstütz für das *<elementFormDefault>*-Element nur den Wert qualified. Eine mögliche Namensraumdefinition zeigt, das Beispiel in Abbildung 23 .

```
<schema targetNamespace="http://www.ifs.uni-
linz.ac.at/XRay/AccommodationSchema"
elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ac="http://www.ifs.uni-
lnz.ac.at/XRay/AccommodationSchema"
xmlns:su="http://www.ifs.uni-linz.ac.at/XRay/Supervision">
```

Abbildung 23 . Beispiel Namensräume

9. Schema Management §4.1 / §4.5 / §5.1

Wenn Schemata größer werden, ist es wünschenswert, ihren Inhalt auf mehrere Dateien aufzuteilen. Dadurch wird die Wartung, die Zugriffsregelung und auch die Lesbarkeit des Schemas verbessert.

Um ein XML Schema in ein anderes einzubinden, bzw. in dessen Namensraum, wird das *<include>*-Element verwendet. Dieses *<include>*-Element holt die Deklarationen und Definitionen aus einem fremden XML Schema und macht sie im Namensraum verfügbar. Wenn das *<include>*-Element verwendet wird, müssen jedoch beide Schemata denselben Zielnamensraum haben. Das *<redefine>*-Element erfüllt den selben Zweck, jedoch mit dem Unterschied, dass Modifikationen an den eingebundenen Definitionen und Deklarationen vorgenommen werden können (vgl. Abbildung 24). Sollen nun Definitionen oder Deklarationen aus einem XML Schema eingebunden werden, das nicht denselben Zielnamensraum hat, kann das *<import>*-Element verwendet werden (vgl. Abbildung 24).

```
<schema targetNamespace="http://www.ifs.uni-
      linz.ac.at/XRay/AccommodationSchema"
      elementFormDefault="qualified"
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:ac="http://www.ifs.uni-
      linz.ac.at/XRay/AccommodationSchema"
      xmlns:su="http://www.ifs.uni-
      linz.ac.at/XRay/Supervision">
<import namespace="http://www.ifs.uni-
      linz.ac.at/XRay/Supervision"
      schemaLocation="xray_supervision.xsd"/>
<redefine schemaLocation="xray_management.xsd">
      <complexType name="managerType">
            <complexContent>
                  <extension base="ac:managerType">
                        <sequence>
                              <element name="salary"
                                    type="positiveInteger"/>
                        </sequence>
                        <attribute name="mgrId"
                              type="positiveInteger"/>
                  </extension>
            </complexContent>
      </complexType>
</redefine>
…
```

Abbildung 24 - Beispiel Schema Management

4.2. Nicht unterstützte XML Schema Konzepte

1. Komplexe Datentypen mit gemischtem Inhalt §2.5.2

Bei komplexen Datentypen mit gemischtem Inhalt ist der Wert des *<mixed>*-Attributes „true“. Somit können Elemente, die mit einem komplexen Datentyp mit gemischtem Inhalt definiert werden, aus weiteren Elementen und einem eigenen einfachen Text bestehen.

2. Einfache Datentypen §2.3

Das *<simpleType>*-Element wird verwendet um Ableitungen von einfachen Typen durch Einschränkung (*restriction*), sowie Listtypen und Vereinigungstypen zu definieren.

3. Schema Dokumentation §2.6

XML Schema definiert drei Elemente, um Schemata mit Anmerkungen zu versehen, die sowohl für menschliche Leser als auch für Anwendungen gedacht sind.
Für Dokumentation, die für Personen gedacht sind, ist das *<documentation>*-Element vorgesehen. Für Anwendungen hingegen das *<appInfo>*-Element. *<dokumentation>* und *<appInfo>* sind Subelemente vom *<annotation>*-Element, welches wiederum am Anfang der meisten Schema-Konstrukte erscheinen darf.

4. Notation

Notationen dienen zur Beschreibung des Formats von Nicht-XML-Daten in einem XML-Dokument. Notationen werden in X-Rayxs nicht unterstützt.

5. Any Konzept *§5.5*

Um flexible Dokumente zu erzeugen, gibt es die vordefinierten Typen *anySimpleType* und *anyType*. Sie erlauben für Elemente, die mit diesen Typen deklariert werden, alle einfachen bzw. jeden beliebigen Typ im Instanzdokument anzunehmen.

Des Weiteren stellt XML Schema die Platzhalter *any* und *anyAttribute* zur Verfügung, die verwendet werden können, damit Elemente und Attribute aus einem angegebenen Namensraum in einem Inhaltsmodell auftreten können.

6. Element/Attribut Gruppe *§2.7 / §2.8*

Gruppen verbessern die Strukturierung und Wiederverwendbarkeit von Elementen. Mit ihnen lassen sich verschiedene Elemente zusammenfassen. Dabei sind die gleichen Mechanismen zu verwenden wie bei komplexen Datentypen. Diese Gruppen können nur global erzeugt werden und sind somit an beliebiger Stelle im Schema referenzierbar.

Gruppierung kann ebenso für Attribute verwendet werden. Da Attribute keine Strukturen beinhalten, werden in Attributgruppen die Attribute in Form einer Liste angeführt.

4.3. Anwendungsfälle

In diesem Abschnitt werden die Anwendungsfälle (*use cases*) für die Laufzeitphase des X-Rayxs Prototyps vorgestellt. Die Anwendungsfälle beschreiben den typischen Ablauf der drei Hauptfunktionen des Prototys sowie das Login an den Prototyp. Es gibt demnach vier Anwendungsfälle (vgl. Abbildung 25):

1. Login am X-Rayxs-Prototyp
2. Import von Daten einer XML-Datei
3. Export von Daten in eine XML-Datei
4. XQuery-Abfrage auf gespeicherte Daten

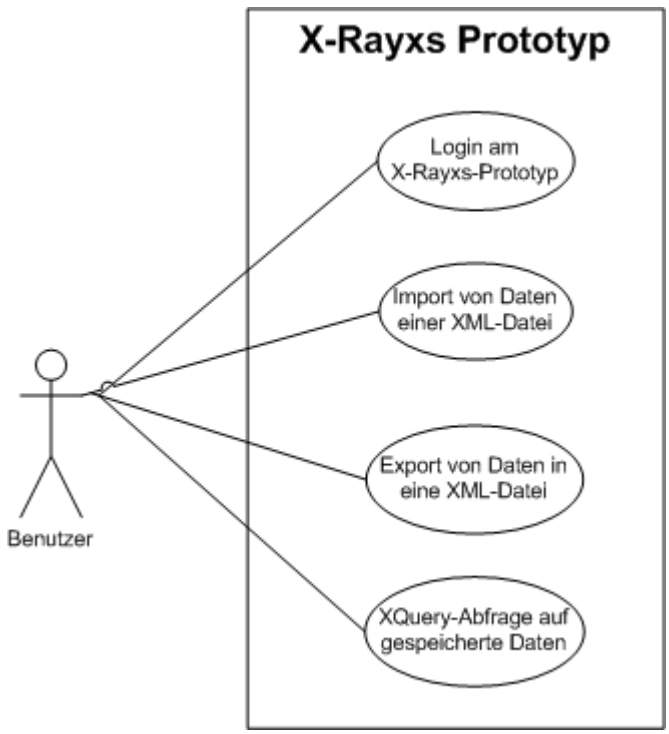


Abbildung 25 - Use case Diagramm

Nachfolgend werden die Anwendungsfälle detailliert beschrieben.

Use Case Name	**Login am X-Rayxs Prototyp**
Kurzbeschreibung	Um Zugriff auf die Funktionen des Prototyps zu erhalten, muss sich der Benutzer mit einem gültigen Benutzernamen und Passwort am Prototyp anmelden. Erst durch Eingabe dieser Daten ist der indirekte Zugriff auf das Metaschema und dadurch auf die Funktionalität des Prototypen möglich.
Ablauf	1. Starten des Prototyps 2. Eingabe eines gültigen Benutzernamens 3. Eingabe eines gültigen Passworts 4. Bestätigen des Logins 5. Nach dem erfolgreichen Überprüfen der Eingabedaten wird das Hauptfenster des Prototyps angezeigt
Fehler	1. Verbindung zur Datenbank, in der das Metaschema gespeichert ist, ist unterbrochen oder nicht vorhanden 2. Falsche Kombination von Benutzername und Passwort
Fehlerbehandlung	Tritt während des Logins der Daten einer der oben aufgelisteten Fehler auf, wird eine Fehlermeldung mit entsprechenden Informationen über den Fehler angezeigt. Benutzername und Passwort müssen erneut eingegeben werden.
Auslöser	Benutzer
Vorbedingung	Verbindung zur Metaschema-Datenbank vorhanden
Endzustand	Hauptfenster des Prototypen ist geöffnet und die Auswahl der Funktionen ist ermöglicht. Verbindung zur Datenbank, in der das Metaschema gespeichert ist, ist hergestellt. Verbindung zur Datenbank, in der die Daten der XML – Datei gespeichert werden sollen bzw. sind, ist hergestellt.
Datum	23.02.2005

Use Case Name	**Import von Daten einer XML-Datei**
Kurzbeschreibung	Es werden die in der XML-Datei gespeicherten Daten unter Verwendung des Metaschemas in den entsprechenden Relationen des RDBS gespeichert.
Ablauf	1. Öffnen des Import-Dialogs im Prototyp 2. Auswahl der zu importierenden XML-Datei 3. Auswahl einer gespeicherten Abbildungsvariante 4. Bestätigen des Imports 5. Daten werden aus der XML-Datei gelesen und entsprechend der Daten des Metaschemas in die entsprechenden Relationen gespeichert 6. Nach dem erfolgreichen Import wird eine Erfolgsmeldung angezeigt
Fehler	1. Verbindung zur Datenbank, in der das Metaschema gespeichert ist, ist unterbrochen oder nicht vorhanden 2. Verbindung zur Datenbank, in der die Daten der XML – Datei gespeichert werden sollen, ist unterbrochen oder nicht vorhanden 3. Kein Metaschema in der Datenbank gespeichert 4. Keine Abbildungsvariante für die zu importierende XML-Datei gespeichert 5. XML-Datei ist nicht wohlgeformt bzw. nicht valide 6. Fehler beim Zugriff auf die XML-Datei 7. Fehler beim Speichern der Daten der XML-Datei in die entsprechenden Relationen der Datenbank 8. Wert eines Primärschlüssels schon in einer Relation vorhanden
Fehlerbehandlung	Tritt während des Importes der Daten einer der oben chronologisch aufgelisteten Fehler auf, wird eine Fehlermeldung mit entsprechenden Informationen über den Fehler bzw. mit dem Fehlercode der Datenbank angezeigt. Bei Auftreten eines Fehlers während der Datenbanktransaktionen wird die Datenbank in den Ausgangszustand zurückgesetzt.

Auslöser	Benutzer
Vorbedingung	Verbindung zur Metaschema-Datenbank vorhanden. Verbindung zum RDBS, in dem die Daten der XML-Datei gespeichert werden sollen, ist vorhanden. Benutzer hat gültigen Benutzernamen und Passwort beim Login eingegeben. Abbildungsvariante für die zu importierende XML-Datei ist im Metaschema gespeichert.
Endzustand	Daten aus der XML-Datei sind entsprechend der im Metaschema gespeicherten Abbildungsvariante in Relationen des RDBS gespeichert.
Datum	23.02.2005

Use Case Name	**Export von Daten in eine XML-Datei**
Kurzbeschreibung	Es werden die in der RDBS gespeicherten Daten, die einer Abbildungsvariante und somit einem XML-Schema zugehörig sind, in eine XML-Datei exportiert.
Ablauf	1. Öffnen des Export-Dialogs im Prototyp 2. Auswahl einer gespeicherten Abbildungsvariante 3. Öffnen eines Speicherdialogs, zwecks Auswahl des Speicherorts und Namen der XML-Datei 4. Auswahl, ob Schemadatei an den Speicherort der XML – Datei kopiert werden soll 5. Bestätigen des Exports 6. Daten werden aus RDBS mittels des Metaschemas ausgelesen und die XML-Datei wird generiert 7. Nach erfolgreichem Export wird eine Erfolgsmeldung angezeigt
Fehler	1. Verbindung zur Datenbank, in der das Metaschema gespeichert ist, ist unterbrochen oder nicht vorhanden 2. Verbindung zur Datenbank, in der die Daten der XML – Datei gespeichert sind, ist unterbrochen oder nicht vorhanden

	3. Kein Metaschema in der Datenbank gespeichert 4. Keine Abbildungsvariante für die zu exportierende XML-Datei gespeichert 5. Fehler beim Generieren der XML-Datei (z.B. falsche Abbildungsdaten) 6. Fehler beim Speichern der XML-Datei 7. Fehler beim Kopieren der Schemadatei
Fehlerbehandlung	Tritt während des Exportes der XML-Daten einer der oben chronologisch aufgelisteten Fehler auf, wird eine Fehlermeldung mit entsprechenden Informationen über den Fehler bzw. mit dem Fehlercode der Datenbank angezeigt.
Auslöser	Benutzer
Vorbedingung	Verbindung zur Metaschema-Datenbank vorhanden. Verbindung zum RDBS, in dem die Daten der XML-Datei gespeichert werden sollen, ist vorhanden. Benutzer hat gültigen Benutzernamen und Passwort beim Login eingegeben. Abbildungsvariante für die zu exportierende XML-Datei ist im Metaschema gespeichert. Daten sind in den Relationen der RDBS vorhanden.
Endzustand	XML-Datei mit exportierten Daten sowie zugehörige XML Schema-Datei (wenn vom Benutzer gewünscht) befindent sich an dem vom Benutzer spezifizierten Speicherort.
Datum	23.02.2005

Use Case Name	**XQuery-Abfrage auf gespeicherte Daten**
Kurzbeschreibung	Es können beliebige XQuery-Abfragen erstellt werden. Diese werden auf eine temporäre XML-Datei abgesetzt, die mittels einer zuvor selektierten Abbildungsvariante erzeugt wird.
Ablauf	1. Öffnen des XQuery-Dialogs im Prototyp 2. Auswahl einer gespeicherten Abbildungsvariante 3. Eingabe einer validen XQuery-Abfrage 4. Daten werden aus RDBS mittels des Metaschemas ausgelesen und es wird eine temporäre XML-Datei generiert 5. XQuery wird auf temporäre XML-Datei abgesetzt 6. Anzeige des Abfrageresultats
Alternativwege 1	1. Um das Abfrageresultat in eine Datei zu exportieren, muss der Benutzer einen Speicherdialog öffnen. Mit diesem werden die Auswahl des Speicherorts und die Eingabe eines Namens der Datei durchgeführt. 2. Bestätigen des Speicherns des Abfrageresultats 3. Speichern der Datei 4. Nach erfolgreichem Speichern wird eine Erfolgsmeldung angezeigt
Auslöser	Benutzer
Vorbedingung	Verbindung zur Metaschema-Datenbank vorhanden Verbindung zum RDBS, in dem die Daten der XML-Datei gespeichert werden sollen, ist vorhanden Benutzer hat gültigen Benutzernamen und Passwort beim Login eingegeben. Abbildungsvariante für die zu exportierende XML-Datei ist im Metaschema gespeichert. Daten sind in den Relationen der RDBS vorhanden
Endzustand 1	Resultat der XQuery-Abfrage ist angezeigt
Endzustand 2	Datei mit dem Resultat der XQuery-Abfrage an dem vom Benutzer spezifizierten Speicherort gespeichert
Fehler	1. Verbindung zur Datenbank, in der das Metaschema gespeichert ist, ist unterbrochen oder nicht vorhanden

	2. Verbindung zur Datenbank, in der die Daten der XML – Datei gespeichert sind, ist unterbrochen oder nicht vorhanden 3. Kein Metaschema in der Datenbank gespeichert 4. Keine Abbildungsvariante für die zu exportierende XML-Datei gespeichert 5. Fehler beim Generieren der temporären XML-Datei (z.B. falsche Abbildungsdaten) 6. Ungültige XQuery (Syntaxfehler, flasche Abfragekriterien) 7. Fehler beim Speichern der Datei mit dem Resultat der XQuery-Abfrage
Fehlerbehandlung	Tritt während dem Erstellen der temporären XML-Datei oder beim Durchführen der XQuery-Abfrage einer der oben chronologisch aufgelisteten Fehler auf, wird eine Fehlermeldung mit entsprechenden Informationen über den Fehler angezeigt.
Datum	23.02.2005

4.4. Benutzerschnittstelle

In diesem Kapitel sollen die Benutzerschnittstellen des Prototypen X-Rayxs dargestellt werden. Eine genaue Beschreibung der Benutzerschnittstellen in Verbindung mit deren Funktionen befindet sich im Benutzerhandbuch, das dem Anhang beigefügt ist. Bei der Umsetzung der Benutzerschnittstellen ist auf das einheitliche Erscheinungsbild der drei Hauptfunktionen und eine möglichst einfache Bedienung, geachtet worden.
Aus den Anwendungsfällen lassen sich folgende Benutzerschnittstellen ableiten:

- Login am XRAYxs-Prototypen
- Import - Daten aus einem XML-Dokument importieren
- Export - Daten in ein XML-Dokument exportieren
- XQuery – Abfragen auf ein gespeichertes XML Schema anwenden

Abbildung 26 - X-Rayxs Login

Abbildung 26 zeigt das Anmeldefenster des Prototyps. Der Login ist zum Schutz vor unbefugtem Zugriff auf das Metaschema eingerichtet worden. Erst nach einem erfolgreichen Login lassen sich die Funktionen des Prototyps ausführen (siehe dazu Kapitel **Laufzeitphase** - Login bzw. **Anwendungsfall** - Login am X-Rayxs Prototyp).

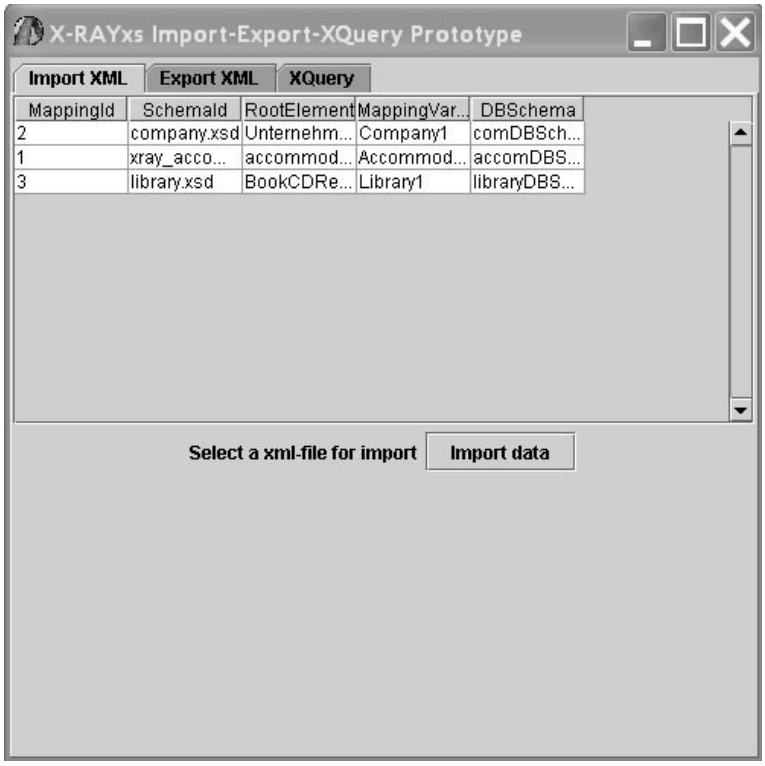


Abbildung 27 - Übersicht Prototyp X-Rayxs

Nachdem sich der Benutzer erfolgreich angemeldet hat, wird dem Benutzer die in Abbildung 27 dargestellte Tabelle angezeigt. Sie beinhaltet folgende Informationen:

- *MappingId* Identifiziert eindeutig eine Abbildungsvariante
- *SchemaId* Name der XSD-Datei (Schemadatei)
- *RootElement* Name des Wurzelelementes
- *MappingVariant* Bezeichnung für die Abbildungsvariante
- *DBSchema* Name des DB-Schemas, auf welche das jeweilige XML Schema abgebildet ist

Um eine klare Trennung der Funktionen herbeizuführen, wurde die Bedienung der Funktionen in drei unterschiedlichen Tabs, mit den Bezeichnungen *Import XML, Export XML* und *XQuery* realisiert. In jedem der drei Tabs findet sich die obige Tabelle wieder. Durch die Auswahl der Abbildungsvariante in der Tabelle gibt der Benutzer an, welche der aufgelisteten Varianten für den Import, Export oder der Abfrage mittels XQuery verwendet werden soll. Bei jedem Tab der drei Funktionen ist die Auswahl der Abbildungsvariante nötig.

Wird ein Import durchgeführt, wählt der Benutzer zusätzlich die zu importierende XML-Datei mit Hilfe eines Datei-Auswahl-Dialogs, der mit dem Button „Import data“ geöffnet wird (siehe dazu Kapitel **Laufzeitphase** - Import bzw. **Anwendungsfälle** - Import von Daten einer XML-Datei).

Äquivalent dazu wählt der Benutzer beim Export den Speicherort für die zu exportierende XML-Datei, und zusätzlich, ob die dazugehörigen Schema-Dateien an denselben Speicherort kopiert werden sollen (siehe dazu Kapitel **Laufzeitphase** – Export bzw. **Anwendungsfälle** - Export von Daten in eine XML-Datei).

Im dritten Tab (XQuery) kann der Benutzer durch die Eingabe einer XQuery-Anweisung in ein Textfeld Abfragen über bereits importierte Daten erstellen. Das Ergebnis der Abfrage kann in einer beliebigen Datei gespeichert werden, dessen Speicherort vom Benutzer angegeben wird (siehe dazu Kapitel **Laufzeitphase** – Abfragen mittels XQuery bzw. **Anwendungsfälle** - Erstellen von XQuery-Abfrage auf gespeicherte Daten).

5. Architektur

In diesem Abschnitt wird die Architektur des Prototyps vorgestellt. In Abbildung 28 werden die Pakete des Prototypes sowie ihre Beziehungen untereinander in Form eines UML-Paketdiagramms dargestellt.

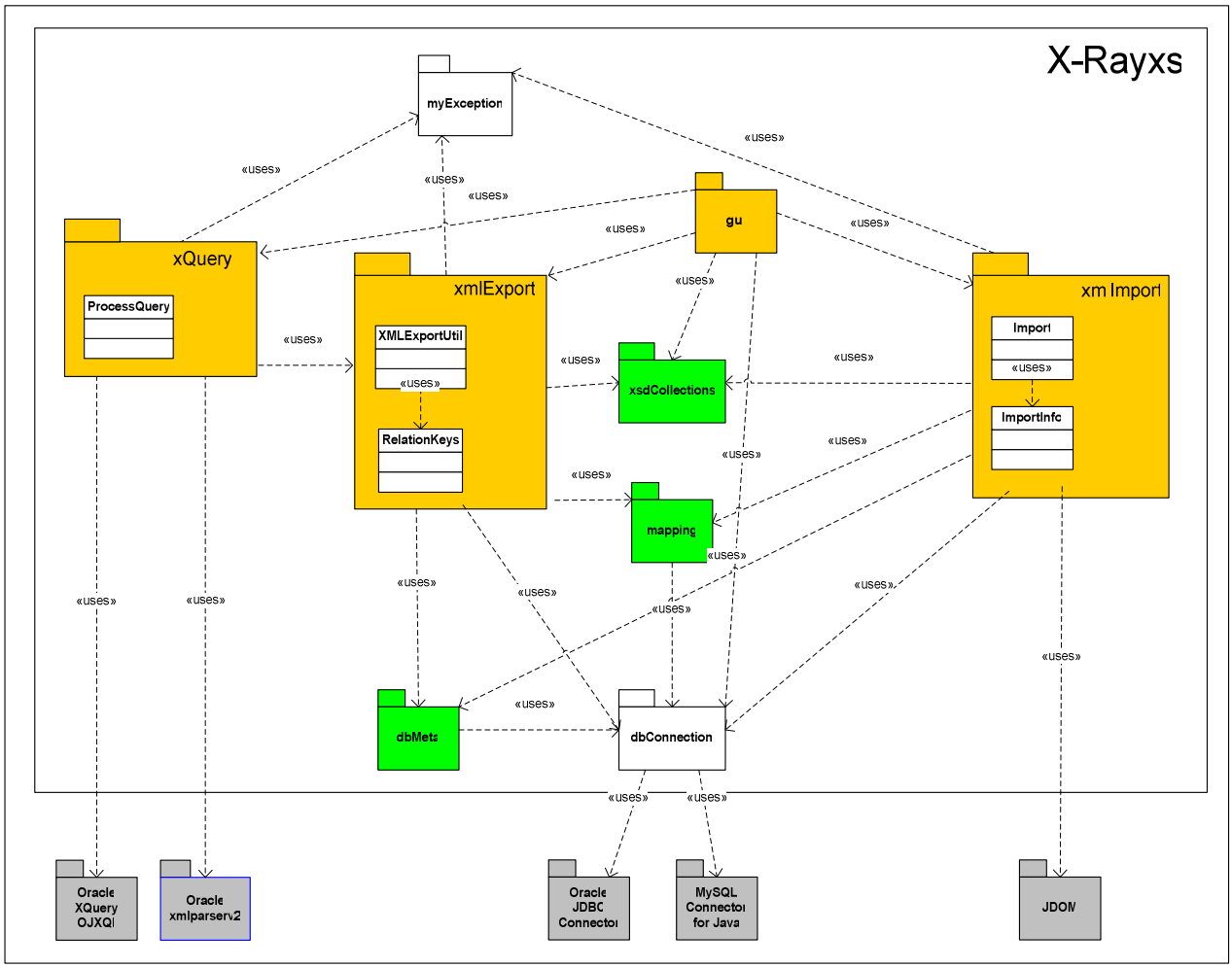


Abbildung 28 - Architektur Prototyp X-Rayxs

Die Java-Klassen des Prototyps wurden gemäß ihrer Funktionalitäten und Aufgaben in *Pakete* zusammengefasst. In Abbildung 28 sind die Pakete farblich gruppiert. Dadurch soll die Zusammengehörigkeit der Pakete verdeutlicht werden. So bilden die Pakete *xQuery*, *gui*, *xmlExport* und *xmlImport* dahingehend eine Gruppe, da diese die Verwendung des Metaschemas durch den Benutzer ermöglichen.

