Hochschule für Technik und Wirtschaft Dresden
Fakultät Geoinformation
Masterstudiengang Geoinformation und Management

Masterarbeit

Entwicklung eines XSchemas für das webbasierte Informationssystem OpenInfRA

Eingereicht von
Lisa Henker
Seminargruppe: 11/063/71
Matrikelnummer: 33256

1. Gutachter: Herr Prof. Dr.-Ing. F. Schwarzbach
2. Gutachter: Herr Dr. F. Schäfer

Eingereicht am: 08.11.2013

Erklärung über die eigenständige Erstellung der Arbeit

Hiermit erkläre ich, dass ich die vorgelegte Arbeit mit dem Titel

„Entwicklung eines XSchemas für das webbasierte Informationssystem OpenInfRA“

selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit als solche und durch Angabe der Quelle gekennzeichnet habe. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.

Mir ist bewusst, dass die Hochschule für Technik und Wirtschaft Dresden Prüfungsarbeiten stichprobenartig mittels der Verwendung von Software zur Erkennung von Plagiaten überprüft.

_____ _____

Ort, Datum Unterschrift Student

Inhaltsverzeichnis

Abkürzungsverzeichnis

CIDOC	**C**omité **i**nternational pour la **doc**umentation (*engl*. International Committee for Documentation)
CRM	**C**onceptual **R**eference **M**odel
DAI	**D**eutsches **A**rchäologisches **I**nstitut
DTD	**D**ocument **T**ype **D**efinition
GMD	**G**eographic **M**eta**D**ata
GML	**G**eography **M**arkup **L**anguage
ID	**Id**entifier
ISO	**I**nternational **O**rganization for **S**tandardization
ISO/TS	**T**echnical **S**pecification
KML	**K**eyhole **M**arkup **L**anguage
OGC	**O**pen **G**eospatial **C**onsortium
OMG	**O**bject **M**anagement **G**roup
OpenInfRA	**Open Inf**ormation System for **R**esearch in **A**rchaeology
UML	**U**nified **M**odeling **L**anguage
URI	**U**niform **R**esource **I**dentifier
UUID	**U**niversal **U**nique **I**dentifier
SKOS	**S**imple **K**nowledge **O**rganization **S**ystem
SQL	**S**tructured **Q**uery **L**anguage
W3C	**W**orld **W**ide **W**eb **C**onsortium
XLink	**X**ML **Link**ing Language
XMI	**X**ML **M**etadata **I**nterchange
XML	E**x**tensible **M**arkup **L**anguge
XPath	**X**ML **Path** Language
XPointer	**X**ML **Pointer** Language
XSchema	**X**ML **Schema**
XSD	**X**ML **S**chema **D**efinition
XSL	E**x**tensible **S**tylesheet **L**anguage
XSLT	E**x**tensible **S**tylesheet **L**anguage **T**ransformation

1 Einleitung

Jährlich wird vom Deutschen Archäologischen Institut (DAI) zusammen mit nationalen und internationalen Wissenschaftseinrichtungen eine Vielzahl von Daten bei Feldforschungen aus unterschiedlichen Ländern erhoben. Diese Datenmengen werden heutzutage größtenteils in digitaler Form produziert. Die Verwaltung dieser Daten erfolgt dabei mit Hilfe von zwei Sofwarelösungen, dem webbasierte Informationssystem für Archäologie und Bauforschung, sowie dem Digitalen Dokumentationssystem für Feldforschungen. Aufgrund von einigen Defiziten dieser Softwarelösungen wird ein neues Dokumentationssystem für archäologische Feldforschungsprojekte, mit dem Namen OpenInfRA, geschaffen. Die Abkürzung OpenInfRA steht dabei für *Open Information System for Research in Archaeology.*[1]Zielsetzung ist dabei

> *„der Entwurf und die Implementierung eines Dokumentationssystemes, das in verschiedenen archäologischen, althistorischen, architekturgeschichtlichen oder landeskundlichen Feldforschungsprojekten eingesetzt werden kann – sowohl von Projekten, die weltweit durch das DAI durchgeführt werden, als auch für Forschungen, die andere Institutionen (z.B. Universitäten, Akademien, unabhängige Einrichtungen, ausländische Kooperationspartner) unternehmen."*[2]