Die Pakete *xsdCollections*, *dbmeta* und *mapping* entsprechen den drei Komponenten des Metaschemas. Die Pakete *myException* und *dbConnection* stellen reine Hilfsklassen dar. Im unteren Bereich von Abbildung 28 befinden sich die externen Pakete (Oracle XQuery, Oracle xmlparserv2, Oracle JDBC Connector, MySQL Connector, JDOM), durch die sich der Implementierungsaufwand deutlich verringert. Im Anschluss werden die einzelnen Pakete näher vorgestellt.

5.1. Interne Pakete

gui

Die Klassen dieses Pakets sind für die Benutzerschnittstellen des Logins und für das Hauptfenster, in welchem die Funktionen des X-Rayxs Prototyps ausgeführt werden können, zuständig.

xmlExport

Die beiden Klassen dieses Pakets übernehmen den Aufbau und somit den Export einer XML-Datei entsprechend der Benutzereingaben. Die Klasse *RelationKeys* verwaltet die Schlüsselattribute und deren Werte (siehe **Laufzeitphase – Export**). Die Methoden der Klasse *XMLExportUtil* bauen die XML-Datei, durch Auslesen der Werte aus den Relationen gemäß der Abbildungsstrategie und Verwendung der Daten des Metaschemas, auf.

xQuery

Mit diesem Paket werden, unter Verwendung der externen Pakete *Oracle XQuery OJXQI* und *Oracle xmlparserv12,* XQuery-Abfragen, die im Hauptfenster des Prototyps (*Paket* gui) eingegeben werden können, auf XML-Dateien (temporär generiert) ausgeführt. Es wird dann das Resultat dem Textfeld des Hauptfensters übergeben. (siehe **Laufzeitphase – XQuery**)

xmlImport

Die Klasse Import baut unter Verwendung des externen Pakete *JDOM* eine DOM-Repräsentation einer zu importierenden XML-Datei auf. Mit Hilfe des DOM und den Daten des Metaschemas werden die Werte der XML-Datei in den entsprechenden Relationen gespeichert (siehe **Laufzeitphase – Import**)

xsdCollections

Dieses *Paket* entspricht der *XML Schema–Komponente* des X-Rayxs Metaschemas. (siehe **X-Rayxs – Metaschema**). Zu Beginn einer X-Rayxs-Session werden die Daten aus den Relationen der *XML Schema-Komponente* des Metaschema in die Instanzen dieser Klassen geladen. Diese Instanzen repräsentieren somit die verwalteten XML Schemata. Es gibt für jedes XML Schema-Konzept die entsprechenden Klassen. Nach Aufbau eines gespeicherten XML-Schemas (durch Selektion im Hauptfenster des Prototyps) stellen die Instanzen die objektorientierte Repräsentation dieses XML-Schemas dar.

mapping

Dieses *Paket* entspricht der *Mapping–Komponente* des X-Rayxs Metaschemas. (siehe **X-Rayxs – Metaschema**). Zu Beginn einer X-Rayxs-Session werden die Daten aus den Relationen der *Mapping–Komponente* des Metaschemas in die Instanzen der in diesem *Paket* enthaltenen Klassen geladen. Diese Instanzen repräsentieren dann das Abbildungswissen in objektorientierter Form.

dbmeta

Dieses *Paket* entspricht der *RDB Schema–Komponente* des X-Rayxs Metaschemas. (siehe **X-Rayxs – Metaschema**).

Zu Beginn einer X-Rayxs-Session werden die Daten aus den Relationen der *RDB Schema–Komponente* des Metaschema in die Instanzen der in diesem *Paket* enthaltenen Klassen geladen. Diese Instanzen enthalten somit die Metadaten jener Relationen, die als Zielrelationen für die Werte von importierten XML-Dateien bzw. Quellrelationen für die Werte der zu exportierenden XML-Dateien dienen.

dbconnection

Die Klassen dieses Pakets dienen zum Aufbau der Verbindungen zu den jeweiligen Datenbanken und der Datenbank des Metaschemas entsprechend der Daten in der Datei *properties* (siehe Anhang – Installationsanleitung).

myException

Die Klassen dienen dazu, um etwaige Fehler bei Ausführen einer gewählten Funktion des Prototyps dem Benutzer adäquat anzuzeigen.

5.2. Externe Pakete

Oracle XQuery OJXQI

Dieses Paket dient zum Durchführen von XQuerys auf eine XML-Datei. Die XML-Datei wird mittels der Klassen des Pakets *xmlparserv12* als spezieller Oracle DOM in den Arbeitsspeicher geladen um dann auf diesen die Abfrage durchzuführen.

Oracle xmlparserv12

Die Klassen dieses Pakets erzeugen einen spezifischen DOM einer XML-Datei, auf den dann unter Verwendung des *Pakets OJXQI* eine XQuery abgesetzt werden kann. Das Resultat der XQuery ist ein aus *XMLNodes* bestehendes *XQueryResultSet* (siehe Abbildung 43 - Abfragen mit XQuery). Dies wird anschließend durchlaufen und ausgegeben werden.

Oracle JDBC Connector (classes12.zip)

Dieses Paket beinhaltet die Treiberklassen um die Konnektivität zu einer Oracle - Datenbank zu realisieren.

MySQL Connector for Java

Dieses Paket beinhaltet die Treiberklassen um die Konnektivität zu einer MySQL - Datenbank zu realisieren.

JDOM

Die Klassen dieses Pakets dienen zum Erzeugen eines DOM für eine XML-Datei.
Mittels des erzeugten DOM wird der Import von XML-Dateien realisiert.

6. Laufzeitphase des Prototyps X-Rayxs

Wie bereits in Kapitel **X-Ray** erläutert, lässt sich die Verwendung von X-Rayxs in zwei Hauptphasen unterteilen, die Initialisierungsphase und die Laufzeitphase. Der im Zuge dieser Arbeit entwickelte Prototyp verfügt ausschließlich über die Funktionalitäten der Laufzeitphase (Import, Export, Abfragen mit XQuery). Da die Initialisierungsphase Voraussetzung für die Laufzeitphase ist, wird sie, für drei ausgewählte Beispiele, in den Installationsprozess eingebunden. Dadurch erst wird ermöglicht, dass aufbauend auf diesen Daten Importe, Exporte und Abfragen mittels XQuery durchgeführt werden können. Die gesamte Installationsanleitung für den Prototyp befindet sich im Anhang. Ziel des Prototyps ist es, die Funktionalität und Funktionsweise von X-Rayxs zu veranschaulichen und zu validieren. Im folgenden Kapitel wird der Ablauf und die Funktionsweise des Prototyp näher erläutert, wobei dies durch Beispiele unterstützt wird.

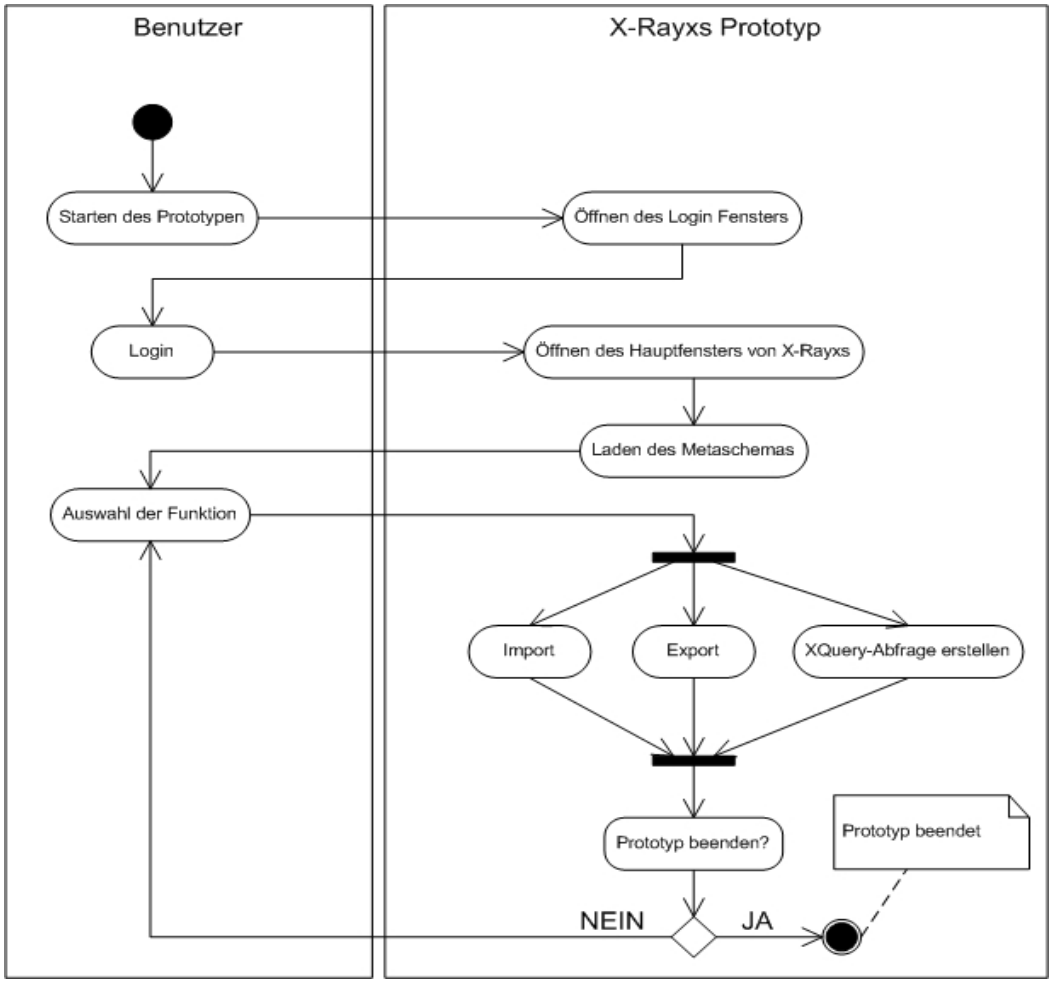


Abbildung 29 - Übersicht X-Rayxs Prototyp

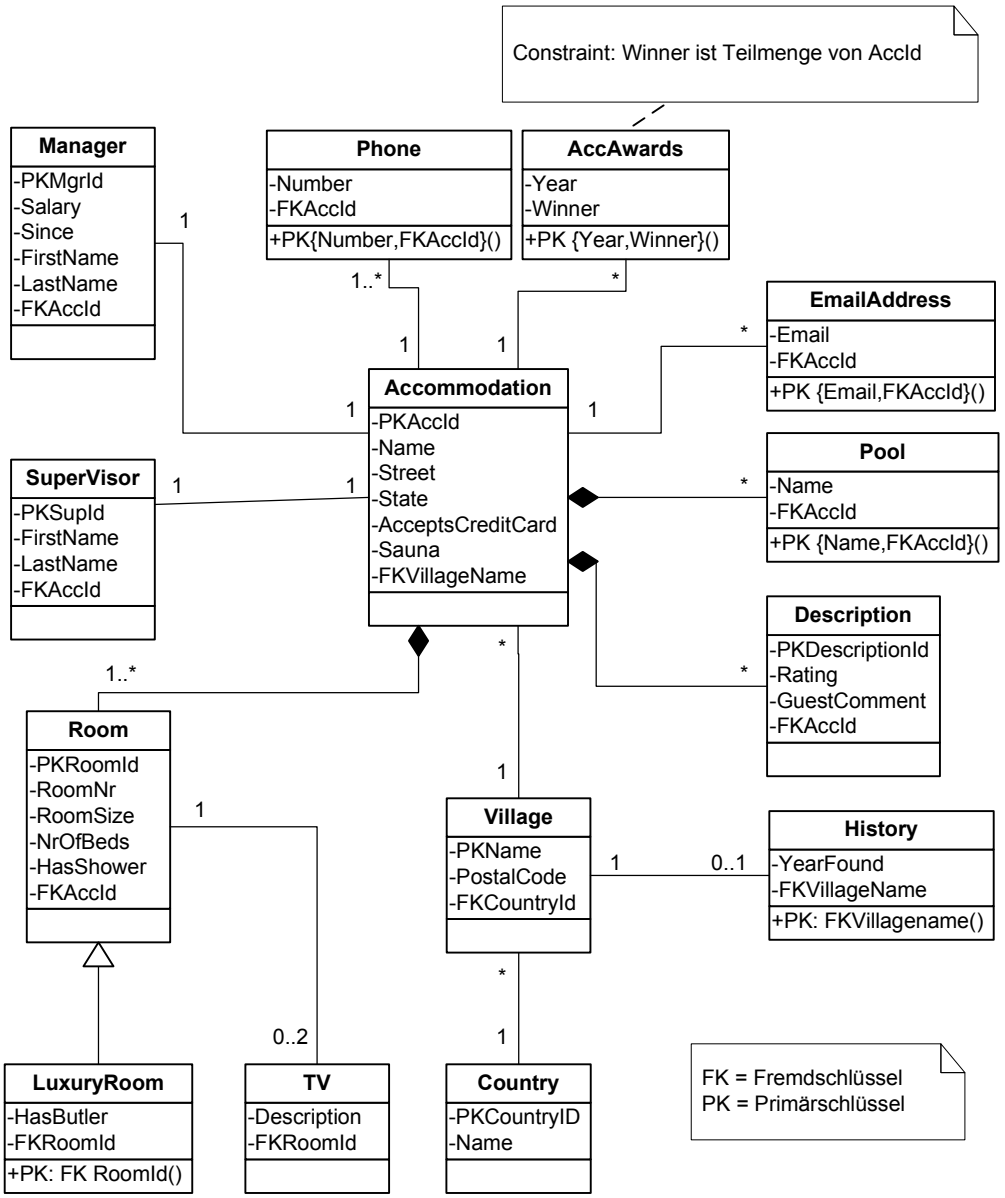


Abbildung 30 - UML-Diagramm des relationalen Schemas

Abbildung 29 zeigt einen Überblick über den Prototyp und die Funktionen der Laufzeitphase. Jede Funktion in dieser Abbildung wird in einem der folgenden Unterkapitel verfeinert und näher erläutert.

Ziel der Daten einer XML-Datei für den Import und zugleich Quelle der Daten einer XML-Datei für den Export bildet das in Abbildung 30 dargestellte RDB Schema.

6.1. Login

Um auf die Funktionen des X-Rayxs-Prototyps zugreifen zu können, muss sich der Benutzer mittels eines Logins am Prototyp anmelden.
Dieses Login hat zwei wesentliche Aufgaben:

1. Schutz vor unbefugtem Zugriff auf das Metaschema
2. Auslösen des Vorganges zum Laden des Metaschemas

Sämtliche Daten des Metaschemas sind in einer Datenbank gespeichert. Diese Datenbank ist vor unbefugtem Zugriff gesichert und der Benutzer erhält nur durch Eingabe eines korrekten Benutzernamens und Passwortes Zugriff auf das Metaschema.
Zusätzlich wird durch das erfolgreiche Anmelden das Laden des Metaschemas aus der Datenbank in die objektorientierte Repräsentation gestartet. Danach wird das Hauptfenster von X-Rayxs geöffnet, woraufhin der Benutzer auf die Funktionen des Prototyps, wie Import und Export von XML-Dokumenten, sowie das Erstellen von XQuery-Abfragen auf gespeicherte Daten, zugreifen kann.
Der Ablauf des Login-Vorganges ist in Abbildung 31 schematisch dargestellt.

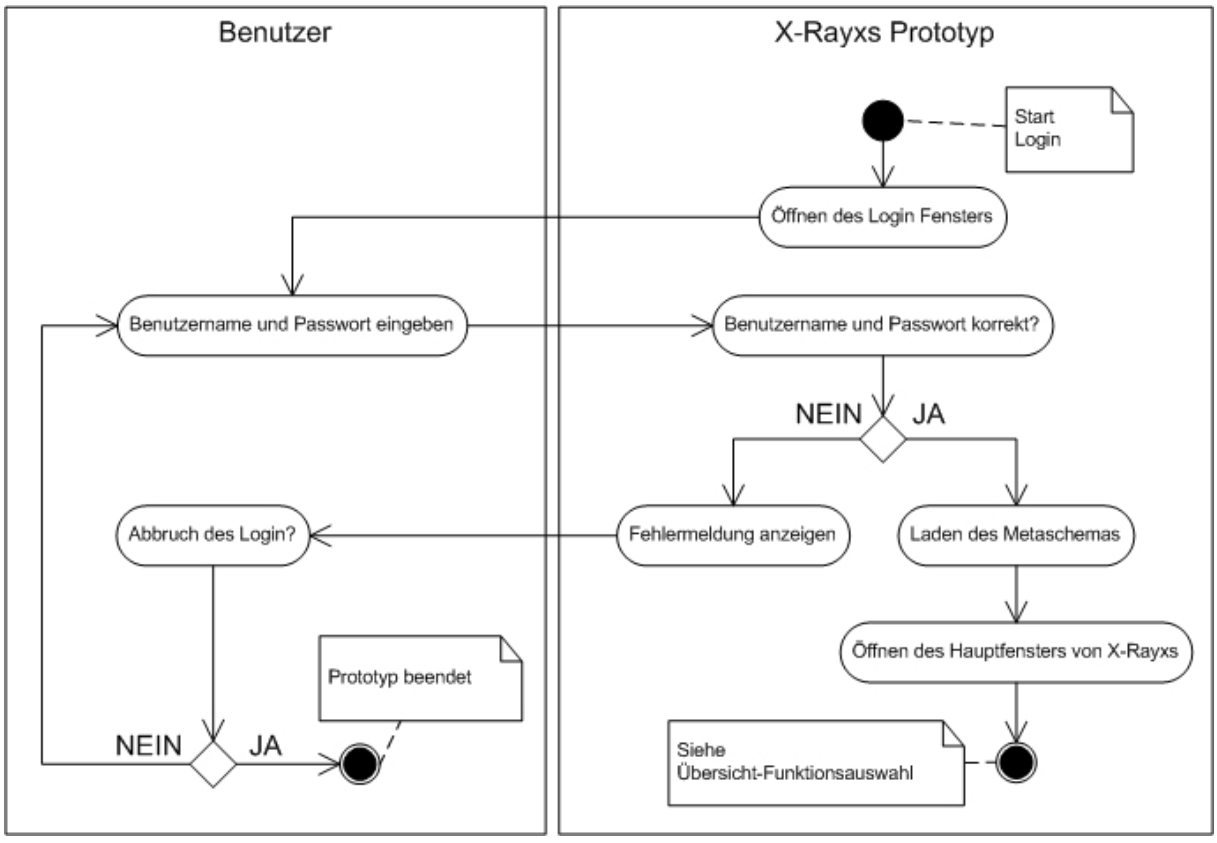


Abbildung 31 - Login X-Rayxs Prototyp

6.2. Laden des Metaschemas

Das gesamte Metaschema von X-Rayxs, wie es in Kapitel X-Rayxs der Diplomarbeit [KRAU05] beschrieben wurde, ist in einem RDBS persistent gespeichert. Da bei den Funktionen der Laufzeitphase (Export, XQuery-Abfragen, Import) die Daten des Metaschemas benötigt werden, und somit eine Vielzahl von Datenbankzugriffen nötig wären, wird eine objektorientierte Repräsentation des Metaschemas erstellt. Dieser nicht unbedingt nötige Aufwand begründet sich durch ein wesentlich verbessertes Laufzeitverhalten beim Ausführen der Funktionen des Prototyps. Diese objektorientierte Repräsentation wird direkt nach dem Login, wie zuvor erläutert wurde, erstellt. Nachteilig dabei ist jedoch die deutlich verlängerte Ladephase nach dem Login. Je nach Rechnerleistung und Verbindung zur Datenbank entsteht eine Wartezeit für den Benutzer zwischen dem Login und dem Öffnen des Hauptfensters.

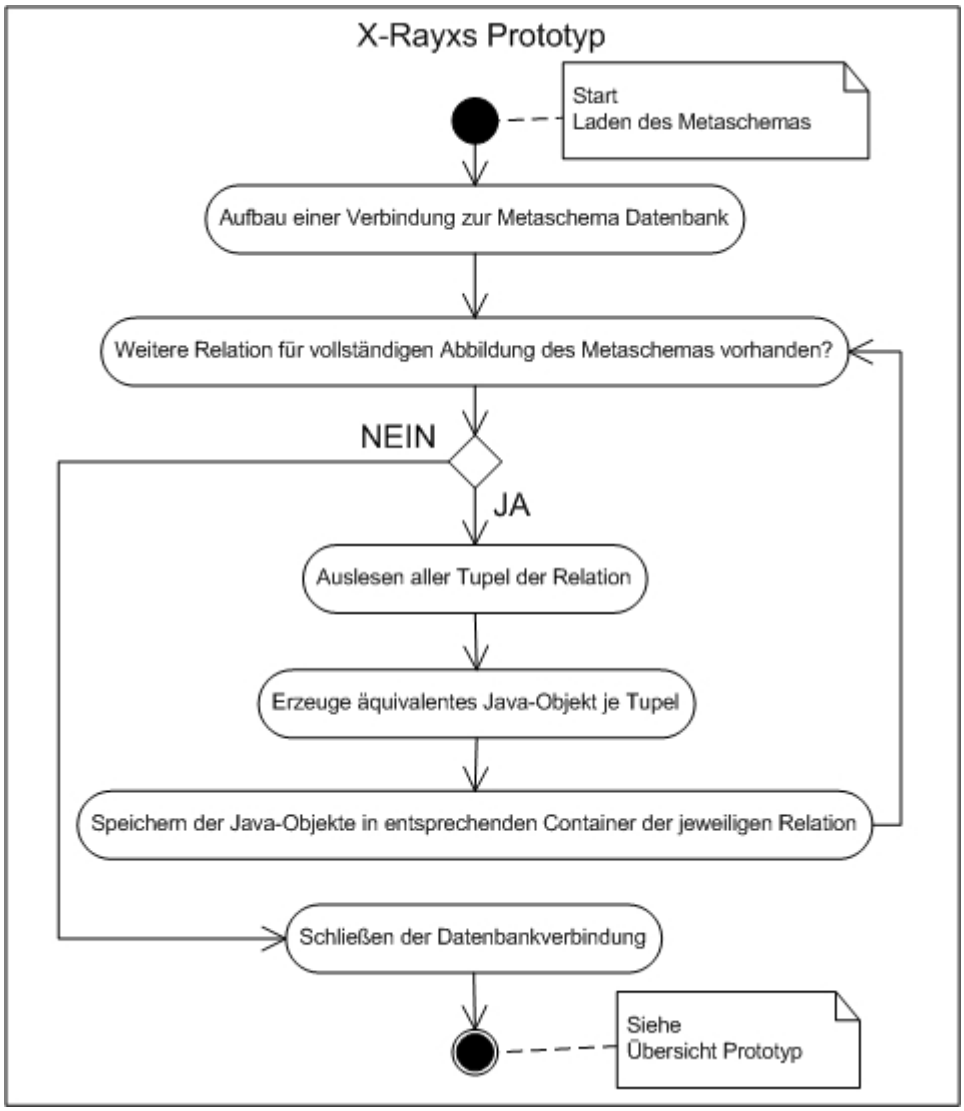


Abbildung 32 - Ladeprozess

Der Ladeprozess bezieht sich auf die *RDB Schema*-, *XML Schema*- und *Mapping-Komponente* des Metaschemas, das im Kapitel X-Rayxs [KRAU05] vorgestellt wurde. Abbildung 32 zeigt den Ablauf des Ladeprozesses. Im Prototyp wird dazu je Relation des Metaschemas ein Container erzeugt, der sämtliche Daten der Relation beinhaltet. Nachdem die Verbindung zum Datenbankserver, der das Metaschema beinhaltet, hergestellt wurde, wird jede Relation in ein eigenes „ResultSet“ ausgelesen. Jedes Tupel des jeweiligen „ResultSet“ wird anschließend in ein äquivalentes Java-Objekt übergeführt und im dafür erzeugten Container abgelegt. Wenn dies für jede Relation des Metaschemas ausgeführt worden ist, kann die Datenbankverbindung geschlossen werden. Für die *RDB Schema*- und *Mapping-Komponente* ist nach Abschluss des Ladevorganges das objektorientierte Abbild fertiggestellt.

Für die *XML Schema-Komponente* wurden nur die Daten und Beziehungsdaten geladen, die Beziehungen der Schemabestandteile untereinander jedoch noch nicht erstellt. Dies erfolgt erst nach Auswahl der Abbildungsvariante beim Durchführen einer der drei Hauptfunktionen des Prototyps. Diese geladenen Schemata bleiben dann für die Dauer der gesamten X-Rayxs-Session verfügbar.

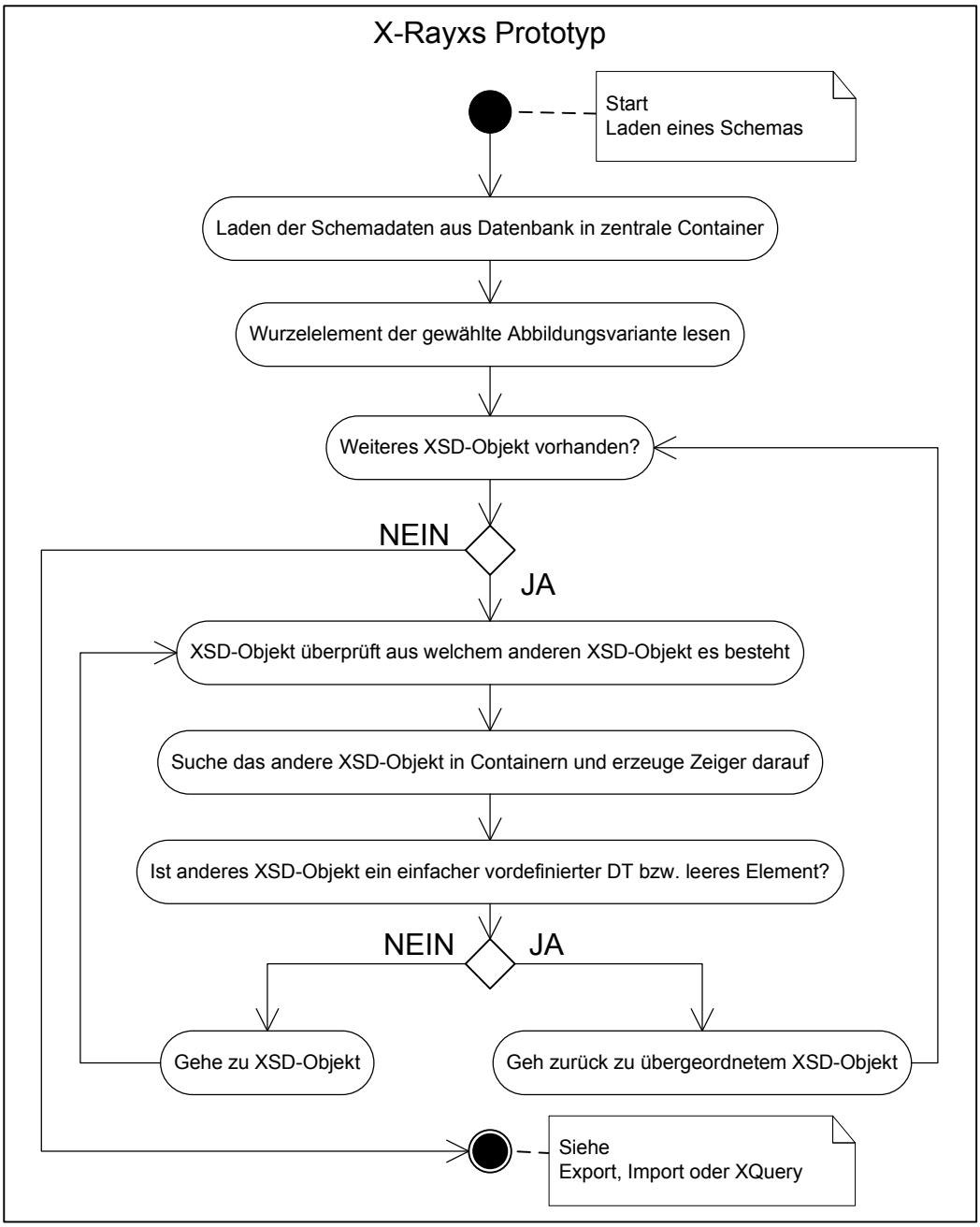


Abbildung 33 - Aufbau XML Schema

Abbildung 33 zeigt den Prozess zur Verknüpfung sämtlicher Bestandteile eines gespeicherten XML-Schemas. Diese Bestandteile eines XML-Schemas werden in Folge XSD-Objekte genannt und beziehen sich in diesem Kontext auf alle möglichen Elemente/Tags die in X-Rayxs auf Grund der in Abschnitt **Unterstützte Konzepte von XML Schema** gemachten Einschränkungen auftreten dürfen.
Demnach kann ein XSD-Objekt u.a. Folgendes sein:
<element>, *<sequence>, <complexType>, <attribute>*, etc.

Sowohl die zu verbindenden XSD-Objekte, als auch die möglichen und notwendigen Verknüpfungen selbst, können aus dem UML-Diagramm (siehe Anhang) für XML Schema abgeleitet werden. Somit können die XSD-Objekte und deren Beziehungen zueinander, wie sie in der Datenbank bereits bestanden haben, wiederhergestellt werden. Beispielsweise wird das XSD-Objekt *<element name="accommodations">* mit dem im XML-Schema lokal deklarierten Datentyp *<complexType>*, der wiederum aus einer *<sequence>* besteht usw., verknüpft (siehe dazu Anhang).
Dieser Prozess wird ausgeführt, nachdem die Daten des Metaschemas geladen wurden (siehe Abbildung 32) und der Benutzer ausgewählt hat, mit welcher Abbildungsvariante ein Import oder Export durchgeführt werden soll. Dies hat den Vorteil, dass nur jene XSD-Objekte miteinander verknüpft werden müssen, die für das Ausführen der jeweiligen Funktion benötigt werden. Dieser Verknüpfungsprozess vereinfacht die Navigation zwischen den XSD-Objekten beim Ausführen der vom Benutzer gewählten Funktion erheblich.
Die Beziehungen zwischen den Java-Objekten, die durch den Ladeprozess erzeugt wurden und XSD-Objekte repräsentieren, werden mit Hilfe von Zeigern realisiert. Um dies in Java umzusetzen, besitzt jedes dieser Java-Objekte, das zur *XML Schema-Komponente* gehört, die Methode *build()*. Wie aus Abbildung 33 ersichtlich, wird das Wurzelelement des XML-Schemas ermittelt, und durch den Aufruf der *build()*-Methode dieses Elements der Verknüpfungsprozess gestartet. Das Wurzelelement ist deshalb der Ausgangspunkt, weil von hier aus alle weiteren, zum XML Schema gehörenden XSD-Objekte erreicht werden können. Solange das durch die *build()*-Methode erreichte XSD-Objekt kein einfacher vordefinierter Datentyp oder kein leeres Element ist, wird vom Vorgänger-XSD-Objekt die *build()*-Methode des jeweiligen Nachfolgeobjekts aufgerufen. Es sucht sich somit jedes XSD-Objekt sein unmittelbares Kindobjekt selbständig durch die *build()*-Methode und richtet einen Zeiger auf dieses. Dieser Vorgang setzt sich so lange fort, bis alle XSD-Objekte behandelt wurden und ein XML-Schema fertig geladen wurde.

6.3. Export

Dieser Abschnitt beschreibt, wie ein XML-Dokument mit Hilfe von Schemadaten und Werten aus einer Datenbank erstellt wird. Abbildung 34 stellt den groben Ablauf des Exports dar.

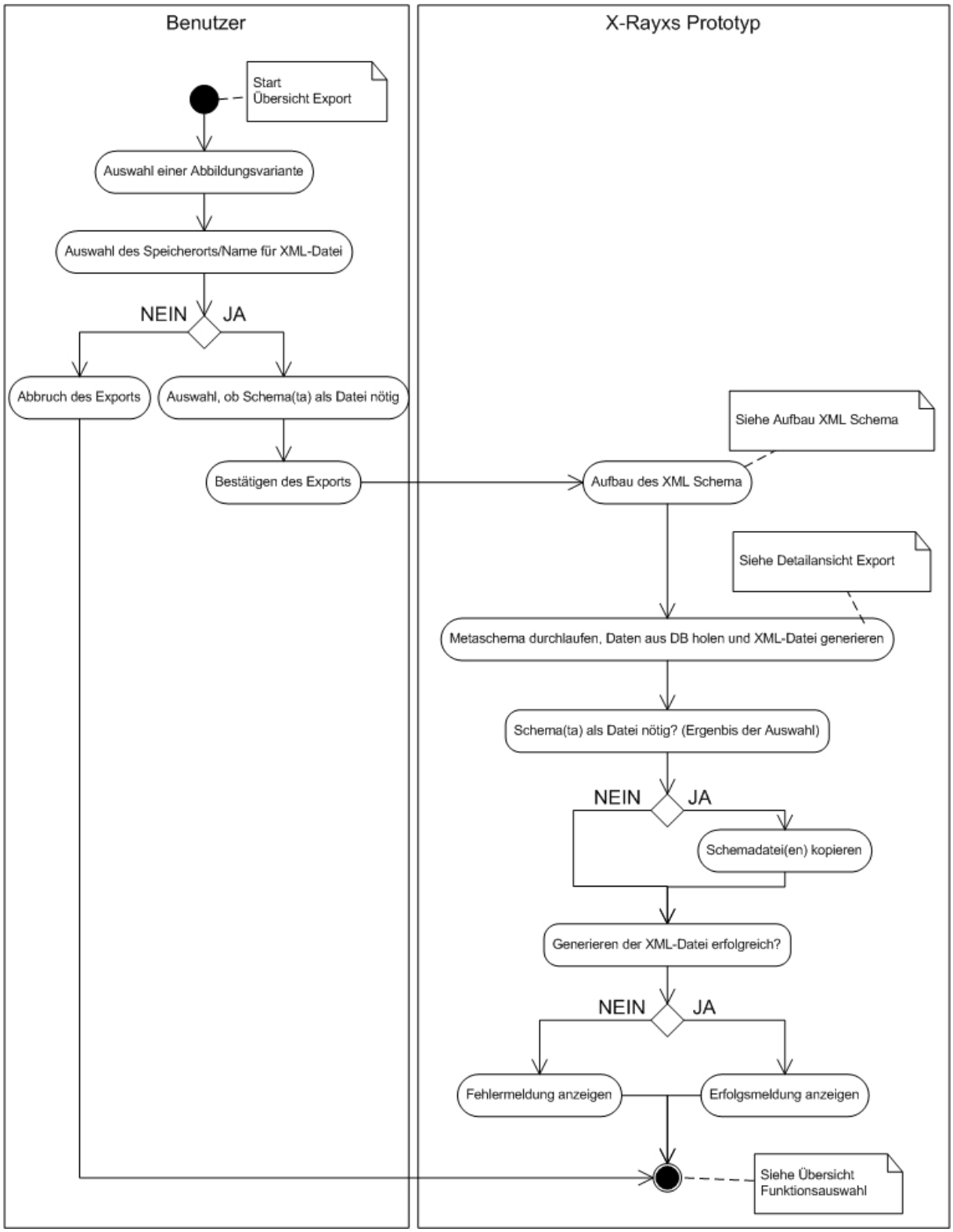


Abbildung 34 - Übersicht Export

Nach dem Login sind die Relationen des Metaschemas in eine objektorientierte Repräsentation übergeführt. Nachdem die Metaschemadaten geladen sind, werden die Abbildungsvarianten aus der Relation *Mappings* (siehe **X-Rayxs Metaschema Mapping-Komponente**) im Hauptfenster des Prototyps dargestellt. Nach Selektion einer Abbildungsvariante sowie der Auswahl des Speicherortes und Dateinamens für die zu exportierende XML Schema Instanz (XML-Dokument), wird der Aufbau der objektorientierten Schemastruktur mittels Instanzen entsprechender Klassen und Zeigern zwischen den Instanzen gestartet („Aufbau des XML Schemas“ in Abbildung 34).

Die Auswahl der zu exportierenden Abbildungsvariante stellt sich wie in Abbildung 35 dar. Der genaue Ablauf des Exportes aus Sicht des Benutzers kann im Anhang - Benutzerhandbuch nachgelesen werden.

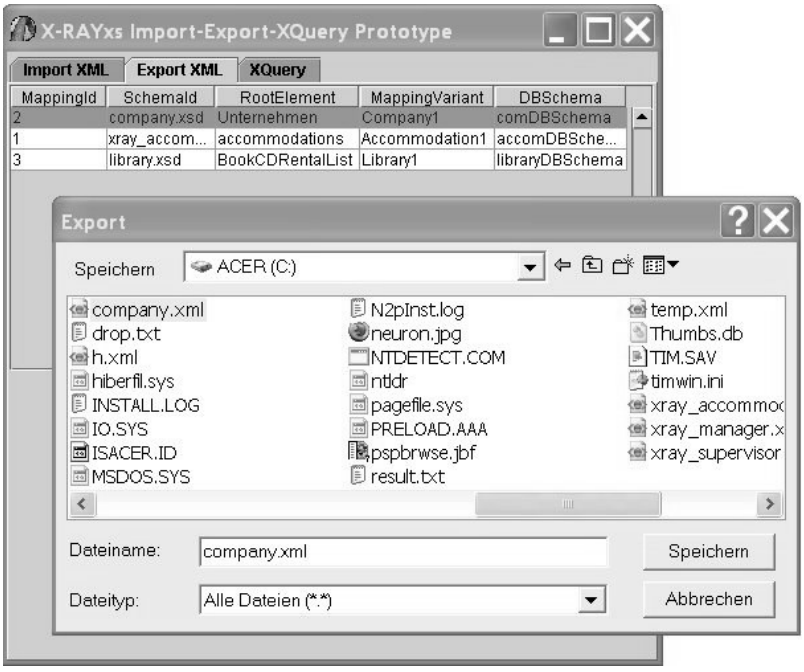


Abbildung 35 - Export GUI des X-Rayxs Prototyps

6.4. Export-Algorithmus

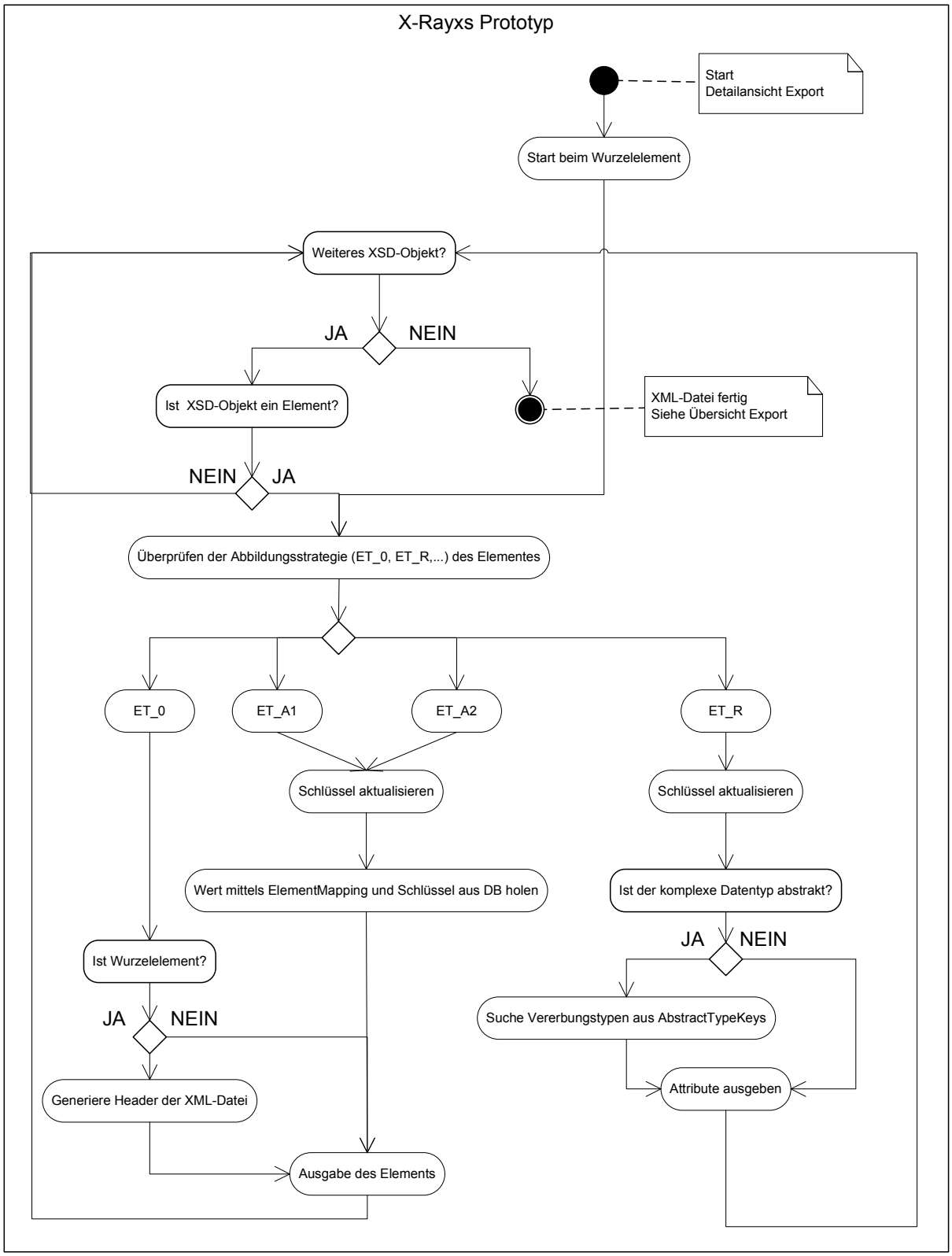


Abbildung 36 - Detailansicht Export

Den Einstiegspunkt in ein Schema bildet das Wurzelelement, von welchem aus, über Zeiger, jedes einzelne Element des Schemas der gewählten Abbildungsvariante erreichbar ist. Für den Aufbau der XML-Datei wird mit dem Wurzelelement und dem *Header,* wie in Abbildung 37 erkennbar, begonnen. Der *Header,* welcher sich aus Attributen des Wurzelelementes zusammensetzt, enthält u.a. Informationen über Namensräume (*xmlns*).

```
<accommodations
xmlns=http://www.ifs.uni-linz.ac.at/XRay/AccommodationSchema
xmlns:su="http://www.ifs.uni-linz.ac.at/XRay/Supervision"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ifs.uni-
linz.ac.at/XRay/AccommodationSchema xray_accommodation.xsd">
   <accommodation id="45" state="Austria">
   …
   </accommodation>
<accommodation id="46" state="Austria">
   …
   </accommodation>
</accommodations>
```

Abbildung 37 - Header von Accommodation.xml

In Abbildung 33 ist der detaillierte Ablauf zum Aufbau einer XML-Datei dargestellt. Es wird also die objektorientierte Repräsentation des XML Schemas durchlaufen. Beim Erreichen eines Elements wird mit Hilfe der Daten aus der Relation *ElementMapping* überprüft, welche Abbildungsstrategie (ET_0, ET_R etc) für das Element vorgesehen ist. Die Art der Ausgabe eines Elementes ist abhängig von der Abbildungsstrategie. Eine Datenrelation stellt eine Relation dar, in welcher Werte für eine oder mehrere XML-Datei(en) gespeichert sind. Die Relation *Accommodation* ist ein Beispiel für eine Datenrelation.

Die Werte für die Elemente der XML-Datei müssen sukzessiv aus den Relationen der Datenbank gelesen werden. Die Verwaltung von Schlüsselwerten für die Schlüsselattribute von Datenrelationen ist daher absolut notwendig, um die Werte im korrekten Kontext zu erhalten. Die Schlüssel sind notwendig, um das korrekten Tupel einer Datenrelation zu selektieren und für aktuelle Instanz eines Elementes in Bezug auf dessen Kontext zu erhalten. Um dies zu realisieren wurde eine Klasse eingeführt, mittels deren Instanzen für jede Datenrelation der Relationsname, das Schlüsselattribut und der aktuelle Schlüsselwert gespeichert werden.

Diese Klasse *RelationKeys* im Paket *xmlExport* hat vier Attribute

- *relationName*: Name der Datenrelation
- *keyAttribute*: Name des Schlüsselattributes
- *keyValue*: Liste mit den verwendeten Schlüsselwerten (der erste in der Liste ist immer der aktuelle Wert)
- *finished*: Markiert, ob schon alle Schlüsselwerte für eine Datenrelation verwendet worden sind.

Der Aufbau der XML-Datei wird von der Klasse *XMLExportUtil* im Paket *xmlExport* durchgeführt. Diese Klasse ist verantwortlich für das Auslesen der Werte aus den Datenrelationen, dem Verwalten der Schlüssel und Schlüsselwerte und der Ausgabe der Elemente entsprechend ihrer Struktur und Abbildungsstrategie. Diese Klasse enthält u.a. das Attribut *DBKeys*, welches einen Container darstellt, in dem alle Instanzen der Klasse *RelationKeys* und somit die Schlüssel samt Werten für alle notwendigen Datenrelationen, gespeichert sind.

Wird beim Durchlauf der Schemastruktur ein Element erreicht, eruiert man unter Verwendung der *XSDElementId* und der *MappingId* in der Klasse *ElementMapping*, welche Abbildungsstrategie für dieses Element vorliegt.

ElementMapping				
XSDElementID	KindOfMapping	DBAttr	DBRelShipId	MappingId
33	ET_0	NULL	NULL	1
28	ET_R	NULL	NULL	1
2	ET_A1	2	NULL	1
6	ET_0	NULL	NULL	1
3	ET_A1	3	NULL	1
4	ET_A2	28	1	1

Tabelle 1 - Relation und Klasse ElementMapping

Je nach Abbildungsstrategie für ein Element wird unterschiedlich verfahren.

6.4.1. ET_0

Es müssen keine Werte für das Element selbst aus der Datenbank gelesen werden. Dies ist z.B. bei Elementen mit fixem Wert der Fall. Es kann sich jedoch auch um ein leeres Element mit Kardinalität 1 und Attributen handeln. In diesem Fall müssen die Werte für die Attribute mittels der Daten aus den Relationen *AttributeMapping*, *DBAttribute* und, bei Bedarf ($A\_A_{indirekt}$), *DBJoinSegment* aus den jeweiligen Datenrelationen geholt werden. Einen Sonderfall stellt das Wurzelelement dar, da bei diesem, wie eingangs beschrieben, der *Header* anstatt von Attributen ausgegeben wird.