Als Grundlage für die Entwichlung von OpenInfRA wurde ein konzeptuelles Anwendungsschema in Form eines UML Diagramms erarbeitet. Die Datenhaltung erfolgt in OpenInfRA in einer objektrelationalen Geodatenbank. Auf Basis dieses UML Anwendungsschemas wird innerhalb der vorliegenden Arbeit ein XML Schema abgeleitet. Anschließend erfolgt ein Export der Daten aus der Datenbank in ein Format, welches diesem XSchema entspricht. Weiterhin wird eine Transformation der Daten in das Format der CIDOC CRM vorgenommen. Die Prozesse, die zum Erreichen dieser Ergebnisse benötigt werden, sind in nachfolgenden Kapiteln beschrieben. Dabei werden zunächst die gegeben Ausgangsdaten, die als Grundlage aller nachfolgenden Schritte bereitstehen, erläutert. Nachfolgend wird dargestellt, nach welchen Regeln das XML Schema generiert wird um folglich die Schritte zu beschreiben, die nötig sind um die Daten der Datenbank in ein XML Dokument zu überführen, das gültig zum diesem XSchema ist. Anschließend wird die Transformation eines Datensatzes in das CIDOC CRM Format beschrieben. Schlussendlich erfolgt ein Ausblick auf mögliche weiterführende Schritte, die in Anlehnung an diese Arbeit umgesetzt werden können.

---

[1] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.5

[2] Grobkonzept OpenInfRA (23.07.2013) s.23

2 Ausgangsdaten

Zum Umsetzen dieser Aufgabe sind entsprechende Ausgangsdaten gegeben. Nachfolgend ist zum einen das UML Anwendungsschema des Informationssystems OpenInfRA und zum anderen die, aus diesem Schema abgeleitete, Datenbank dargestellt. Die Beschreibung der Datenbank, sowie des UML Anwendungsschemas richten sich nach dem aktuellen Stand zum Zeitpunkt der Anfertigung dieser Arbeit[3]. Änderungen, die sich im Nachhinein ergeben haben, werden nicht dargestellt. Erläutert werden alle Punkte, die für diese Abeit wichtig sind.

2.1 UML Anwendungsschema

Das UML Anwendungsschema des webbasierten Informationssystems OpenInfRA weist einen hohene Abstraktionsgrad auf und ist die Entwicklungsgrundlage für dieses Informationssystem. Das UML Anwendungsschema ist unterteilt in die Datensicht (Abbildung 1) und die komplexen Datentypen (Abbildung 2).

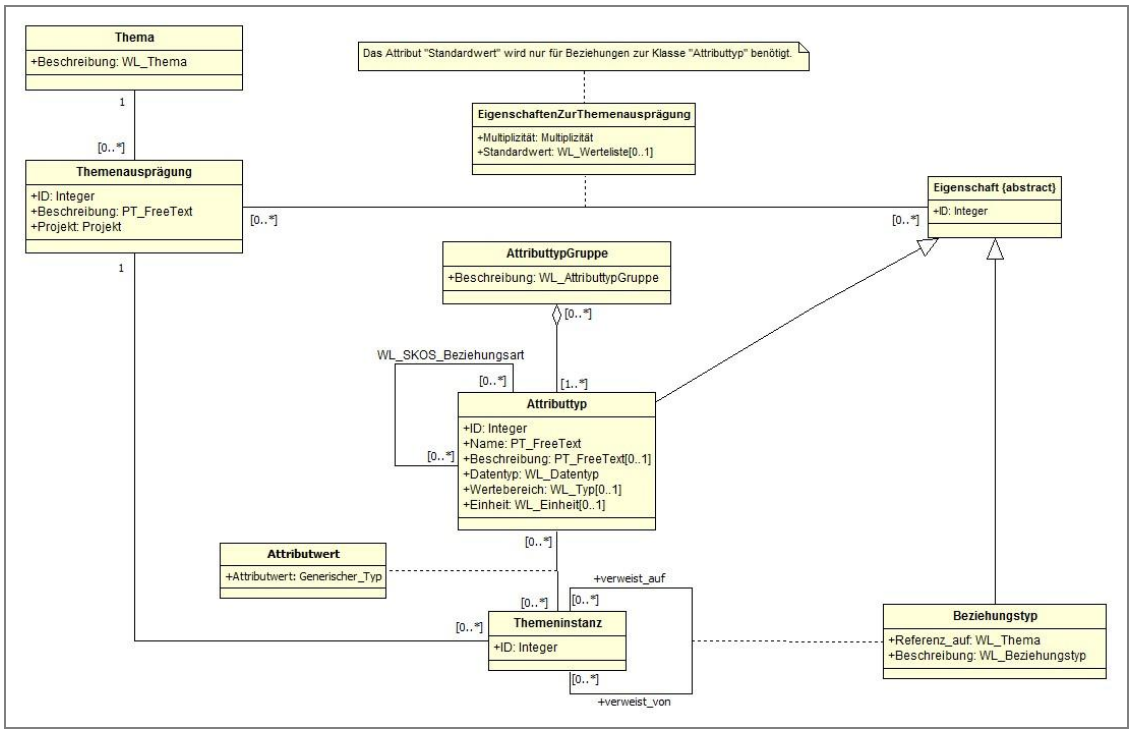


Abbildung 1: UML Anwendungsschema – Datensicht[4]