6.4.2. ET_R$_{direkt/indirekt}$

Der Unterschied zwischen direktem und indirektem Abbilden wird, wie im Abschnitt **X-Rayxs Metaschema** beschrieben, durch den Wert des Attribut *DBRelShipId* der Klasse *ElementMapping* realisiert. Es wird zwischen drei Arten der Verarbeitung von ET_R unterschieden.

Direktes Abbilden:

1. Basisrelation - Das Attribut *DBRelshipId* ist *null* und man erhält mittels der Daten der Klasse *ElementBaseRelation* Zugriff auf die Basisrelation. Der Name der Basisrelation wird in der Klasse *RelationKeys* gespeichert (*<accommodation>* in Abbildung 38)

Indirektes Abbilden:

2. Leeres Element mit Kardinalität + oder * und Attributen (Abbildung 38 *<phone>*)
3. Element mit komplexem Inhalt (Abbildung 38 *<address>*)

```
<accommodations ….>
   <accommodation id="45" state="Austria">
      <address postalCode="4020">
      …
      </address>
      <phone number="+4373278990"/>
      …
   </accommodation>
   …
</accommodations>
```

Abbildung 38 - Beispiele für mit ET_R abgebildete Elemente

Zuerst wird das Schlüsselattribut der Zielrelation mitsamt des ersten Schlüsselwertes gesucht und in der Instanz der Klasse *RelationKeys* aktualisiert. Das Schlüsselattribut für eine Relation findet man durch Überprüfung des Attributes *IsKey* der Relation *DBAttribute*.
Danach wird ein Schlüsselwert nach dem anderen gelesen. Der aktuelle Schlüsselwert dient zum Selektieren der Tupel mit den Werten für die untergeordneten Elemente und Attribute des mit ET_R abgebildeten Elementes. Mit Hilfe der Schlüsselwerte und den Daten der Klasse *DBJoinSegment* ist es möglich von einer Datenrelation zu weiteren Datenrelationen zu navigieren. Beim Aufbau der XML-Datei wird der hierarchische Aufbau eines XML-Dokumentes ausgenützt. Dadurch ist es möglich, durch Kenntnis der Werte eines Elementes die korrekten Werte für dessen mögliche Subelemente aus der Datenbank zu lesen.

6.4.3. ET_A$_{direkt}$

Bei direktem Abbilden eines Elementes auf ein Attribut einer Basisrelation ist ET_A1 in der Klasse *ElementBaseRelation* im Attribut *KindOfMapping* zu finden. Die Datenrelation und das Zielattribut werden mittels des Attributes *DBAttr* der Klasse *ElementMapping* und der Klasse *DBAttribute* gefunden. Um den korrekten Wert für das Element zu allokieren, wird mittels des Schlüsselattributes und dem aktuellen Schlüsselwert, beides in der Klasse *RelationKeys* gespeichert, das entsprechende Tupel der Zielrelation selektiert.

```
<accommodations ….>
  <accommodation id="45" state="Austria">
    <name>Steigenberger MAXX Hotel Linz</name>
    …
  </accommodation>
  …
</accommodations>
```

Abbildung 39 - Beispiel für mit ET_A1 abgebildetes Element

Accommodation						
AccId	Name	Street	State	*VillageName*	AcceptsCreditCards	Sauna
45	Steigenberger MAXX Hotel Linz	Am Winterhafen 13	Austria	Linz	true	false
46	Hotel Wolf-Dietrich	Wolf-Dietrich-Straße 7	Austria	Salzburg	true	true

Tabelle 2 - Datenrelation Accommodation

Das Zusammenspiel zwischen den Relationen bzw. Klassen des Metaschemas wird im Abschnitt **Metaschema** des Kapitels **X-Rayxs** [KRAU05] beschrieben.

6.4.4. ET_A$_{indirekt}$

Indirektes Abbilden liegt vor, wenn die Werte für/von Elementen aufgrund von Normalisierung von Datenrelationen in eigene Relationen ausgelagert werden.
In diesem Fall enthält das Attribut *KindOfMapping* der Klasse *ElementBaseRelation* den Wert ET_A2 für das abgebildete Element.
Es sind auch bei ET_A2 verschiedene Fälle zu unterscheiden.

1. Leeres Element mit Kardinalität + oder *, jedoch ohne Attribute (z.B. <*pool*>)
2. Atomares Element mit Kardinalität + oder * (z.B. <*email*>)
3. Element, abgeleitet auf ausgelagerte Relationen durch Normalisierung (z.B. <*village*>)

Es wird mit Hilfe des Schlüsselattributes, dem Schlüsselwert und den Daten aus der Relation bzw. Klasse *DBJoinSegment* das korrekte Tupel selektiert und der Wert des entsprechenden Attributes ausgelesen.

Beispiel leeres Element *<pool>:*

```
<accommodations ….>
  <accommodation id="45" state="Austria">
  …
    <address postalCode="4020">
      …
      <village yearOfFoundation="1600">Linz</village>
    …
    </address>
    …
    <phone number="+4373278990"/>
    …
    <pool/>
    …
  </accommodation>
  <accommodation id="46" state="Austria">
    …
    <phone number="0043 662 871275"/>
    …
    <pool/>
    <pool/>

    …
  </accommodation>
</accommodations>
```

Abbildung 40 - Beispiele für ET_A2 abgebildete Elemente

In diesem Fall ist nur die Anzahl der Tupel wichtig. Gemäß der Anzahl selektierter Tupel werden entsprechend viele leere Elemente ausgegeben.

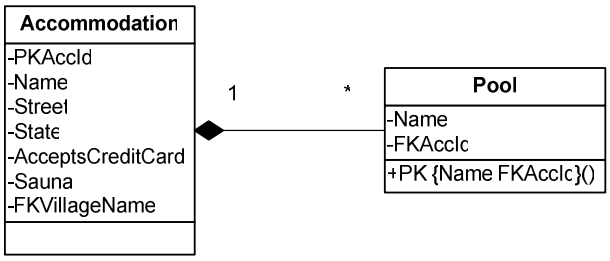


Abbildung 41 - UML-Diagramm der Relation Accommodation und Pool

Da es sich bei *<pool>* um ein leeres Elemente handelt, werden keine realen Werte in die Relation *Pool* geschrieben. Bei leeren Elementen, wenn sie zusätzlich keine Attribute enthalten, mit Kardinalität * oder + ist letztendlich nur die Anzahl der Tupel interessant.

Da jedoch in der Relation *Pool* der Fremdschlüssel allein zugleich Primärschlüssel wäre, musste in diesem Fall das Attribut *Name* der Relation *Pool* hinzugefügt werden. Anders wäre es unmöglich, mehrere Pools für eine *<accommodation>*-Instanz zu speichern. Das Attribut *Name* enthält daher nur, wie in Tabelle 3 ersichtlich, rein fiktive Surrogatnamen, die beim Auslesen der Relation unberücksichtigt bleiben.

Pool	
AccId	Name
45	Pool1
46	Pool1
46	Pool2

Tabelle 3 - Relation Pool

Die Relationen der *RDB Schema–Komponente* enthalten somit folgende Tupel.

DBAttribute					
DBAttId	*DBSchema*	DBRelation	DBAttribute	DBIsKey	DBDataType
35	accomDBSchema	Pool	AccId	true	INT
36	accomDBSchema	Pool	Name	true	VARCHAR(50)

Tabelle 4 - Relation DBAttribute

Die Relation *ElementMapping* enthält folgendes Tupel

ElementMapping				
XSDElementID	KindOfMapping	DBAttr	DBRelShipId	MappingId
20	ET_A2	36	5	1

Tabelle 5 - Relation ElementMapping

DBJoinSegment						
DBRelShipId	DBAtt1	DBAtt2	DBRelation1	DBRelation2	*DBSchema*	DBJoinDirection
5	AccId	AccId	Pool	Accommodation	accomDBSchema	21

Tabelle 6 - Relation DBJoinSegment

Unter Verwendung aller obigen Informationen ist das Resultat eine XML-Datei wie in Abbildung 40 ausschnittsweise dargestellt.

6.4.5. A_A$_{direkt/indirekt}$

Handelt es sich um direktes Abbilden eines XML-Attributes enthält das Attribut *KindOfMapping* der Relation *AttributeMapping* den Wert A_A1. Hat ein XML-Attribut einen fixen Wert (z.B. *state* in Element <*accommodation*>), ist der Wert im Metaschema gespeichert und *KindOfMapping* hat den Wert A_0. Ist das Element, zu dem ein Attribut gehört ET_A2 (ET_A$_{indirekt}$), abgebildet, so sind auch dessen Attribute A_A2 (A_A$_{indirekt}$) abgebildet. Die Werte der Attribute, egal ob A_A1 oder A_A2, werden unter Verwendung des beschrieben Schlüsselkonzepts (Verwendung von *RelationKeys*) aus den jeweiligen Datenrelationen gelesen und die Attribute entsprechend bei ihrem Element ausgegeben. Zusätzlich muss jedoch beachtet werden, zu welchem Element ein Attribut gehört. Der Grund ist, dass Attribute bei ihrer Deklaration einem komplexen Datentyp zugewiesen sind. Unerheblich ist, ob das Attribut lokal oder global definiert ist. Es ist nicht unüblich, dass zwei Elemente unterschiedlichen Namens den gleichen komplexen Datentyp (mit Attributen) als Basis haben. Es werden aber nur die Elemente und nicht die Datentypen abgebildet. Somit ist auch das Abbilden der Attribute abhängig von dem Element (*XSDElementID* in Tabelle 7), zu dem sie letztendlich gehören.

AttributeMapping					
XSDElementID	XSDAttributeID	KindOfMapping	DBAttr	DBRelShipId	MappingId
28	7	A_A1	1	NULL	1
28	8	A_0	NULL	NULL	1
6	2	A_A2	29	1	1

Tabelle 7 - Beispiel für AttributeMapping

6.4.6. Elemente mit abstrakten komplexen Datentypen

Wie mit abstrakten komplexen Datentypen und damit einhergehenden, Vererbung von Datentypen im Metaschema, verfahren wird, ist im Abschnitt **Metaschema des** Kapitels **X-Rayxs** [KRAU05] beschrieben. Beim Export erhält das Element, welches als Basis einen abstrakten komplexen Datentyp hat, zusätzlich das Attribute *xsi:type*, wie in Abbildung 42 ersichtlich.

```
<room xsi:type='standardRoomType'>
      <roomNumber>101</roomNumber>
      <size>25</size>
      <nrOfBeds>2</nrOfBeds>
      <tv>Panasonic TX-32PS11D</tv>
      <shower/>
</room>

<room xsi:type='luxuryRoomType' hasButler='true'>
      <roomNumber>305</roomNumber>
      <size>95</size>
      <nrOfBeds>3</nrOfBeds>
      <tv>Sony KV-32FQ85</tv>
</room>
```

Abbildung 42 - Vererbung von abstrakten Datentypen

Der Wert für dieses Attribut muss aus der Relation bzw. Klasse *AbstractTypeKeys* geholt werden.

AbstractTypeKeys			
DBRelation	DBKeyAttr	DBKeyValue	HeritageType
Room	RoomId	1	standardRoomType
Room	RoomId	2	luxuryRoomType
Room	RoomId	3	standardRoomType

Tabelle 8 - Relation und Klasse zum Speichern von Vererbungsdaten

Ein kleines Beispiel soll dies verdeutlichen:
Der erste Raum (*roomNumber*=101) in Abbildung 42 hat beim Import in die Relation *Room* die *RoomId*=1 (Surrogat) zugewiesen bekommen. Danach wird in der Relation *AbstractTypeKeys* gespeichert, dass die *RoomId*=1 dem *xsi:type* - Attribute entsprechend den Wert *standardRoomType* hat.

Dies muss beim Import eines XML-Dokumentes für jedes Element, welches das XML Schema-spezifische Attribut *xsi:type* besitzt, durchgeführt werden. Deshalb muss nach jedem Import bzw. vor jedem Export diese Relation erneut in Instanzen der objektorientierten Klasse ausgelesen werden.
Das Auslesen der Werte für solche Elemente erfolgt entsprechend der zuvor beschriebenen Abbildungsstrategien.

6.5. Abfragen mittels XQuery

Diese Funktion des Prototyps ermöglicht es, Abfragen in XQuery [W3C05] auf gespeicherte Daten abzusetzen. Abbildung 43 stellt den Ablauf der Funktion dar. Als unterstützte Abfragesprache für mit X-Rayxs gespeicherte Daten wurde XQuery gewählt, welche als Basis XPath2.0 [W3C00], Quilt [QUIL00] etc. hat und laut W3C als Abfragesprache für XML Daten zum Standard erhoben werden soll.
XQuery hat u.a. den Vorteil, die Datentypen von XML Schema zu unterstützen. Da die Spezifikation von XQuery noch nicht gänzlich abgeschlossen ist, gibt es kaum Unterstützung (Bibliotheken) für Java. Für den X-Rayxs Prototypen fiel die Wahl auf OJXQI [ORAC05]. Oracle stellt dieses Java API für XQuery zur Verfügung. Ein großer Vorteil ist die einfache Handhabung von OJXQI sowie die volle Unterstützung von XQuery. Einer Methode (vgl. Abbildung 44Abbildung 43) des API's übergibt man ein XML-Dokument und eine gültige XQuery (*PreparedXQuery*), eingelesen von einem Textfeld des Prototyps, und erhält als Resultat ein *XQueryResultSet*, das man durchlaufen und z.B. wiederum in einem Textfeld oder einer Datei ausgeben kann. Abbildung 44 zeigt die Verwendung von OJXQI im Rahmen des Prototyps. Ein Nachteil ist, dass als Datenquelle nur eine Datenbank oder eine XML-Datei verwendet werden kann. Erstellt der Anwender eine XQuery, muss eine XML-Datei gemäß der selektierten Abbildungsvariante (siehe Abschnitt **Export**) temporär erstellt werden, die dann als Ziel für die XQuery verwendet wird. Da die XML-Dateien in einem Systemordner für temporäre Dateien abgelegt werden, können sie vom System nach Verwendung gelöscht werden.

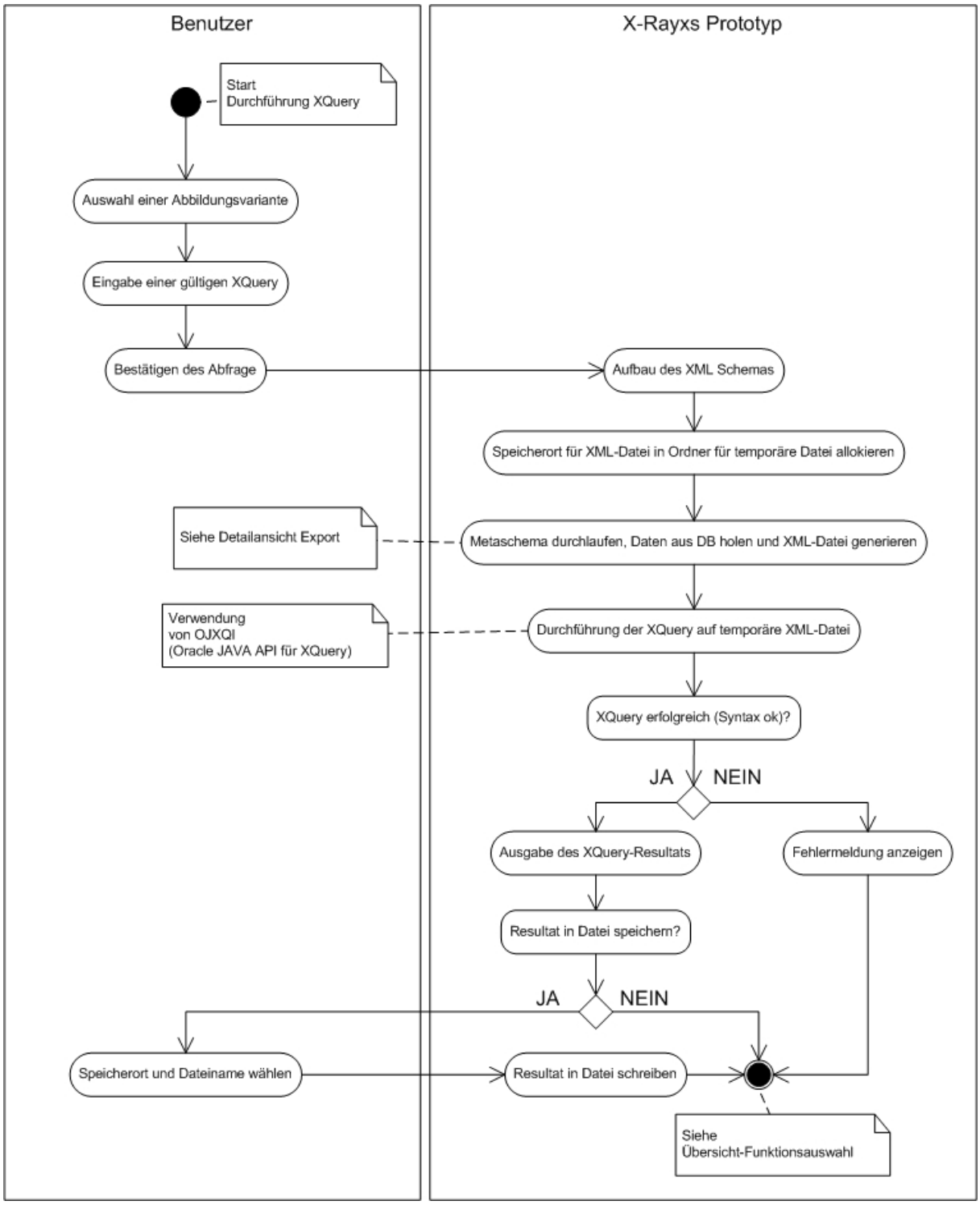


Abbildung 43 - Abfragen mit XQuery

```
public class ProcessXQuery{
      public void doTheQuery(String filename,String query,
JTextArea result){
           try{
                XQueryContext ctx = new XQueryContext();
                PreparedXQuery xq = ctx.prepareXQuery(query);
           XQueryResultSet rset = xq.executeQuery(true);

           PrintStream exportStream = new PrintStream(new
PipedOutputStream());
           //Durchlaufen des Resultats der XQuery
           while (rset.next()){
                     XMLNode node = rset.getNode();
                     node.print(exportStream);
           }
           }
           catch(Exception e){}
      }
}
```

Abbildung 44 - Anwendung von OJXQI

Das Resultat der XQuery wird in einem Textfeld des Prototyps angezeigt und kann vom Anwender bei Bedarf in eine beliebige Datei exportiert werden. Siehe dazu Abbildung 45.

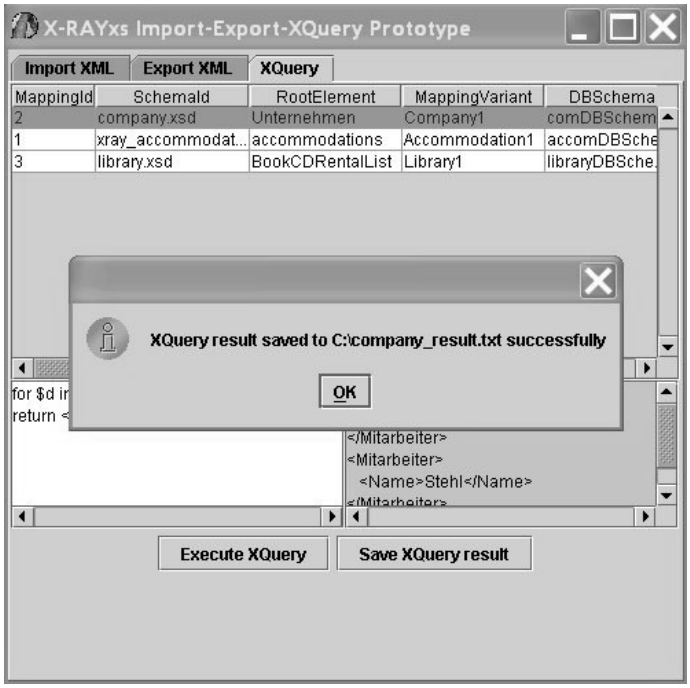


Abbildung 45 - GUI für XQuery

Bei der vom Benutzer eingegebenen XQuery-Abfragen muss die folgende Zeile weggelassen werden:

let $... = document("....xml")

Der Grund ist, dass *document* als Quelle automatisch die temporäre vom Prototyp erzeugte XML-Datei erhält. Nachstehend ein Beispiel für eine gültige XQuery-Abfrage mit dem Resultat. Die gewählte Abbildungsvariante bezieht sich auf *Accommodation.xml*.

```
for $d in $doc/accommodations/accommodation
return <accommodation>{$d/name}</accommodation>
```

Abbildung 46 - Beispiel XQuery für X-Rayxs

```
<accommodation>
      Steigenberger MAXX Hotel Linz
</accommodation>
<accommodation>
      Hotel Wolf-Dietrich
</accommodation>
```

Abbildung 47 - Resultat XQuery für X-Rayxs

6.6. Import

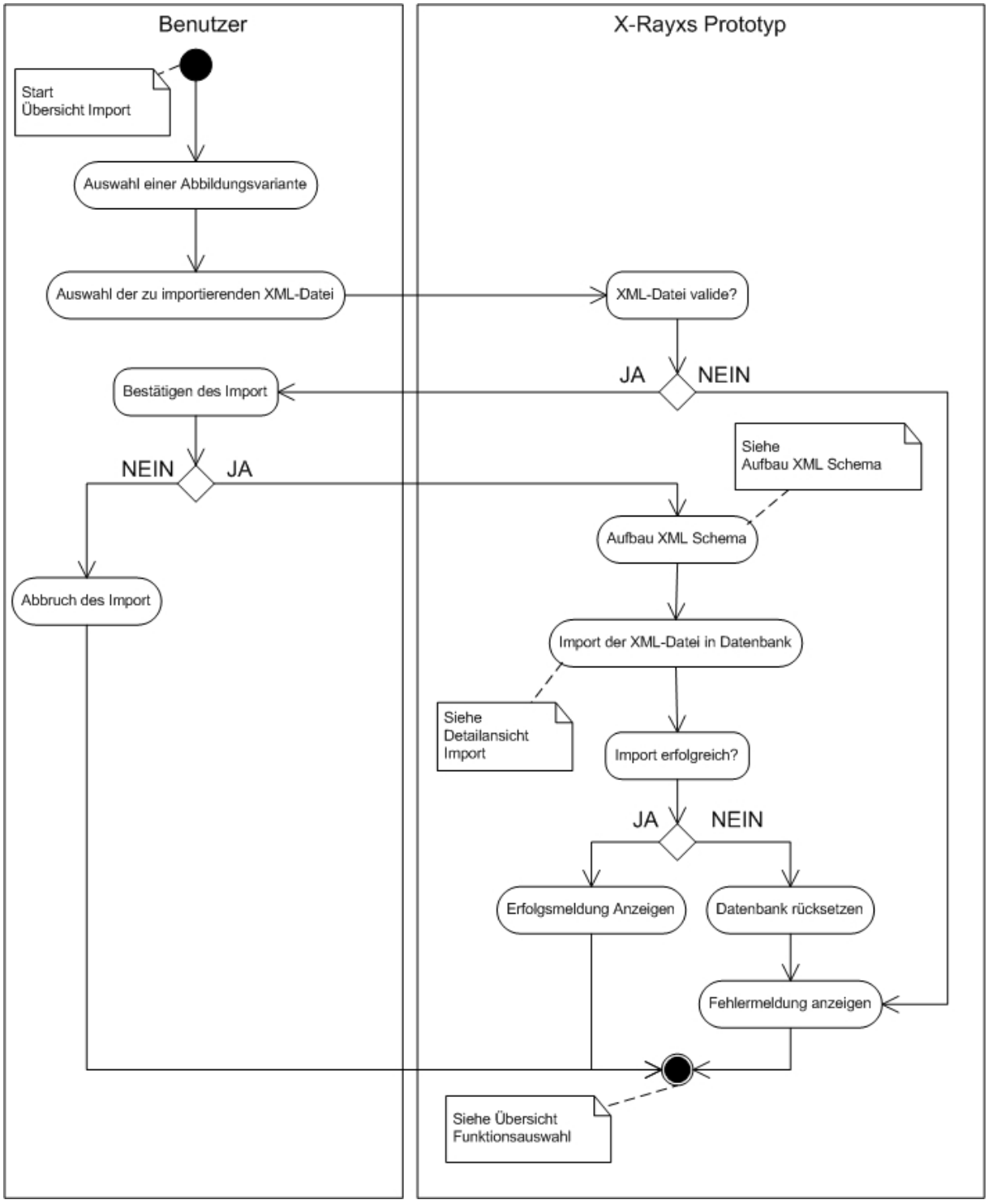


Abbildung 48 - Übersicht Import

Abbildung 48 zeigt den grundsätzlichen Ablauf des Imports einer XML-Datei in eine relationale Datenbank. Die Initialisierungsphase ist hier bereits vorweggenommen.

Der Prozess wird vom Benutzer durch das Starten des Prototyps X-Rayxs und der Auswahl des Import-Tabs angestoßen (siehe Abbildung 29). Wie bereits erwähnt, wird beim Starten des Prototyps ein lokales Speicherabbild der gesamten Metadatenbank, in Form von äquivalenten Instanzen von Java Klassen, erstellt (siehe Abbildung 32/Abbildung 33). Dem Benutzer werden nun die in der Initialisierungsphase eingerichteten Abbildungsvarianten angezeigt, aus denen er die, für die zu importierende XML-Datei passende auswählt. Anschließend werden die zuvor erzeugten Instanzen der Java-Klassen, die der *XML Schema-Komponente* zugehörig sind, für die gewählte Abbildungsvariante in Beziehung gesetzt. Das heißt, mit Hilfe von Zeigern und Containern wird die Struktur und Ordnung, die durch ein XML Schema vorgegeben ist, in das objektorientierte Speicherbild übergeführt. Wählt der Benutzer jetzt eine wohlgeformte und valide XML-Datei aus, ist dies der Start für den eigentlichen Import-Prozess. Dieser wird im folgenden Abschnitt genauer behandelt. Abgeschlossen wird der Import durch eine Mitteilung über den erfolgreichen Import, bzw. durch eine entsprechende Fehlermeldung. Im letzteren Fall wird die Datenbank, in der die Werte der XML-Datei gespeichert werden sollen, in den ursprünglichen Zustand zurückgesetzt.

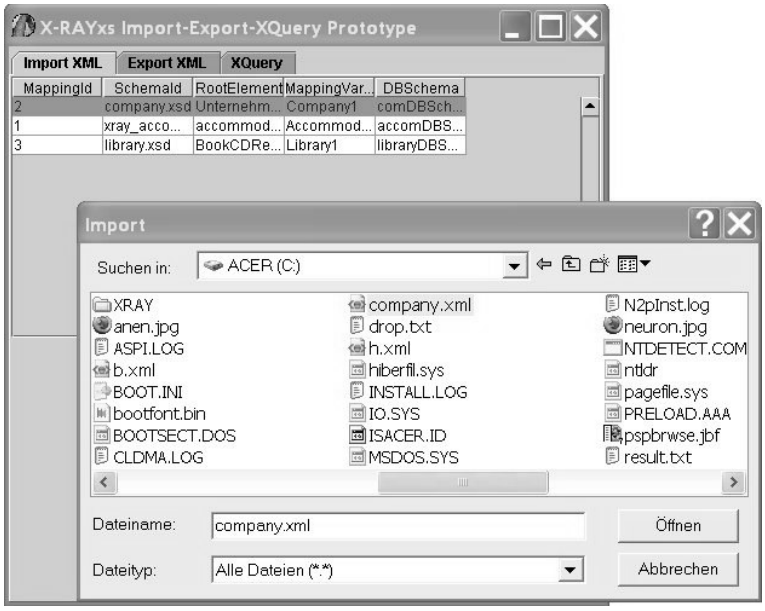


Abbildung 49 - Prototyp X-Rayxs, Import

6.7. Import Algorithmus

Die folgende Abbildung zeigt den internen Ablauf des Import Algorithmus. Sie stellt eine Verfeinerung der Aktivität „Import der XML-Datei in Datenbank“ aus Abbildung 48 dar.

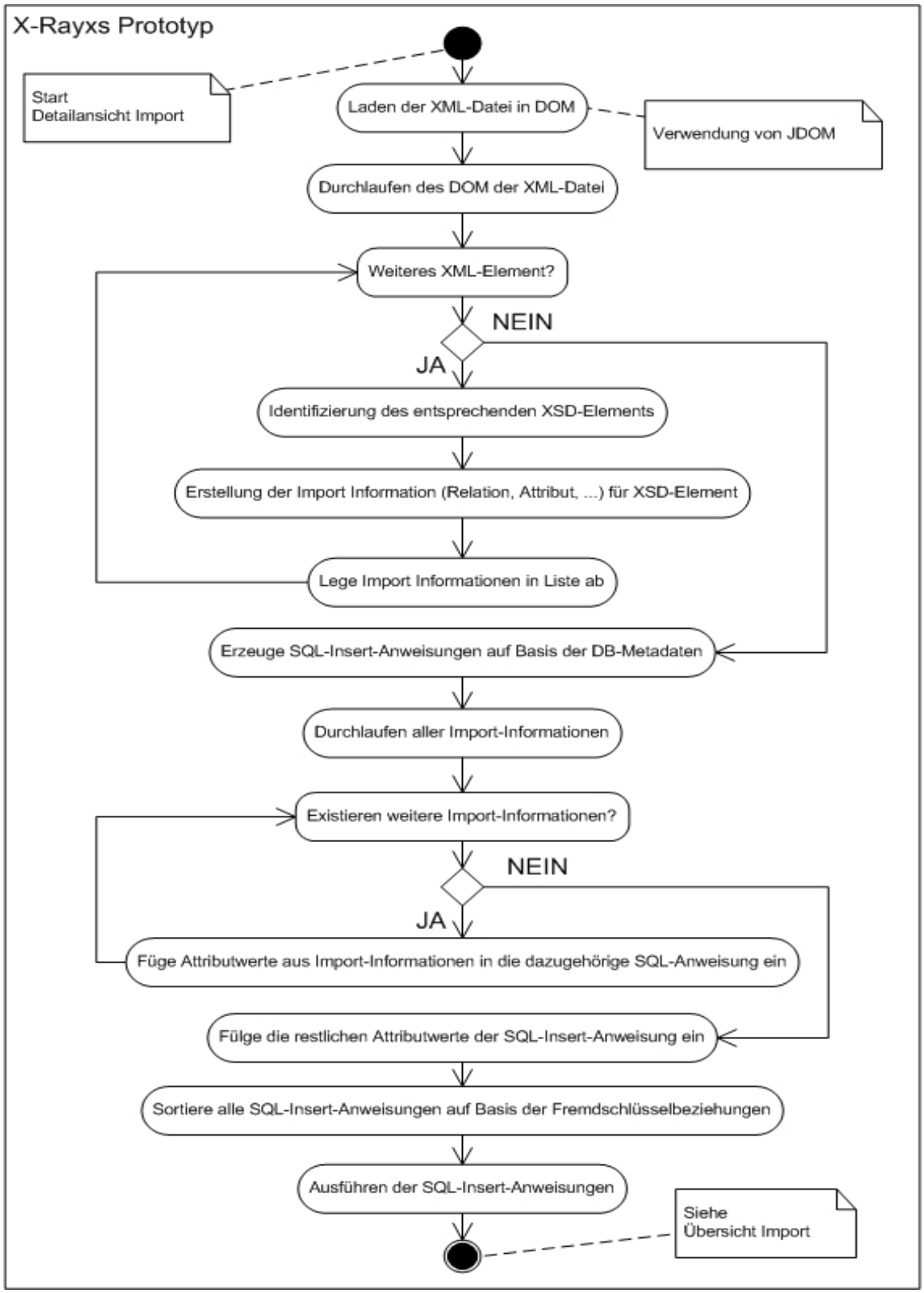


Abbildung 50 - Aktivitätsdiagram Import Algorithmus

Nachdem der Benutzer die zu importierende XML-Datei ausgewählt hat, wird diese Datei geladen. Um einen besseren Zugang zu den XML-Elementen und deren Attributen zu erhalten, kommt JDOM zum Einsatz. JDOM ist eine speziell auf Java zugeschnittene Klassenbibliothek zur Bearbeitung von XML-Dokumenten. JDOM ist ein Opensource-Projekt, das von Jason Hunter und Brett McLaughlin entwickelt wird und durch seine Java-typischen Eigenschaften und leichte Anwendung hervorsticht. Die Dokumentenmodelle DOM oder SAX liegen eine Ebene unter JDOM. JDOM ist der gemeinsame Aufbau, der die Vorteile von beiden Technologien vereint [JDOM05].

```
<accommodations xmlns='http://www.ifs.uni-
     linz.ac.at/XRay/AccommodationSchema'
     xmlns:su='http://www.ifs.uni-
     linz.ac.at/XRay/Supervision'
     xmlns:xsi='http://www.w3.org/2001/XMLSchema-
     instance' xsi:schemaLocation='http://www.ifs.uni-
     linz.ac.at/XRay/AccommodationSchema
     xray_accommodation.xsd'>
<accommodation id='45' state='Austria'>
<name>Steigenberger MAXX Hotel Linz</name>
...
```

Abbildung 51 - Beispiel Import

Liefert der Parser nun ein Element aus dem XML-Dokument, z.B. *<name>*, muss das dazugehörige XSD-Element ermittelt werden. Es muss also herausgefunden werden, welches Tupel aus der Tabelle *XSDElement* das XML-Element *<name>* beschreibt.
Dazu wird die Hierarchie einer XML-Datei bzw. des zugehörigen XML-Schemas ausgenutzt. Das XML-Element *<name>*, als auch das dazugehörige XSD-Element, befindet sich hierarchisch unter *<accommodation>*, und das wiederum unter *<accommodations>*. Somit lässt sich die Zusammengehörigkeit der beiden Elemente eindeutig identifiziert. Im Falle des XML-Elements *<name>* wird die *ElementId = 2* aus Tabelle *XSDElement* ermittelt. Dieser Eintrag zeigt ein Element, das auf ein global definiertes Element mit der *ElementId = 1* verweist.

XSDElement							
ElementId	Name	*TargetElementId (Ref)*	*Schema DeclarationId*	PrefixId	KindOfElement	KindOf ElementId	**…**
1	name		1		SP	1	…
2		1	1				…
3	street		1		SP	1	…
4	village		1	1	CT	1	…
….	…						…

Tabelle 9 - Auszug Relation XSDElement

Ist nun das XSD-Element bekannt, wird mit Hilfe der Information aus der Tabelle *ElementMapping* (vgl. Tabelle 10) ermittelt, wie mit dem Wert des XML-Elements weiter verfahren werden soll. Dazu wird für jedes Element, deren Wert importiert werden soll, in einem entsprechenden Java-Objekt (Klasse *ImportInfo* im *Paket xmlImport*) festgehalten, in welche Datenbank, welche Relation und letztendlich in welches Attribut der Relation der Wert abgelegt werden soll. Für *<name>* wird also ein *ImportInfo*-Objekt erzeugt, worin unter anderen die Attribute *RDBConnectionString* = „/localhost:3306/xray“ (Ort und Name der Datenbank), *RDBRelation* = “Accommodation” und *RDBAttribute* = “Name” festgehalten wird. Dieses neu erstellte *ImportInfo*-Objekt wird nun in einem Container abgelegt.

ElementMapping							
XMLElement	XSD ElementID	KindOf Mapping	RDBRelation	RDBAttribute	DBAttr	DBRelShipId	MappingId
accommodations	33	ET_0	NULL	NULL	NULL	NULL	1
accommodation	28	ET_R	Accommodation		NULL	NULL	1
name (ref)	2	ET_A1	Accommodation	Name	2	NULL	1
…							

: Zusatzinformation zur besseren Lesbarkeit

Tabelle 10 - Auszug Relation ElementMapping

Ist dieser Vorgang für jedes XML-Element abgeschlossen, werden sämtliche SQL-Insert-Anweisungen erzeugt, die für die erstellten *ImportInfo*-Objekte nötigt sind. Beim erstellen der Insert-Anweisungen wird jede Relation, in die ein Wert eingefügt

werden soll bzw. mit einer derartigen Relation in Beziehung steht, berücksichtigt. Im nächsten Schritt werden die erzeugten SQL-Insert-Anweisungen vervollständigt, indem die Attributwerte der *ImportInfo*-Objekte in die Anweisungen eingefügt werden. Abbildung 52 zeigt eine solche Insert-Anweisung.

```
INSERT INTO Accommodation (Sauna, AcceptsCreditCards, AccId,
State, Street, Name, VillageName) VALUES (…, …, …, …, …,
'Steigenberger MAXX Hotel Linz', …);
```

Abbildung 52 - Beispiel Insert-Anweisung

Wenn alle Attribute der Relationen berücksichtigt wurden, sind die SQL-Insert-Anweisungen fertig und können je nach Abhängigkeiten untereinander aufgrund der Schlüsselbeziehungen sortiert werden, um anschließend in der richtigen Reihenfolge ausgeführt zu werden.
Nach Abschluss dieser Schritte ist der Import beendet.

7. Ausblick

Bei der Implementierung des Prototyps wurden ausschließlich die Kernfunktionen gemäß den Anwendungsfällen (vgl. Kapitel **Anwendungsfälle**) berücksichtigt. Die objektorientierte Implementierung bietet aufgrund ihrer Anwendungsvielfalt zahlreiche Lösungsmöglichkeiten an. Deshalb stellt die vorliegende Lösung nur eine Mögliche dar. Im Hinblick auf die Zukunft gibt es noch einige offene Punkte, die im Folgenden aufgezählt werden.

- Derzeit müssen beim Export die Werte für die XML Elemente mittels Schlüssel einzeln aus der Datenbank ausgelesen werden. Dies bedeutet eine hohe Anzahl von Datenbankzugriffen beim Export bzw. beim Absetzen von XQuery-Abfragen. Durch Verwendung generischer Klassen in denen die Daten zu Beginn des Exportes geladen werden, könnte die Anzahl der Datenbankzugriffe drastisch reduziert werden. Diese generischen Klassen müssten auf Grund der Metadaten in der RDBS Schema-Komponente zur Laufzeit erzeugt werden. Dies könnte auch eine Modifikation des Schlüsselkonzepts (vgl. Abschnitt **Export**) nach sich ziehen.

- Derzeit wird beim Export von XML-Dokumenten auf JDOM verzichtet. JDOM bietet jedoch Methoden zum Erstellen von XML-Objekten die beim Export verwendet werden könnten, um die Generizität zu steigern.

- Um XQuery-Abfragen auszuführen, wird in der aktuellen Version OJXQI verwendet (vgl. **Architektur – Externe Pakete**). Die darin zur Verfügung gestellten Klassen ermöglichen es nicht, XQuery-Abfragen auf einen bestehenden DOM-Baum auszuführen. Um die Effizienz des Exports zu steigern wäre dies jedoch nötig. Zum Zeitpunkt der Entwicklung existierte eine derartige, ausgereifte Lösung jedoch noch nicht.

- Wie im Kapitel **X-Ray** erläutert, lässt sich auch X-Rayxs in die Initialisierungs- und Laufzeitphase unterteilen. Der X-Rayxs Prototyp berücksichtigt jedoch nur die Laufzeitphase. Die Implementierung der Initialisierungsphase von X-Rayxs ist derzeit noch offen.

- Die vorliegende Implementierung berücksichtigt ausschließlich das X-Rayxs Metaschema. Um auch XML-Dokumente zu unterstützen, die mittels DTD's beschrieben sind, wäre die Integration der Metaschemata von X-Ray und X-Rayxs nötig.

- Wie bereits erwähnt, konnten nicht alle Konzepte von XML Schema bei der Entwicklung des Metaschemas berücksichtigt werden. (vgl. **Nicht unterstützte XML Schema Konzepte**). Das Metaschema müsste dahingehend erweitert werden, um auch diese bisher unberücksichtigten Konzepte abbilden zu können. Entsprechend müsste auch der Prototyp erweitert werden, um sämtliche XML Schema Konzepte zu unterstützen.

Abbildungsverzeichnis

Abbildung 1 - Beispiel für DTD 13
Abbildung 2 - Beispiel für XML Schema (XSD) 15
Abbildung 3 - Architektur von X-Ray 23
Abbildung 4 - Grundlegende Abbildungsmuster zwischen XML und DB 24
Abbildung 5 - Verfeinerte Abbildungskonzepte 25
Abbildung 6 - Charakteristische Eigenschaften von XML-Elementtypen 27
Abbildung 7 - Sinnvolle Abbildungsstrategien für XML-Elememttypen 27
Abbildung 8 - Charakteristische Eigenschaften von XML-Attributen 29
Abbildung 9 - Sinnvolle Abbildungsstrategien für XML-Attribute 29
Abbildung 10 - Komponenten des X-Ray Metaschemas 30
Abbildung 11 - DB-Schema Komponente des Metaschemas 31
Abbildung 12 - XMLDTD-Komponente des Metaschemas 32
Abbildung 13 - Verfeinerung des zusammengesetzten ET 32
Abbildung 14 - Ausschnitt aus dem Metaschema (XMLDBSchemaMapping-Komponente) 33
Abbildung 15 - Beispiel Atomares Element 35
Abbildung 16 - Übersicht Datentypen 36
Abbildung 17 - Beispiel komplexer Datentyp 37
Abbildung 18 - Beispiel leeres Element 37
Abbildung 19 - Beispiel Vererbung 38
Abbildung 20 - Beispiel Identifizierungen und Referenzierungen 40
Abbildung 21 - XML Schema Datentyp Hierarchie [W3C04D] 41
Abbildung 22 - Beispiel lokale bzw. globale Elemente, Attribute und Datentypen 42
Abbildung 23 . Beispiel Namensräume 43
Abbildung 24 - Beispiel Schema Management 44
Abbildung 25 - Use case Diagramm 47
Abbildung 26 - X-Rayxs Login 54
Abbildung 27 - Übersicht Prototyp X-Rayxs 54
Abbildung 28 - Architektur Prototyp X-Rayxs 56
Abbildung 29 - UML-Diagramm des relationalen Schemas 61
Abbildung 30 - Übersicht X-Rayxs Prototyp 60

Abbildung 31 - Login X-Rayxs Prototyp 62
Abbildung 32 - Ladeprozess 63
Abbildung 33 - Aufbau XML Schema 65
Abbildung 34 - Übersicht Export 67
Abbildung 35 - Export GUI des X-Rayxs Prototyps 68
Abbildung 36 - Detailansicht Export 69
Abbildung 37 - Header von Accommodation.xml 70
Abbildung 38 - Beispiele für mit ET_R abgebildete Elemente 72
Abbildung 39 - Beispiel für mit ET_A1 abgebildetes Element 73
Abbildung 40 - Beispiele für ET_A2 abgebildete Elemente 75
Abbildung 41 - UML-Diagramm der Relation Accommodation und Pool 75
Abbildung 42 - Vererbung von abstrakten Datentypen 78
Abbildung 43 - Abfragen mit XQuery 80
Abbildung 44 - Anwendung von OJXQI 81
Abbildung 45 - GUI für XQuery 81
Abbildung 46 - Beispiel XQuery für X-Rayxs 82
Abbildung 47 - Resultat XQuery für X-Rayxs 82
Abbildung 48 - Übersicht Import 83
Abbildung 49 - Prototyp X-Rayxs, Import 84
Abbildung 50 - Aktivitätsdiagram Import Algorithmus 85
Abbildung 51 - Beispiel Import 86
Abbildung 52 - Beispiel Insert-Anweisung 88

Tabellenverzeichnis

Tabelle 1 - Relation und Klasse ElementMapping 71
Tabelle 2 - Datenrelation Accommodation 74
Tabelle 3 - Relation Pool 76
Tabelle 4 - Relation DBAttribute 76
Tabelle 5 - Relation ElementMapping 76
Tabelle 6 - Relation DBJoinSegment 76
Tabelle 7 - Beispiel für AttributeMapping 77
Tabelle 8 - Relation und Klasse zum Speichern von Vererbungsdaten 78
Tabelle 9 - Auszug Relation XSDElement 87
Tabelle 10 - Auszug Relation ElementMapping 87

Quellenverzeichnis

[W3C04A]

http://www.w3.org/TR/REC-xml

Extensible Markup Language (XML) 1.0 (Third Edition)

W3C Recommendation 04 February 2004

Editors: Tim Bray, Textuality and Netscape
Jean Paoli, Microsoft
C. M. Sperberg-McQueen, W3C05
Eve Maler, Sun Microsystems, Inc. - Second Edition
François Yergeau - Third Edition

zuletzt besucht: 15.03.2005

[W3C04B]

http://www.w3.org/TR/xmlschema-1/

XML Schema Part 1: Structures Second Edition

W3C05 Recommendation 28 October 2004

Editors: Henry S. Thompson, University of Edinburgh
David Beech, Oracle Corporation
Murray Maloney, for Commerce One
Noah Mendelsohn, Lotus Development Corporation

zuletzt besucht: 15.03.2005

[SCHN04]

http://www-lehre.inf.uos.de/~tschnied/HTML/

XML Schema & XSL-Transformationen

Tanja Schniederberend

Universität Osnabrück 11.06.2003

zuletzt besucht: 15.03.2005

[GALI04]

http://www.galileocomputing.de/glossar/

Glossar von Galileo Press

zuletzt besucht: 15.03.2005

[W3C04C]

http://www.w3.org/TR/xmlschema-0/

XML Schema Part 0: Primer Second Edition

W3C05 Recommendation 28 October 2004

Editors: David C. Fallside, IBM

Priscilla Walmsley - Second Edition

zuletzt besucht: 15.03.2005

[W3C04D]

http://www.w3.org/TR/xmlschema-2/

XML Schema Part 2: Datatypes Second Edition

W3C05 Recommendation 28 October 2004

Editors: Paul V. Biron, Kaiser Permanente, for Health Level Seven

Ashok Malhotra, Microsoft (formerly of IBM)

zuletzt besucht: 15.03.2005

[W3C05]

http://www.w3.org

The World Wide Web Consortium (W3C)

zuletzt besucht: 15.03.2005

[DVIN03A]

http://www.xml.dvint.com/docs/SchemaStructuresQR-2.pdf

XML Schema – Structures Quick Reference

2003 D Vint Productions

http://www.xml.dvint.com

ver 1/03

zuletzt besucht: 15.03.2005

[KENN00]

http://www.mut.de/media_remote/katalog/bsp/3827259444bsp.htm

SOAP developer's guide

ISBN 3-8272-5944-4

Kennard Scribner / Marc C. Stive

Dezember 2000

zuletzt besucht: 15.03.2005

[ECKS04]

http://www.dbis.informatik.hu-berlin.de/lehre/WS0405/XMLSW/VL/xml_VLKap4.4.pdf

Script zur Vorlesung XML und Semantic Web 2004

Dr. Rainer Eckstein

zuletzt besucht: 15.03.2005

[DVIN03B]

http://www.xml.dvint.com/docs/SchemaDataTypesQR-2.pdf

XML Schema - Data Types Quick Reference

2001 D VInt Productions

http://www.xml.dvint.com

ver 1/03

zuletzt besucht: 15.03.2005

[OBAS04]

http://www.microsoft.com/germany/msdn/library/data/xml/W3C05XMLSchemaEntwurfsmusterVermeidenVonKomplexitaet.mspx

W3C05 XML Schema-Entwurfsmuster - Vermeiden von Komplexität

Veröffentlicht: 09. Apr 2003

Aktualisiert: 22. Jun 2004

Von Dare Obasanjo

zuletzt besucht: 15.03.2005

[HOLZ04]

http://www.xml-schnittstelle.de/xml-schemas.html

Studie: Entwicklung einer XML-Schnittstelle für den deutschen Lebensversicherungsmarkt am Beispiel der Antragstellung

Dipl.Math.oec. Martin Holzwarth

Kapitel 6: XML-Schemas

zuletzt besucht: 15.03.2005

[JDOM05]

http://www.jdom.org/

JDOM 2005

Jason Hunter

Brett McLaughlin

zuletzt besucht: 15.03.2005

[ULLE04]

http://www.galileocomputing.de/openbook/javainsel4/index.htm

Java ist auch eine Insel

Christian Ullenboom

Programmieren für die Java 2-Plattform in der Version 5 (Tiger-Release)

zuletzt besucht: 15.03.2005

[KAPP04]

ftp://ftp.ifs.uni-linz.ac.at/pub/publications/2004/0304.pdf

Integrating XML and Relational Database Systems

GERTI KAPPEL, ELISABETH KAPSAMMER and WERNER RETSCHITZEGGER

Kluwes Academic Publishers, Vol 7 (4) December 2004, pp. 343-384

zuletzt besucht: 15.03.2005

[KRAU05]

Entwurf eines Metaschemas zur Integration von relationalen Datenbanken und XML Schema.