[3] Arbeitsstand vom 23.07.2013

[4] Vgl. Grobkonzept OpenInfRA (23.07.2013) s. 54

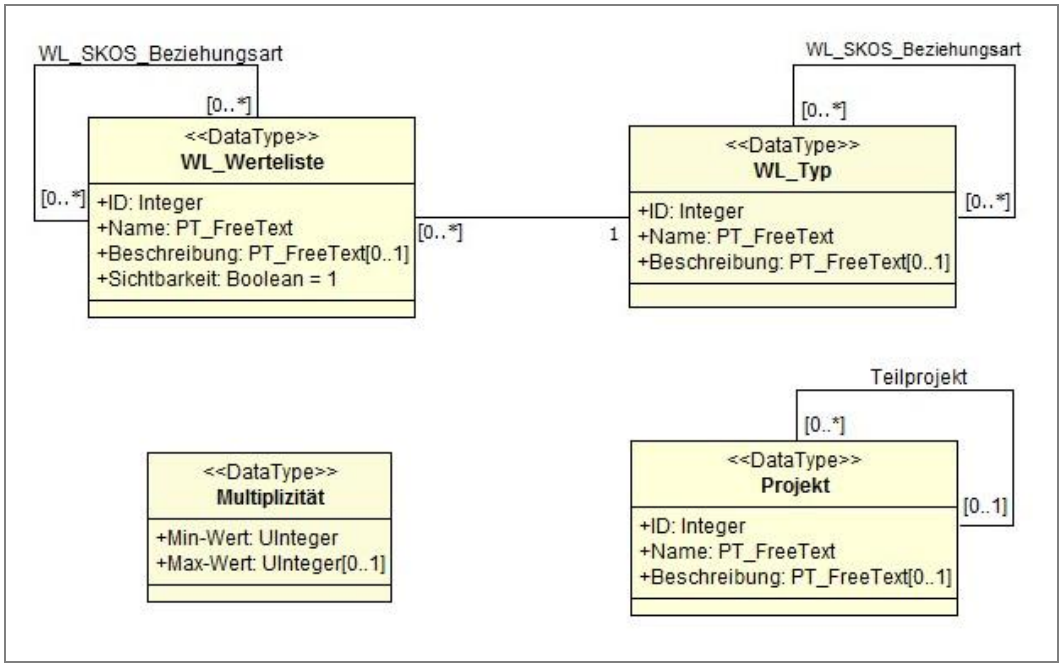


Abbildung 2: UML Anwendungsschema - komplexe Datentypen[5]

Das komplette UML Anwendungsschema ist in angemessener Größe unter Anlage A.1 vorzufinden. Nachfolgend werden die in obigen Abbildungen dargestellten Klassen, Datentypen und Beziehungen erläutert.

Thema

Die Klasse *Thema* beschreibt alle fachlich relevanten Themen. Konkrete Themen sind dabei in der Werteliste enthalten.[6]

Beispiel: Instanz **Gebäude**

Themenausprägung

Für jedes Thema gibt es keine, eine oder mehrere Themenausprägungen. Eine Instanz der Klasse *Themenausprägung* wird durch eine entsprechende Beschreibung und eine ID beschrieben. Die ID ist ein numerischen Identifikator, der automatisch vom System generiert wird. Jede Instanz dieser Klasse gehört einem Projekt an. Das Attribut *Projekt* weist dabei einen Datentyp *Projekt* auf, welcher nachfolgend erläutert wird. Eine Instanz der Klasse *Themenausprägung* kann selbstbezogene Eigenschaften

[5] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.55
[6] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.57

(Attributtypen) und bzw. oder fremdbezogene Eigenschaften (Beziehungstypen) aufweisen.[7]

Beispiel: Instanz **Raum**

Eigenschaft

Die abstrakte Klasse *Eigenschaft* ist die Oberklasse bzw. Superklasse der Klassen *Attributtyp* und *Beziehungstyp* und weist lediglich das Attribut *ID* auf, welches an die beiden Unterklassen bzw. Subklassen vererbt wird. Weiterhin besitzt diese Klasse eine Beziehung zur Klasse *Themenausprägung*.[8]

EigenschaftenZurThemenausprägung

Die Klasse *EigenschaftenZurThemenausprägung* ist eine Assoziationsklasse, über die der Themenausprägung die selbstbezogenen oder fremdbezogenen Eigenschaften zugeordnet werden. Den selbstbezogenen Eigenschaften wird so zum einen das Attribut *Multiplizität* zugeordnet, das definiert, wie oft die jeweilige Eigenschaft auftreten kann, zum anderen wird das Attribut *Standartwert* zugewiesen. Dieses Attribut kann optional aufgeführt werden und setzt einen Eintag aus der Werteliste als Standartwert für die entsprechende Eigenschaft fest. Fremdbezogene Eigenschaften weisen hingegen nur das Attribut *Multiplizität* auf.[9]

Beispiel: Ein **Raum** hat eine bestimmte **Grundfläche** (Multiplizität: 1).