Thomas Kraushofer, Diplomarbeit; Johannes Kepler Universität Linz; 2005

[QUIL00]

http://www.almaden.ibm.com/cs/people/chamberlin/quilt.html

Quilt: An XML Query Language, 2000

Jonathan Robie, Daniela Florescu, Don Chamberlin

zuletzt besucht: 15.03.2005

[ORAC05]

http://www.oracle.com/technology/sample_code/tech/xml/xmldb/jxqi.html

JXQI: Java XQuery API (developed by Oracle)

zuletzt besucht: 15.03.2005

Anhang A: Beispiel XML-Dokument mit Schemadateien

xray_accommodation.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.ifs.uni-linz.ac.at/XRay/AccommodationSchema"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:ac="http://www.ifs.uni-
linz.ac.at/XRay/AccommodationSchema" xmlns:su="http://www.ifs.uni-linz.ac.at/XRay/Supervision"
elementFormDefault="qualified">
    <import namespace="http://www.ifs.uni-linz.ac.at/XRay/Supervision"
schemaLocation="xray_supervision.xsd"/>
    <redefine schemaLocation="xray_management.xsd">
        <complexType name="managerType">
            <complexContent>
                <extension base="ac:managerType">
                    <sequence>
                        <element name="salary" type="positiveInteger"/>
                    </sequence>
                    <attribute name="mgrId" type="positiveInteger"/>
                    <attribute name="since" type="gYear"/>
                </extension>
            </complexContent>
        </complexType>
    </redefine>
    <!-- Start of elements-->
    <element name="name" type="string"/>
    <element name="accommodations">
        <complexType>
            <sequence>
                <element name="accommodation" type="ac:accommodationType" minOccurs="0"
                    maxOccurs="unbounded"/>
                <element name="accAwards" type="ac:accAwardsType"/>
            </sequence>
        </complexType>
        <key name="AccKey">
            <selector xpath="ac:accommodation"/>
            <field xpath="@id"/>
        </key>
```

```
        <keyref name="AwardWinnerRef" refer="AccKey">
            <selector xpath="ac:accAwards/ac:award"/>
            <field xpath="ac:winner"/>
        </keyref>
    </element>
    <!-- Start of complex types-->
    <complexType name="accommodationType">
        <sequence>
            <element ref="ac:name"/>
            <element name="address" type="ac:addressType"/>
            <element name="manager" type="ac:managerType"/>
            <element name="supervisor" type="su:supervisionType"/>
            <element name="email" type="string" minOccurs="0" maxOccurs="unbounded"/>
            <element name="phone" type="ac:phoneType" maxOccurs="unbounded"/>
            <element name="acceptsCreditCards" minOccurs="0">
                <complexType/>
            </element>
            <element name="facilities">
                <complexType/>
            </element>
            <element name="sauna" type="ac:saunaType"/>
            <element name="pool" minOccurs="0" maxOccurs="unbounded">
                <complexType/>
            </element>
            <element name="description" type="ac:descriptionType" minOccurs="0"/>
            <element name="room" type="ac:roomType" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="id" type="positiveInteger" use="required"/>
        <attribute name="state" type="string" fixed="Austria"/>
    </complexType>
    <complexType name="roomType" abstract="true">
        <sequence>
            <element name="roomNumber" type="positiveInteger"/>
            <element name="size" type="positiveInteger"/>
            <element name="nrOfBeds" type="positiveInteger"/>
        </sequence>
    </complexType>
    <complexType name="standardRoomType">
        <complexContent>
            <extension base="ac:roomType">
                <sequence>
```

```
                <element name="tv" type="string" minOccurs="0" maxOccurs="2"/>
                <element name="shower" minOccurs="0">
                    <complexType/>
                </element>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<complexType name="luxuryRoomType">
    <complexContent>
        <extension base="ac:standardRoomType">
            <attribute name="hasButler" type="boolean" use="required"/>
        </extension>
    </complexContent>
</complexType>
<complexType name="villageType">
    <simpleContent>
        <extension base="string">
            <attribute name="yearOfFoundation" type="gYear"/>
        </extension>
    </simpleContent>
</complexType>
<complexType name="addressType">
    <sequence>
        <element name="street" type="string"/>
        <element name="village" type="ac:villageType"/>
        <element name="country" type="string"/>
    </sequence>
    <attribute name="postalCode" type="string" use="required"/>
</complexType>
<complexType name="descriptionType">
    <all minOccurs="0" maxOccurs="unbounded">
        <element name="rating" type="string"/>
        <element name="comment" type="string"/>
    </all>
</complexType>
<complexType name="phoneType">
    <attribute name="number" type="string" use="required"/>
</complexType>
<complexType name="saunaType">
    <attribute name="available" type="boolean" use="required"/>
```

```
    </complexType>
    <complexType name="accAwardsType">
        <sequence>
            <element name="award" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                    <sequence>
                        <element name="year" type="gYear"/>
                        <element name="winner" type="positiveInteger"/>
                    </sequence>
                </complexType>
            </element>
        </sequence>
    </complexType>
</schema>
```

xray_survision.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.ifs.uni-linz.ac.at/XRay/Supervision"
xmlns:su="http://www.ifs.uni-linz.ac.at/XRay/Supervision"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <complexType name="supervisionType">
        <sequence>
            <element name="supId" type="positiveInteger"/>
            <element name="firstName" type="string"/>
            <element name="lastName" type="string"/>
        </sequence>
    </complexType>
</schema>
```

xray_management.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.ifs.uni-linz.ac.at/XRay/AccommodationSchema"
xmlns:ac="http://www.ifs.uni-linz.ac.at/XRay/AccommodationSchema"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <complexType name="managerType">
        <sequence>
            <element name="firstName" type="string"/>
            <element name="lastName" type="string"/>
        </sequence>
    </complexType>
</schema>
```

accommodation.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<accommodations xmlns="http://www.ifs.uni-linz.ac.at/XRay/AccommodationSchema"
xmlns:su="http://www.ifs.uni-linz.ac.at/XRay/Supervision"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.ifs.uni-
linz.ac.at/XRay/AccommodationSchema xray_accommodation.xsd">
    <accommodation id="45" state="Austria">
        <name>Steigenberger MAXX Hotel Linz</name>
        <address postalCode="4020">
            <street>Am Winterhafen 13</street>
            <village yearOfFoundation="1600">Linz</village>
            <country>Oberoesterreich</country>
        </address>
        <manager mgrId="1" since="1995">
            <firstName>Karin</firstName>
            <lastName>Maier</lastName>
            <salary>20000</salary>
        </manager>
        <supervisor>
            <su:supId>2</su:supId>
            <su:firstName>Klaus</su:firstName>
            <su:lastName>Klammer</su:lastName>
        </supervisor>
        <email>service.linz@maxxhotel.at</email>
        <phone number="+4373278990"/>
```

```
        <acceptsCreditCards/>
        <facilities/>
        <sauna available="false"/>
        <pool/>
        <description>
            <comment>noisy</comment>
            <rating>3</rating>
            <comment>nice rooms</comment>
            <rating>2</rating>
        </description>
        <room xsi:type="standardRoomType">
            <roomNumber>101</roomNumber>
            <size>25</size>
            <nrOfBeds>2</nrOfBeds>
            <tv>Panasonic TX-32PS11D</tv>
            <shower/>
        </room>
        <room xsi:type="standardRoomType">
            <roomNumber>105</roomNumber>
            <size>33</size>
            <nrOfBeds>2</nrOfBeds>
        </room>
    </accommodation>
    <accommodation id="46" state="Austria">
        <name>Hotel Wolf-Dietrich</name>
        <address postalCode="5020">
            <street>Wolf-Dietrich-Strasse 7</street>
            <village yearOfFoundation="1500">Salzburg</village>
            <country>Salzburg</country>
        </address>
        <manager mgrId="2" since="1985">
            <firstName>Klaus</firstName>
            <lastName>Klammer</lastName>
            <salary>30000</salary>
        </manager>
        <supervisor>
            <su:supId>1</su:supId>
            <su:firstName>Herman</su:firstName>
            <su:lastName>Maier</su:lastName>
        </supervisor>
        <email>office@salzburg-hotel.at</email>
```

```
        <phone number="0043 662 871275"/>
        <facilities/>
        <sauna available="true"/>
        <pool/>
        <pool/>
        <description>
            <comment>beautiful</comment>
            <rating>1</rating>
            <comment>very nice landscape</comment>
            <rating>1</rating>
        </description>
        <room xsi:type="luxuryRoomType" hasButler="true">
            <roomNumber>305</roomNumber>
            <size>95</size>
            <nrOfBeds>3</nrOfBeds>
            <tv>Sony KV-32FQ85</tv>
        </room>
    </accommodation>
    <accAwards>
        <award>
            <year>2003</year>
            <winner>46</winner>
        </award>
        <award>
            <year>2004</year>
            <winner>45</winner>
        </award>
    </accAwards>
</accommodations>
```

Anhang B: Relationen des Metaschemas mit Beispieldaten

Relationen der XML Schema - Komponente

XSDImport			
Id	*SchemaDeclarationIdBase*	*SchemaDeclarationIdImport*	*NameSpaceId*
	1	2	3

XSDRedefine			
Id	*SchemaDeclarationIdBase*	*SchemaDeclarationIdRedefine*	*ComplexDataTypeId*
	1	3	4

XSDInclude		
Id	*SchemaDeclarationIdBase*	*SchemaDeclarationIdInclude*

XSDNameSpaces	
NameSpaceId	Uri
1	http://www.ifs.uni-linz.ac.at/XRay/AccommodationSchema
2	http://www.w3.org/2001/XMLSchema
3	http://www.ifs.uni-linz.ac.at/XRay/Supervision

XSDPrefix			
PrefixId	*NameSpaceId*	*SchemaDeclarationId*	Prefix
1	1	1	ac
2	2	1	
3	3	1	su
4	2	2	
5	3	2	su
6	2	3	
7	1	3	ac

XSDComplexDataType								
ComplexDataTypeId	Id	Abstract	Block	Final	Name	*Schema DeclarationId*	KindOf Type	KindOf TypeId
1					villageType	1	SC	1
2					addressType	1	CM	1
3					managerType	3	CM	2
4					managerType	1	CC	1
5					supervisorType	2	CM	4
6					phoneType	1	EC	1
7						1	EC	2
8						1	EC	3
9					saunaType	1	EC	4
10						1	EC	5
11					descripionType	1	CM	5
12		true			roomType	1	CM	6
13					accommodationType	1	CM	7
14						1	CM	8
15					accAwardsType	1	CM	9
16						1	CM	10
17						1	EC	6
18					standardRoomType	1	CC	2
19					luxuryRoom	1	CC	3

ComplexDataType – KindOfType:

CM ContentModel
SC SimpleContent
CC ComplexContent
EC EmptyComplexDataType

XSDSimplePredefinedDataType	
SimplePredefinedDataTypId	Name
1	string
2	normalizedstring
3	token
4	language
5	name
6	ncname
7	nmtoken
8	nmtokens
9	id
10	idref
11	idrefs
12	entities
13	base64binary
14	hexbinary
15	boolean
16	float
17	double
18	decimal
19	integer
20	nonpoitiveinteger
21	negativeinteger
22	long
23	int
24	short
25	byte
26	nonnegativeinteger
27	unsignedlong
28	unsignedint
29	unsignedschort
30	unsignedbyte
31	positiveinteger
32	anyuri
33	qname
34	notaion
35	duration
36	datetime
37	time
38	date
39	gyearmonth
40	gyear
41	gmonthday
42	gday
43	gmonth

XSDSimpleContent		
SimpleContentId	Id	*PrefixId*
1		

XSDSimpleContentRestriction		
SimpleContentRestricionId	Id	*RestrictsSimpleContentId*

XSDSimpleContentExtension			
SimpleContentExtentionId	Id	KindOfExtension	KindOfExtensionId
1		SP	1

CT ComplexType
SP SimplePredefinedDataType

XSDComplexContent					
ComplexContentId	Id	ContentModelId	KindOfBase	KindOfBaseId	*PrefixId*
1		3	CT	3	1
2		11	CT	12	1
3			CT	18	1

CT ComplexDataType
SP SimplePredefinedDataTypeId

XSDComplexContent Restriction	
ComplexContentRestricionId	Id

XSDComplexContent Extension	
ComplexContentExtentionId	Id
1	
2	
3	

XSDContentModel		
ContentModelId	KindOfContentModel	KindOfContentModelId
1	S	1
2	S	2
3	S	3
4	S	4
5	A	1
6	S	5
7	S	6
8	S	7
9	S	8
10	S	9
11	S	10

A All
C Choice
S Sequence

XSDAll			
AllId	Id	MinOccurs	MaxOccurs
1		0	unbonded

XSDChoice			
ChoiceId	Id	MinOccurs	MaxOccurs

XSDSequence			
SequenceId	Id	MinOccurs	MaxOccurs
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

XSDSequencePosition		
SequenceId	*ContentParticleId*	Position
1	3	1
1	4	2
1	5	3
2	8	1
2	9	2
3	11	1
4	14	1
4	15	2
4	16	3
5	28	1
5	29	2
5	30	3
6	2	1
6	7	2
6	13	3
6	18	4
6	19	5
6	20	6
6	21	7
6	22	8
6	23	9
6	24	10
6	27	11
6	33	12
7	35	1
7	36	2
8	37	1
9	34	1
9	38	2
10	40	1
10	41	2

XSDAllElements	
AllId	*ElementId*
1	21
1	22

XSDContentParticle		
ContentParticleId	KindOfContentParticle	KindOfContentParticleId
1	E	1
2	E	2
3	E	3
4	E	4
5	E	5
6	S	1
7	E	6
8	E	7
9	E	8
10	S	2
11	E	9
12	S	3
13	E	10
14	E	11
15	E	12
16	E	13
17	S	4
18	E	14
19	E	15
20	E	16
21	E	17
22	E	18
23	E	19
24	E	20
25	E	21
26	E	22
27	E	23
28	E	24
29	E	25
30	E	26
31	S	5
32	S	6
33	E	27
34	E	28
35	E	29
36	E	30
37	E	31
38	E	32
39	E	33
40	E	34
41	E	35

C Choice
S Sequence
E Element

XSDChoiceContentParticle	
ChoiceId	*ContentParticleId*

XSDEmptyComplexDataType
EmptyComplexDataTypeId
1
2
3
4
5
6

SP SimplePredefinedDataType
CT ComplexDataType

XSDIdentityConstraints	
IdentityConstraintsId	*ElementId*
1	34

XSDUnique		
UniqueId	Id	Name

XSDKey		
KeyId	Id	Name
1		AccKey

XSDKeyRef			
KeyRefId	Id	Name	*KeyId (refer)*
1		AwardWinnerRef	1

XSDSelector				
SelectorId	Id	XPath	*KindOfMembership*	*KindOfMembershipId*
1		ac:accommodation	K	1
2		ac:awards/ac:award	R	1

XSDField				
FieldId	Id	XPath	*KindOfMembership*	*KindOfMembershipId*
1		@id	K	1
2		ac:winner	R	1

U Unique
K Key
R KeRef

XSDElement																	
ElementId	Id	Abstract	Block	DefaultValue	Final	Fixed	Form	MinOccurs	MaxOccurs	Name	Nillable	*TargetElementId(Ref)*	*SchemaDeclarationId*	PrefixId	KindOfElement	KindOfElementId	isRoot
1										name			1	1	SP	1	0
2												1	1				0
3										street			1		SP	1	0
4										village			1	1	CT	1	0
5										country			1		SP	1	0
6										address			1	1	CT	2	0
7										firstName			3		SP	1	0
8										lastName			3		SP	1	0
9										salary			1		SP	31	0
10										manager			1	1	CT	4	0
11										supId			2		SP	31	0
12										firstName			2		SP	1	0
13										lastName			2		SP	1	0
14										supervisor			1	5	CT	5	0
15								0	unbounded	email			1		SP	1	0
16									unbounded	phone			1	1	CT	6	0
17								0		acceptsCreditCards			1	1	CT	7	0
18										facilities			1		CT	8	0
19										sauna			1	1	CT	9	0
20								0	unbounded	pool			1		CT	10	0
21										rating			1		SP	1	0
22										comment			1		SP	1	0
23								0		description			1	1	CT	11	0
24										roomNumber			1		SP	31	0
25										size			1		SP	31	0
26										nrOfBeds			1		SP	31	0
27									unbounded	room			1	1	CT	12	0
28								0	unbounded	accommodation			1	1	CT	13	0
29										year			1		SP	40	0
30										winner			1		SP	1	0
31								0	unbounded	award			1		CT	14	0
32										accAwards			1	1	CT	15	0
33										accommodations			1		CT	16	1
34								0	2	tv			1		SP	1	0
35								0		shower			1		CT	17	0

XSDAttribute												
AttributeId	Id	Default Value	Fixed	Form	Name	UseValue	*PrefixId*	*TargetAttributeId*	*SchemaDeclarationId*	KindOf Membership	KindOf MembershipId	*SimplePredefinedDataTypeId*
1					yearofFoundation				1	SC	1	40
2					postalCode	required			1	CM	1	1
3					mgrId				1	CC	1	31
4					since				1	CC	1	40
5					number	required			1	EC	1	1
6					available	required			1	EC	4	15
7					id	required			1	CM	7	31
8			Austria		state				1	CM	7	1
9					hasBulter	required			1	CC	3	15

ATTRIBUTE - *KindOfMembership*

CM ContentModel
EC EmptyComplexDataType
SP SimplePredefidedDataType
SC SimpleContend
CC ComplexContent

XSDSchemaDeclaration									
Schema DeclarationId	Id	Attribute Form Default	Block Default	Element Form Default	Final Default	Version	XMLlang	Filename	*TargetNameSpaceId*
1				qualified				xray_accommodation.xsd	1
2				qualified				xray_supervision.xsd	3
3				qualified				xray_management.xsd	1

Relationen der RDB Schema - Komponente

DBSchema				
DBSchema	DBName	DBConnectString	DBUserName	DBPassword
accomDBSchema	lehre92	140.78.90.210:1521:	xray	xray

DBAttribute					
DBAttId	*DBSchema*	DBRelation	DBAttribute	DBIsKey	DBDataType
1	accomDBSchema	Accommodation	AccId	1	INT
2	accomDBSchema	Accommodation	Name	0	VARCHAR
3	accomDBSchema	Accommodation	Street	0	VARCHAR
4	accomDBSchema	Accommodation	State	0	VARCHAR
5	accomDBSchema	Accommodation	VillageName	0	VARCHAR
6	accomDBSchema	Accommodation	AcceptsCreditCards	0	BOOLEAN
7	accomDBSchema	Accommodation	Sauna	0	BOOLEAN
8	accomDBSchema	Manager	MgrId	1	VARCHAR
9	accomDBSchema	Manager	FirstName	0	VARCHAR
10	accomDBSchema	Manager	LastName	0	VARCHAR
11	accomDBSchema	Manager	Salary	0	INT
12	accomDBSchema	Manager	Since	0	INT
13	accomDBSchema	Manager	AccId	0	INT
14	accomDBSchema	Supervisor	SupId	1	VARCHAR
15	accomDBSchema	Supervisor	FirstName	0	VARCHAR
16	accomDBSchema	Supervisor	LastName	0	VARCHAR
17	accomDBSchema	Supervisor	AccId	0	INT
18	accomDBSchema	Room	RoomNr	0	INT
19	accomDBSchema	Room	RoomSize	0	INT
20	accomDBSchema	Room	NrOfBeds	0	INT
21	accomDBSchema	Room	HasShower	0	BOOLEAN
22	accomDBSchema	Room	AccId	0	INT
23	accomDBSchema	Room	RoomId	1	INT
24	accomDBSchema	LuxuryRoom	HasButler	0	BOOLEAN
25	accomDBSchema	LuxuryRoom	RoomId	1	INT
26	accomDBSchema	TV	Description	0	VARCHAR
27	accomDBSchema	TV	RoomId	1	INT
28	accomDBSchema	Village	Name	1	VARCHAR
29	accomDBSchema	Village	PostalCode	0	INT
30	accomDBSchema	Village	CountryId	0	INT
31	accomDBSchema	Country	CountryId	1	INT
32	accomDBSchema	Country	Name	0	VARCHAR
33	accomDBSchema	History	VillageName	1	VARCHAR
34	accomDBSchema	History	YearFound	0	INT
35	accomDBSchema	Pool	AccId	1	INT
36	accomDBSchema	Pool	Name	1	VARCHAR
37	accomDBSchema	Phone	AccId	1	INT
38	accomDBSchema	Phone	Number	1	VARCHAR
39	accomDBSchema	EmailAddress	AccId	1	INT
40	accomDBSchema	EmailAddress	Email	1	VARCHAR
41	accomDBSchema	Description	DescriptionId	1	INT

42	accomDBSchema	Description	AccId	0	INT
43	accomDBSchema	Description	Rating	0	INT
44	accomDBSchema	Description	Comment	0	VARCHAR
45	accomDBSchema	AccAwards	Winner	1	INT
46	accomDBSchema	AccAwards	Year	1	INT

DBJoinSegment						
DBRelShipId	*DBAtt1*	*DBAtt2*	*DBRelation1*	*DBRelation2*	*DBSchema*	DBJoinDirection
1	Name	VillageName	Village	Accommodation	accomDBSchema	12
2	AccId	AccId	Phone	Accommodation	accomDBSchema	21
3	Winner	AccId	AccAwards	Accommodation	accomDBSchema	21
4	AccId	AccId	EmailAddress	Accommodation	accomDBSchema	21
5	AccId	AccId	Pool	Accommodation	accomDBSchema	21
6	AccId	AccId	Description	Accommodation	accomDBSchema	21
7	VillageName	Name	History	Village	accomDBSchema	21
8	CountryId	CountryId	Country	Village	accomDBSchema	12
9	AccId	AccId	Manager	Accommodation	accomDBSchema	21
10	AccId	AccId	Supervisor	Accommodation	accomDBSchema	21
11	AccId	AccId	Room	Accommodation	accomDBSchema	21
12	RoomId	RoomId	TV	Room	accomDBSchema	21
13	RoomId	RoomId	LuxuryRoom	Room	accomDBSchema	21

Relationen der Mapping – Komponente

ElementMapping				
XSDElementID	KindOfMapping	DBAttr	DBRelShipId	MappingId
33	ET_0	NULL	NULL	1
28	ET_R	NULL	NULL	1
2	ET_A1	2	NULL	1
6	ET_0	NULL	NULL	1
3	ET_A1	3	NULL	1
4	ET_A2	28	1	1
5	ET_A2	32	8	1
15	ET_A2	40	4	1
16	ET_R	NULL	2	1
17	ET_A1	6	NULL	1
18	ET_0	NULL	NULL	1
19	ET_0	NULL	NULL	1
20	ET_A2	36	5	1
23	ET_R	NULL	6	1
21	ET_A2	43	6	1
22	ET_A2	44	6	1
10	ET_R	NULL	9	1
7	ET_A2	9	9	1
8	ET_A2	10	9	1
9	ET_A2	11	9	1
14	ET_R	NULL	10	1
11	ET_A2	14	10	1
12	ET_A2	15	10	1
13	ET_A2	16	10	1
32	ET_0	NULL	NULL	1
31	ET_R	NULL	3	1
29	ET_A2	46	3	1
30	ET_A2	45	3	1
27	ET_R	NULL	11	1
24	ET_A2	18	11	1
25	ET_A2	19	11	1
26	ET_A2	20	11	1
34	ET_A2	26	12	1
35	ET_A2	21	11	1

AttributeMapping					
XSDElementID	XSDAttributeID	KindOfMapping	DBAttr	DBRelShipId	MappingId
28	7	A_A1	1	NULL	1
28	8	A_0	NULL	NULL	1
6	2	A_A2	29	1	1
4	1	A_A2	34	7	1
16	5	A_A2	38	2	1
19	6	A_A1	7	NULL	1
10	3	A_A2	8	9	1
10	4	A_A2	12	9	1
27	9	A_A2	24	13	1

Mappings				
MappingId	SchemaDeclarationId	RootElementId	MappingVariant	DBSchema
1	1	33	Accommodation1	accomDBSchema

AbstractTypeKeys			
DBRelation	DBKeyAttr	DBKeyValue	HeritageType
Room	RoomId	1	standardRoomType
Room	RoomId	2	luxuryRoomType
Room	RoomId	3	standardRoomType

ElementBaseRelation		
MappingId	ElementId	DBRelation
1	28	Accommodation

Relationen zum Speichern der Daten von accommodation.xml

Accommodation						
AccId	Name	Street	State	*VillageName*	Accepts CreditCards	Sauna
45	Steigenberger MAXX Hotel Linz	Am Winterhafen 13	Austria	Linz	true	false
46	Hotel Wolf-Dietrich	Wolf-Dietrich-Straße 7	Austria	Salzburg	true	true

Manager					
MgrId	FirstName	LastName	Salary	Since	*AccId*
1	Karin	Maier	20000	1995	45
2	Klaus	Klammer	30000	1985	46

Supervisor			
SupId	FirstName	LastName	*AccId*
1	Herman	Maier	46
2	Hans	Klammer	45

Room					
RoomId	RoomSize	NrOfBeds	HasShower	*AccId*	RoomNr
1	25	2	true	45	101
2	95	3	false	46	301
3	33	2	false	45	105

LuxuryRoom	
RoomId	HasButler
2	true

TV	
RoomId	Description
1	Panasonic TX-32PS11D
2	Sony KV-32FQ85

Village		
Name	PostalCode	*CountryId*
Linz	4020	1
Salzburg	5020	2

Country	
CountryId	Name
1	Oeberoesterreich
2	Salzburg

History	
VillageName	YearFound
Linz	1600
Salzburg	1500

Pool	
AccId	Name
45	Pool1
46	Pool1
46	Pool2

Phone	
AccId	Number
45	+4373278990
46	0043 662 /87 12 75

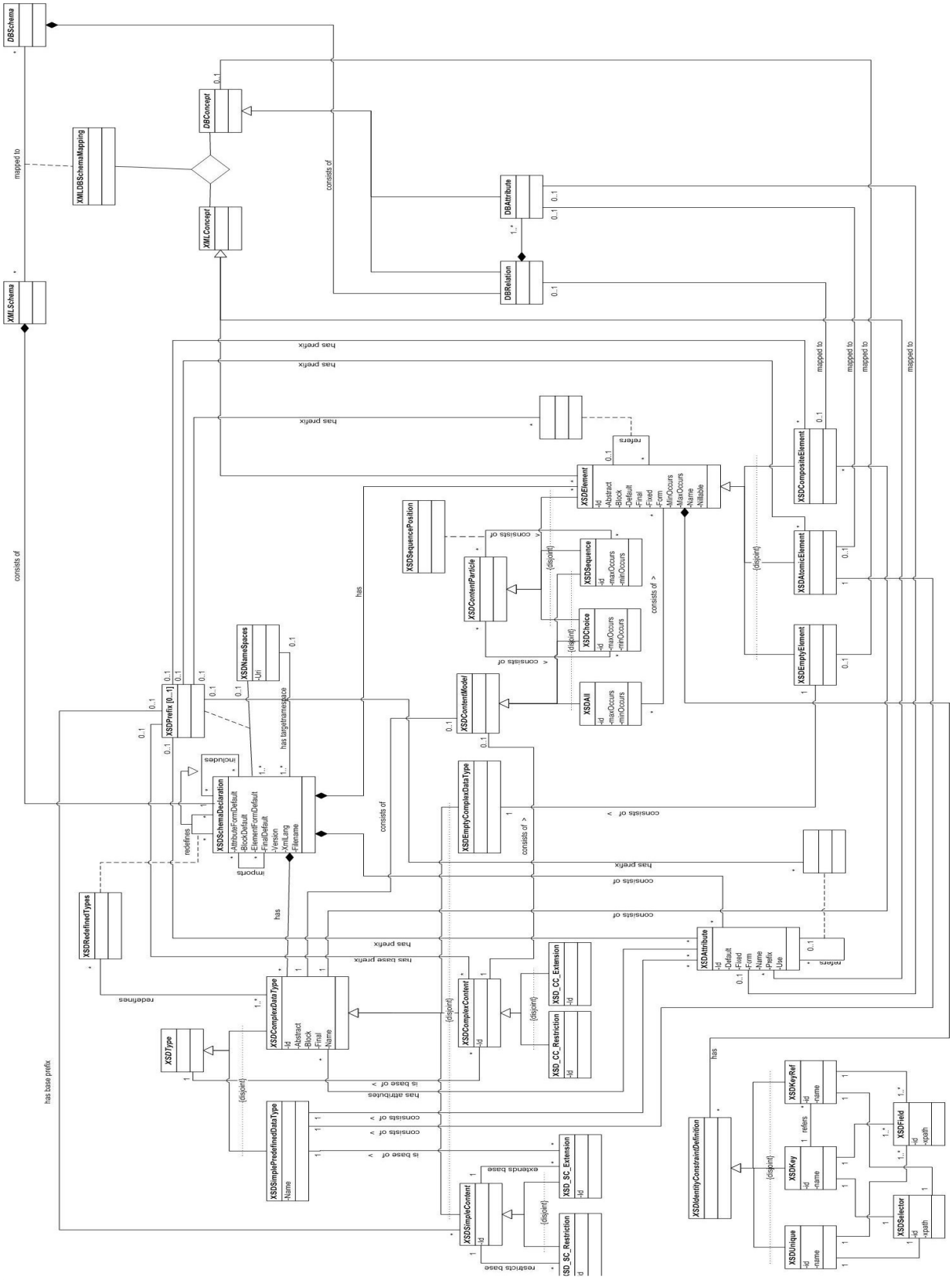
EmailAddress	
AccId	Email
45	service.linz@maxxhotel.at
46	office@salzburg-hotel.at

Description			
DescriptionId	*AccId*	Rating	GuestComment
1	45	3	noisy
2	45	2	nice rooms
3	46	1	beautiful
4	46	1	very nice landscape

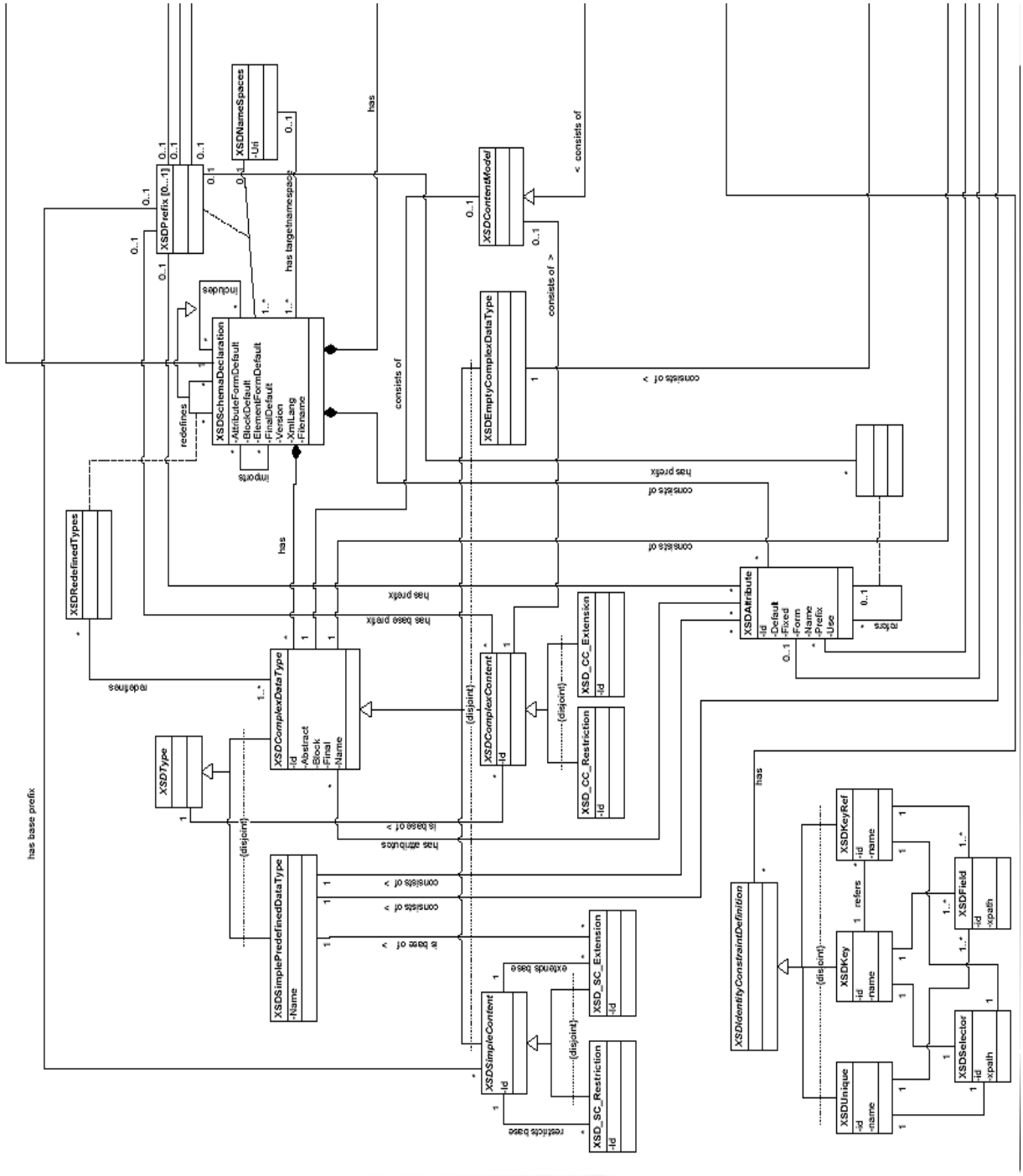
AccAwards	
Winner	Year
46	2003
45	2004

Anhang C: UML Diagramme

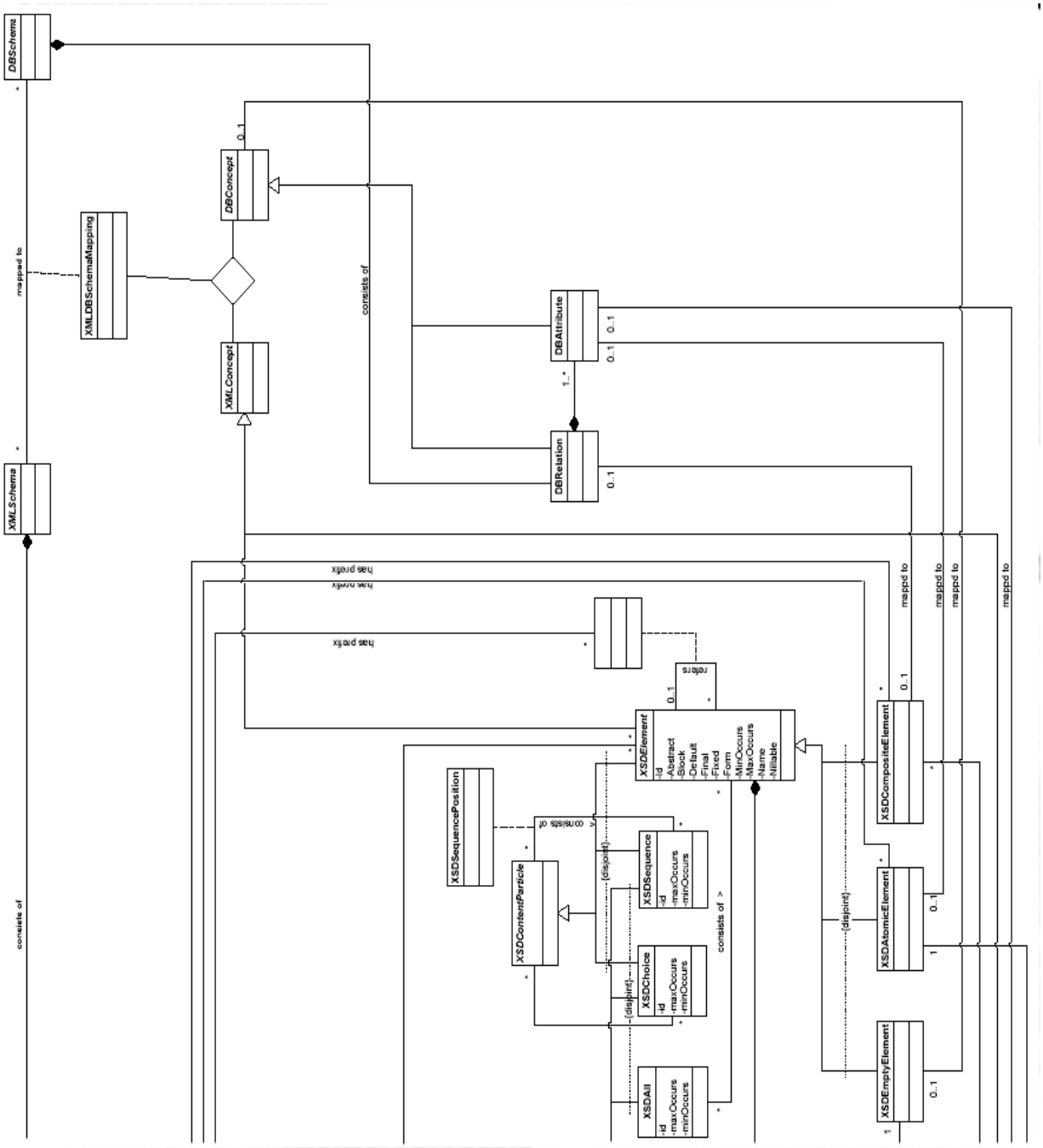
UML Diagramm des Metaschemas von X-Rayxs - Gesamtansicht



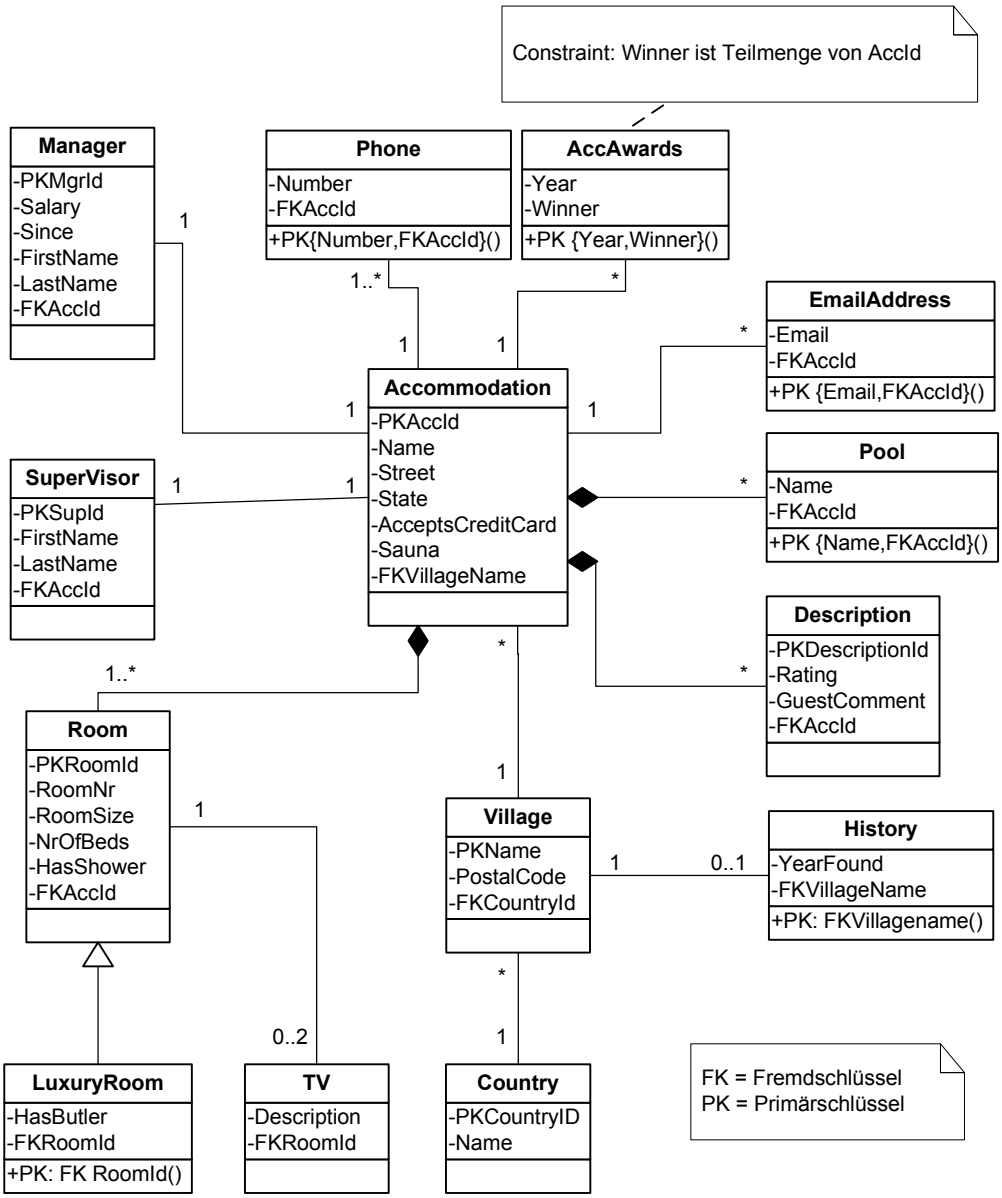
UML Diagramm des Metaschemas von X-Rayxs – Teil 1/2



UML Diagramm des Metaschemas von X-Rayxs – Teil 2/2



UML Diagramm des relationalen Schemas



Anhang D: Installationsanleitung / Benutzerhandbuch

Prototyp X-Rayxs

Installationsanleitung / Benutzerhandbuch

Version 1.2

22.01.2005

Christian Ortner 9956842

Inhaltsverzeichnis

1. Installationsanleitung 125

1.1. Software Vorraussetzungen 125

1.2. Schritt-für-Schritt Installation 126

1.2.1. Entpacken von XRAYxsInstall.zip 126

1.2.2. Anpassen der Demonstrationsbeispiele 127

1.2.3. Erstellen der X-Rayxs-Datenbank 127

1.2.4. Anpassen der Verbindungsdaten für die Metaschemadatenbank 128

2. Benutzerhandbuch 130

2.1. Starten des XRAYxs-Prototyps 130

2.1.1. Wichtige Hinweise: 130

2.2. Login am XRAYxs-Prototypen 131

2.3. Übersicht des XRAYxs-GUI 132

2.4. Import: Daten aus XML-Dokument importieren 133

2.5. Export: Daten in ein XML-Dokument exportieren 134

2.6. XQuery: Abfrage auf ein gespeichertes XMLSchema absetzen 136

1. Installationsanleitung

Im folgenden Abschnitt werden alle nötigen Schritte beschrieben, die zu einer erfolgreichen Installation des Prototypen X-Rayxs nötig sind.
Wenn sämtliche Schritte erfolgreich abgeschlossen sind, befindet sich die X-Rayxs in der Laufzeitphase. Zusätzlich sind drei Beispiele (Accommodations, Company und Library) zu Demonstrationszwecken enthalten. Durch diese drei Beispiele wurde die Initialisierungsphase von X-Rayxs vorweggenommen, um den Export und Import von Daten sowie das Durchführen von Abfragen mittels XQuery zu ermöglichen.

1.1. Software Vorraussetzungen

Um die Installation erfolgreich abzuschließen wird Folgendes benötigt:

1 Die Datei XRAYxsInstall.zip
2 Oracle (Version 9i R2) oder MYSQL (4.0 oder höher) Datenbank und Benutzerdaten.
3 Javafähiges Betriebssystem mit JavaRuntimeEnvironment 1.4.2 oder höher
4 Verbindung zu einer der oben genannten Datenbank
5 7 MB freien Festplattenspeicherplatz
6 Archivierungsprogramm (z.B. WinZip)
7 Texteditor

1.2. Schritt-für-Schritt Installation

1.2.1. Entpacken von XRAYxsInstall.zip

Entpacken Sie die Datei XRAYxsInstall.zip in ein beliebiges Verzeichnis (z.B. nach c:\Programme\) mit Hilfe eines Archivierungsprogramms wie z.B. WinZip. Anschließend finden Sie folgende Dateistruktur an dem von Ihnen gewählten Installationsort vor:

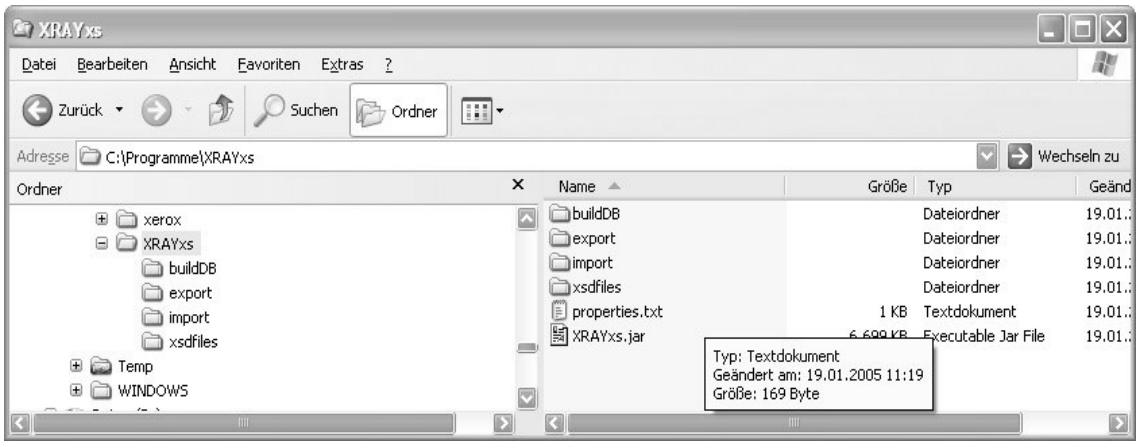


Abbildung 53 Screenshot Installationsverzeichnisstruktur

buildDB: Dieser Ordner beinhaltet sämtliche Scriptdateien (SQL-create und –insert Anweisungen) zum Erstellen und Initialisieren aller benötigten Datenbankrelationen

export: Leerer Ordner zum Ablegen der exportierten XML-Dateien.

import: Dieser Ordner beinhaltet drei XML-Dateien, die als Demonstrationsbeispiele für den Import, Export und für Abfragen benötigt werden.

xsdfiles: Dieser Ordner beinhaltet die XMLSchemata der Demonstrationsbeispiele.

properties.txt: Textdatei zum Einrichten der Datenbankverbindung für XRAYxs.

XRAYxs.jar*:* Ausführbare JAR-Datei, zum Starten des X-Rayxs-Prototype.

1.2.2. Anpassen der Demonstrationsbeispiele

Öffnen Sie dazu die Datei …\buildDB\insertAllDBMeta.sql mit einem beliebigen Texteditor. Um die nötigen Anpassungen vorzunehmen, müssen Sie die ersten drei insert-Anweisungen abändern. Bei Verwendung einer Oracle Datenbank muss die Insert-Anweisung folgendermaßen aussehen:

```
insert into DBSchema values ('xxx', 'DBNAME'
,'HOST:PORT:','DBUSER','DBPASSWORD');
```

Bei Verwendung einer MySQL-Datenbank:

```
insert into DBSchema values ('xxx',
'DBNAME','/HOST:PORT/','DBUSER','DBPASSWORD');
```

Ersetzen Sie in beiden Fällen den DBNAME mit dem Oracle/MySQL Datenbanknamen, HOST mit der IP-Adresse des Datenbankservers, PORT mit dem Datenbankport, DBUSER mit dem Datenbank Login-Namen und DBPASSWORD mit dem dazugehörigen Passwort.
Achten Sie in beiden Fällen auf die unterschiedliche Doppelpunkt- und Backslash -Zeichensetzung.
Speichern Sie diese Änderungen.

1.2.3. Erstellen der X-Rayxs-Datenbank

Dazu müssen sämtliche Skriptdateien, welche die SQL-create und SQL-insert Anweisungen enthalten, ausgeführt werden. Sie befinden sich im Ordner …\buildDB\.
Dies kann beispielsweise mit Hilfe eines Webinterface, wie PHPMyAdmin für MySQL Datenbanken oder iSQL*Plus für Oracle, erfolgen.

Folgende Reihenfolge ist beim Ausführen der Scriptdateien zu beachten:

1. createAll
2. restliche Skriptdateien (insert Anweisungen)

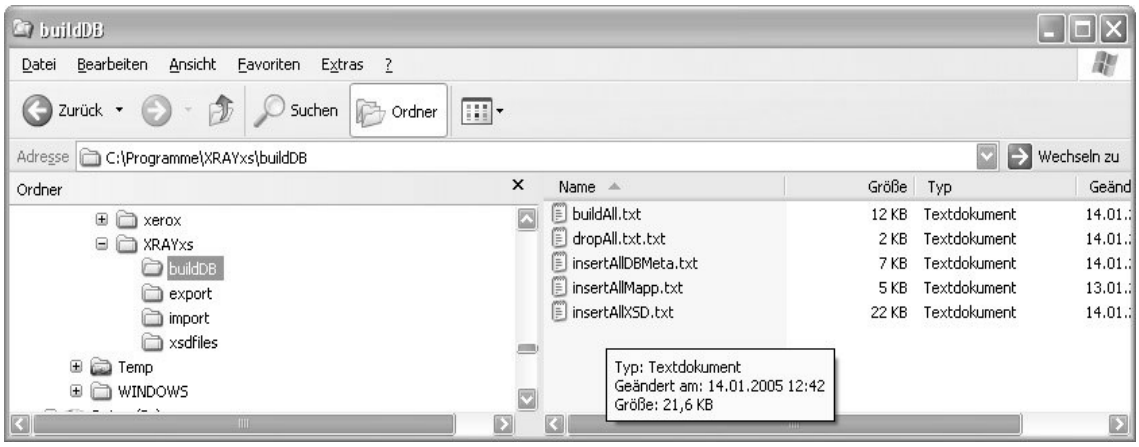


Abbildung 54 Screenshot2 Übersicht der Scriptdateien

createAll.sql: Erzeugt alle nötigen Relationen für die XRAYxs-Metaschema-Datenbank, sowie die Relationen der drei Demonstrationsbeispiele.

insertAllDBMeta.sql: Initialisiert die X-Rayxs-Metaschema-Datenbank mit den Metadaten, welche die Relationen der Demonstrationsbeispiele beschreiben.

insertAllMapp.sql: Befüllt die X-Rayxs-Metaschema-Datenbank mit den Metadaten, die das Abbildungswissen repräsentieren.

insertAllXSD.sql: Befüllt die X-Rayxs-Metaschema-Datenbank mit den Metadaten, welche die XMLSchemata der drei Demonstrationsbeispiele beschreiben.

dropAll.sql: PLSQL-Script zum Löschen aller Relationen der Datenbank, für eine eventuelle Neuinstallation der Datenbank. *ACHTUNG: dieses PLSQL-Script löscht ALLE Relationen eines Benutzers in der jeweiligen Datenbank.*

1.2.4. Anpassen der Verbindungsdaten für die Metaschemadatenbank

Öffnen Sie nun die Datei *properties.txt* mit einem beliebigen Texteditor, und adaptieren Sie die Attribute entsprechend Ihrer Bedürfnisse. Alle Attributwerte in dieser Datei beziehen sich auf die Datenbank, in der die SQL-Anweisungen aus dem Ordner …/buildDB/… ausgeführt wurden.

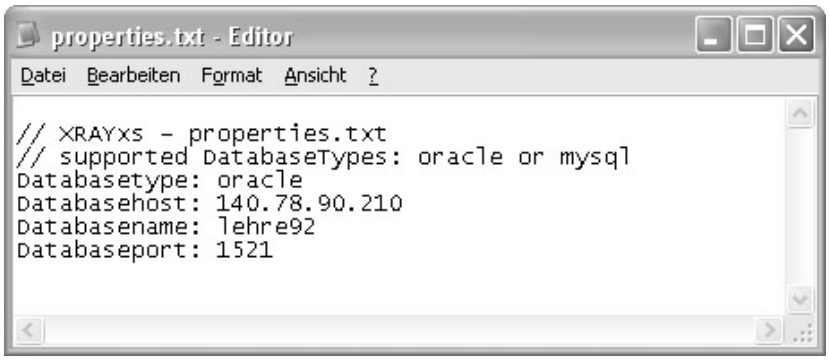


Abbildung 55 Properties-Datei

Databasetype: Gibt die verwendete Datenbank an.

Attributwert für Oracle Datenbanke ist „oracle“, für MySQL „mysql“.

Databasehost: Gibt die IP-Adresse des Datenbankservers an.

z.B. 127.0.0.1 für localhost.

Databasename: Gibt den Name der verwendeten Datenbank an.

Databaseport: Gibt den Port der Datenbank an.

Speichern Sie diese Änderungen.

Die Installation sowie die Initialisierungsphase von X-Rayxs sind nach Durchführung obiger Schritte abgeschlossen.

Informationen zur Benutzung von X-Rayxs finden Sie im Benutzerhandbuch.

2. Benutzerhandbuch

2.1. Starten des XRAYxs-Prototyps

Nach erfolgreicher Installation der X-Rayxs-Applikation kann dieser mittels der Datei *XRAYxs.jar* im Installationsordner (siehe Installationsanleitung) gestartet werden. Daraufhin erscheint das Login-Fenster von X-Rayxs-Prototypen.

2.1.1. Wichtige Hinweise:

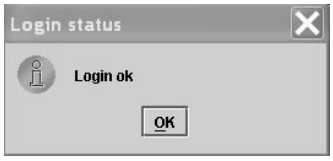
- Durch die Installation wird die Initialisierungsphase von X-Rayxs vorweggenommen und der Prototyp befindet sich nach seinem Start in der Laufzeitphase.
- Es stehen drei XML-Dokumente, samt Metadaten und Daten zum Testen des Prototyps zur Verfügung.
- Die Relationen des Metaschemas sowie jene zur Speicherung der Daten aus/für XML-Dokumente(n) sind in einer Datenbank gemeinsam gespeichert.
- Das Geschwindigkeit des Erstellens der oo-Repräsentation des Metaschemas nach dem erfolgreichen Login am Prototyp kann je Datenbankverbindung unterschiedlich lang dauern.
- Die Dauer des Imports, Exports und Ausführens von Abfragen ist stark von der Datenbankverbindung abhängig.
- Die beiliegenden Beispiele können nur einmal importiert werden. Im Prototype wird nicht überprüft ob eine xml-Datei bereits importiert wurde.
- Das PLSQL-Script, Datei dropAll.sql, dient zum Löschen aller Relationen der Datenbank, für eine eventuelle Neuinstallation der Datenbank. ACHTUNG: dieses PLSQL-Script löscht ALLE Relationen eines Benutzers in der jeweiligen Datenbank.

Es werden die wesentlichen Funktionen des Prototypen XRAYxs vorgestellt.

Diese sind:

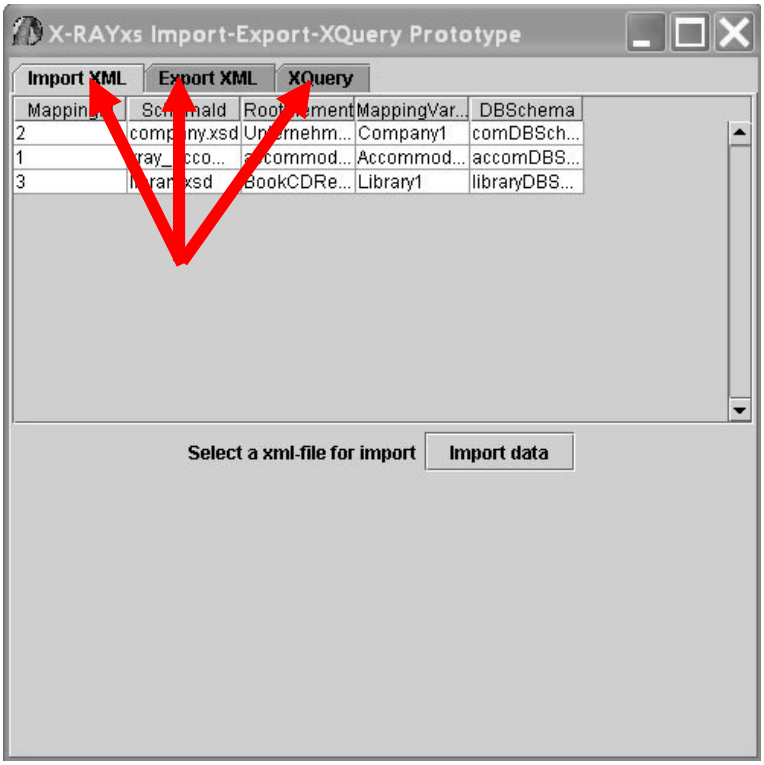
- Login am XRAYxs-Prototypen
- Import - Daten aus einem XML-Dokument importieren
- Export - Daten in eine XML-Dokument exportieren
- XQuery – auf ein gespeichertes XMLSchema anwenden

2.2. Login am XRAYxs-Prototypen

• Starten der Datei XRAYxs.jar	
• Es erscheint ein Login-Fenster	
• Es muss ein gültiger Benutzername und ein gültiges Passwort eingeben werden. • Danach ist der OK-Button zu klicken	
• Es wird eine der drei folgenden Meldungen angezeigt:	
• Login ok Nach Klicken des OK-Buttons wird das Hauptfenster von XRAYxs geöffnet	
• Benutzername/Passwort sind inkorrekt	
• Verbindungsfehler Keine Datenbank-verbindung oder der Treiber ist nicht vorhanden	

Nach erfolgreichem Login wird das Hauptfenster von XRAYxs geöffnet, wo man die drei Hauptfunktionen des Prototypen verwenden kann.
Dabei wird standardmäßig als erstes die *ImportGUI* angezeigt.

2.3. Übersicht des XRAYxs-GUI

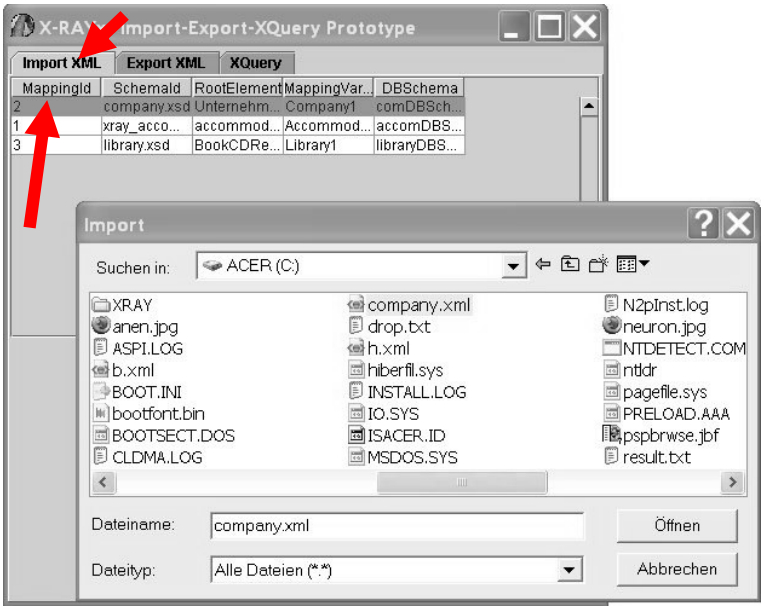
Durch Klicken auf die Tabs (rote Pfeile) wird eine spezifische Funktion des Prototypen ausgewählt. Die Tabelle darunter zeigt die gespeicherten XMLSchemata und Abbildungsvarianten.	

Die Tabelle zeigt folgende Informationen an:

- MappingId — Identifiziert eindeutig eine Schema/Abbildungs-Variante
- SchemaId — Name der XSD-Datei (Schemadatei)
- RootElement — Name des Wurzelelementes
- MappingVariant — Bezeichnung für die Abbildungsvariante
- DBSchema — Name des DB-Schemas, auf welche das jeweilige XMLSchema abgebildet ist

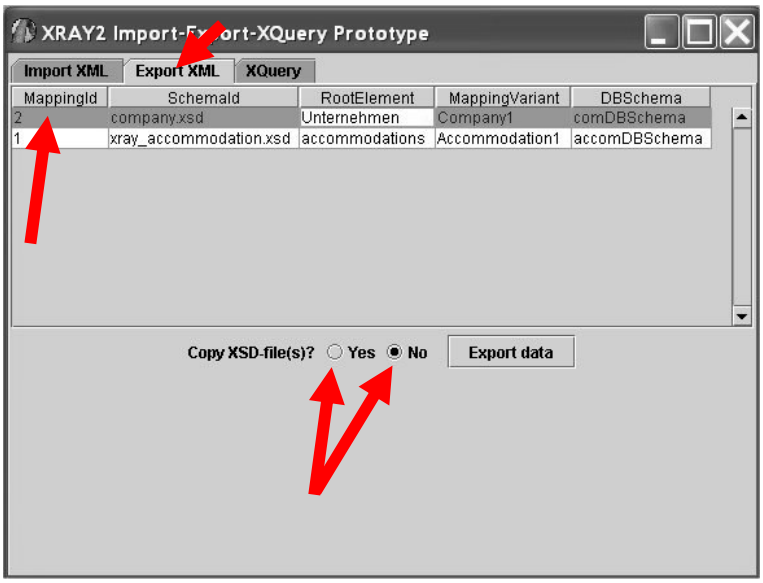
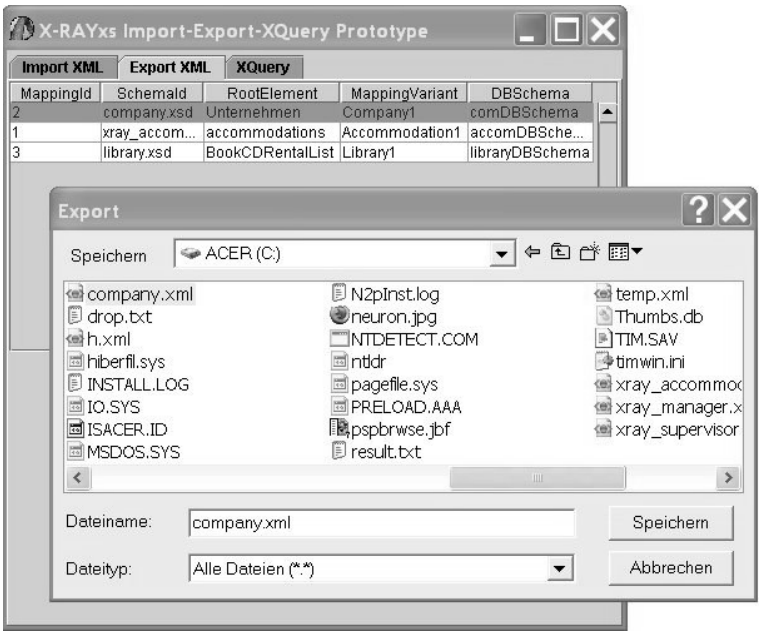
2.4. Import: Daten aus XML-Dokument importieren

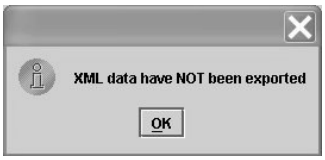
Durch Klicken auf den „Import XML“-Tab wird die GUI zum Importieren von Daten aus einem XML-Dokumentes geöffnet.

Auswahl einer Mappingvariante durch Klicken auf die entsprechende Zeile in der Tabelle. Nach Klicken des „Import“-Buttons wird ein Dialog geöffnet, mit welchem man ein XML-Dokument für den Import der Daten auswählen kann. Nach erfolgreichem Import wird eine Erfolgsmeldung angezeigt	

2.5. Export: Daten in ein XML-Dokument exportieren

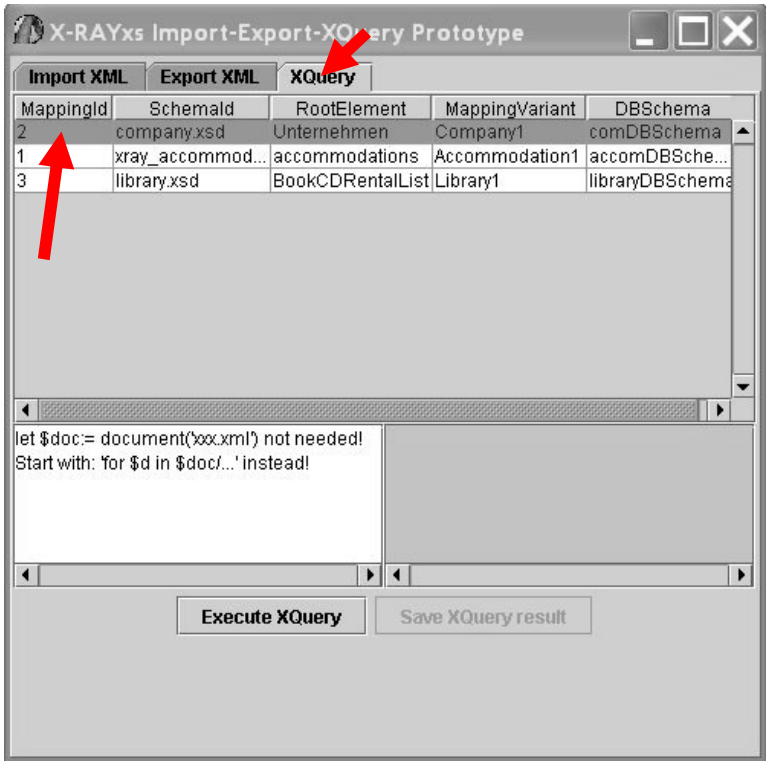
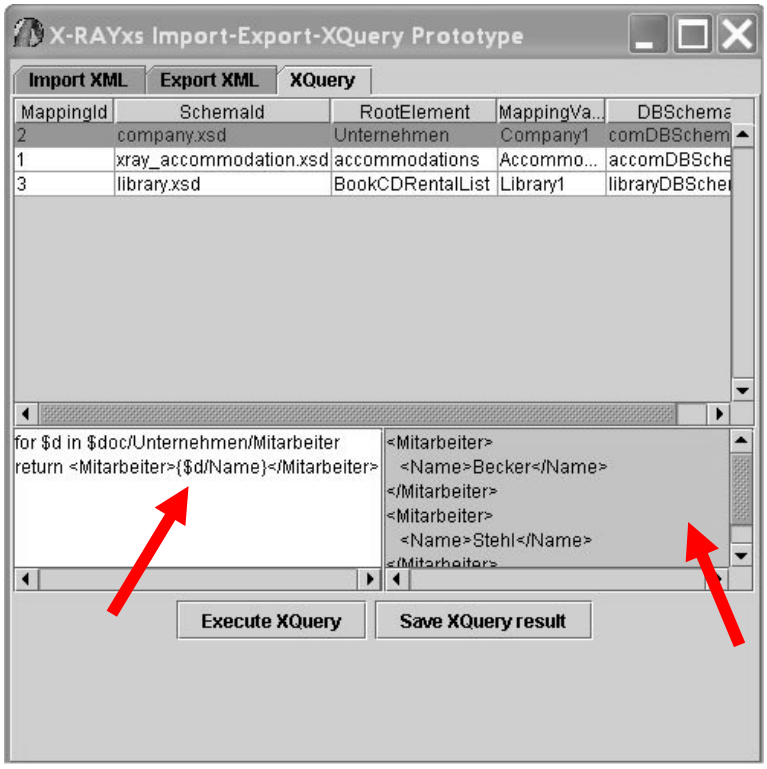
Durch Klicken auf den „Export XML“-Tab wird die GUI zum Exportieren von Daten in ein XML-Dokument geöffnet.

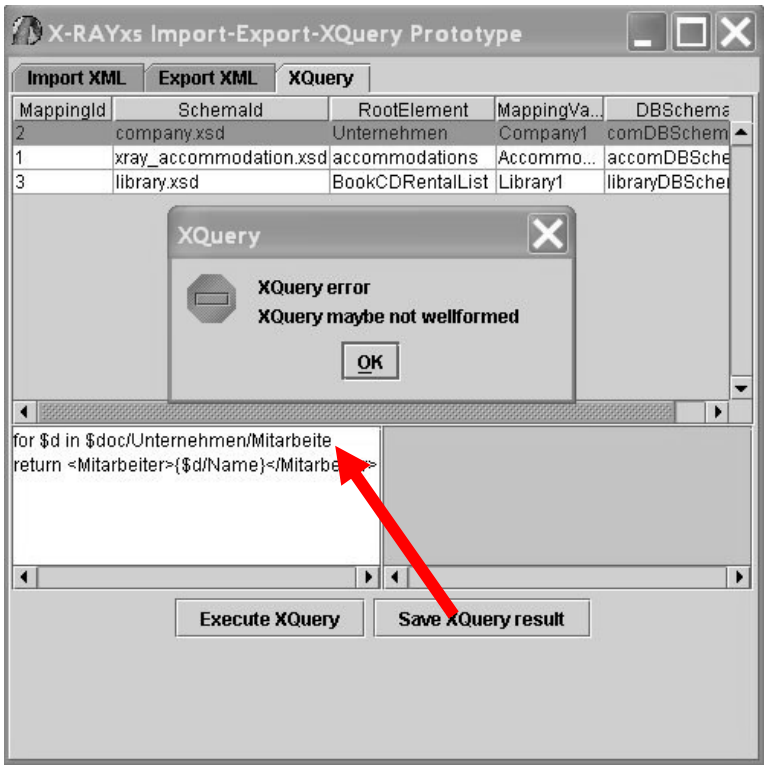
Auswahl einer Mappingvariante durch Klicken auf die entsprechende Zeile in der Tabelle. Zusätzlich kann man mit den YES/NO-Radiobuttons wählen, ob die Schemadatei (sowie alle zugehörige Schemadateien) an den Speicherort des XML-Dokumentes kopiert werden sollen. Nach Klicken des „Export data“-Buttons wird ein Dialog geöffnet, um den Speicherort für das XML-Dokument auszuwählen.	
Mit dem Export-Dialog kann man den Speicherort und Namen des XML-Dokumentes auswählen, in welches die Daten gespeichert werden sollen. Der Dateiname muss die Erweiterung „.xml“ haben. Ist dies nicht der Fall wird eine entsprechende Meldung angezeigt. Der Export wird erst gestartet, wenn ein gültiger Dateiname angegeben wurde.	

Fehlermeldung, falls der Datei nicht die Endung „.xml“ hat.	
Nach erfolgreichem Export wird eine Erfolgsmeldung angezeigt. Es wird auch der Namen und Speicherort des XML-Dokumentes angezeigt.	
Tritt während des Exportes ein Fehler auf, wird das fehlerhafte XML-Dokument gelöscht und diese Fehlermeldung angezeigt.	

2.6. XQuery: Abfrage auf ein gespeichertes XMLSchema absetzen

Durch Klicken auf den „XQuery“-Tab wird die GUI zum Eingeben einer Abfrage (XQuery) sowie des Anzeigens des Abfrageergebnisses geöffnet.

Auswahl einer Mappingvariante durch Klicken auf die entsprechende Zeile in der Tabelle.	
In Feld links unten kann nun eine gültige XQuery eingegeben werden. Eine Zieldatei ist dabei nicht anzugeben. **Achtung:** Als Platzhalter für den Dateinamen ist NUR „$doc“ gültig !!! (siehe Abbildung) Im rechten Fenster wird das Resultat der XQuery angezeigt	

Ist die XQuery ungültig (siehe Abbildung) oder tritt ein anderer Fehler auf (z.B. Unterbrechung der Datenbankverbindung) wird eine Fehlermeldung angezeigt	
Das Abfrageergebnis kann durch Klicken auf den „Save XQuery result“-Button in eine beliebige Datei exportiert werden. Hierfür wird ein Dialog geöffnet, in welchem man Dateiname und Speicherort auswählen kann. Nach erfolgreichem Speichern wird eine Erfolgsmeldung angezeigt. Sie enthält auch den Namen und Speicherort der Datei.