Beziehungstyp

Mit Hilfe der Assoziationsklasse *Beziehungstyp* werden Beziehungen zwischen Themeninstanzen des selben oder eines anderen Themas beschrieben. Dazu stehen die Attribute *referenz_auf*, zur Referenzierung des entsprechenden Themas und *Beschreibung*, zur Benennung der Beziehung, bereit.[10]

Beispiel: Raum Nr. 1 **hat Zugang** zu Raum Nr. 2 aus dem selben Thema Gebäude oder Raum Nr. 1 **beinhaltet** Stuhl Nr. 1 aus dem Thema Mobiliar.

[7] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.57
[8] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.58
[9] Vgl. Grobkonzept OpenInfRA (23.07.2013) s 57
[10] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.58

Attributtyp

Diese Klasse definiert Attributtypen, die benötigt werden um ein Thema zu beschreiben. Jede Instanz weist einen Namen und einen Datentyp auf. Ein Datentyp kann entweder ein einfacher, wie z.B. „*String“* oder ein komplexer Typ sein und beschreibt in welchem Datentyp der entsprechende Attributwert angegeben wird. Weiterhin besteht die Möglichkeit, dass einer Instanz dieser Klasse eine Beschreibung, ein Wertebereich und eine Einheit zugeordnet werden kann.[11]

Beispiel: Instanz **Grundfläche** – Die Grundfläche eines Raumes mit der Einheit „m^2“ und dem Datentyp „*Double“*

AttributtypGruppe

Die Klasse AttributtypGruppe dient der Vorgruppierung von Attributtypen nach funktionaler oder fachlicher Sicht.[12]

Themeninstanz

Eine Instanz der Klasse *Themeninstanz* beschreibt ein bestimmtes Objekt des Themas, welchen es angehört. Eine Instanz dieser Klasse weist lediglich das Attribut *ID* auf. Eine Themeninstanz kann, wie zuvor beschrieben, eine Beziehung zu anderen Themeninstanzen aufweisen. Weiterhin kann eine Instanz der Klasse *Themeninstanz* für jeden Attributtyp, der dem Thema zugeordnet wurde, einen Attributwert aufweisen.[13]

Beispiel: Instanz **Raum Nr. 1**

Attributwert

Die Klasse *Attributwert* ist eine Assoziationsklasse, die der Verbindung zwischen einer Themeninstanz und einem bestimmten Attributtyp einen konkreten Attributwert zuordnet. Der entsprechende Attributtyp muss dabei dem Thema zugeordnet sein. Das Attribut *Attributwert* zeigt dabei den konkreten Wert mit dem Datentyp *„Generischer Typ“* auf. Dieser Typ besagt, dass sich der Datentyp des Attributwertes entsprechend des *Datentyp* Attributs der zugehörigen Attributtypinstanz anpassen muss.[14]

Beispiel: Attributwert **30,5** mit dem Datentyp *„Double“*

[11] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.58
[12] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.58
[13] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.58
[14] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.59

Multiplizität

Der Datentyp *Multiplizität* ist ein komplexer Datentyp, der über zwei Attribute das minimale und maximale Vorkommen einer fremdbezogenen oder selbstbezogenen Eigenschaft eines Themas beschreibt. Der Datentyp der unteren und oberen Grenze ist „*UInteger“* und besagt, dass der Grenzwert eine positive Ganzzahl sein muss.[15]

Projekt

Über den komplexen Datentyp *Projekt* können Projekte für Themen beschrieben werden. Ein Projekt kann dabei aus einem oder mehreren Teilprojekten bestehen bzw. ein Projekt kann ein Teilprojekt eines Projektes sein. Definiert wird das einzelne Projekt über eine ID, einen Namen und wahlweise über eine Beschreibung.[16]

WL_Typ

Der komplexe Datentyp *WL_Typ* definiert Wertelisten bzw. Wertelistentypen. Über die Assoziation zum komplexen Datentyp *WL_Werteliste* werden den einzelnen Wertelistentypen die entsprechenden Wertelisteneinträge zugeordnet. Eine Werteliste wird über die Attribute *ID*, *Name* und *Beschreibung* definiert. Jede Werteliste bzw. jeder Wertelistentyp kann in Beziehung zu einem anderen Wertelistentyp stehen. Diese Beziehung weist dabei eine SKOS Beziehungsart auf.[17]

Beispiel: **Farbe**

WL_Werteliste

Der komplexe Datentyp *WL_Werteliste* stellt alle vordefinierten Attributwerte bereit. Die Wertelisteneinträge sind dabei unabhängig vom Wertelistentyp. Die Festlegung dieses Typs erfolgt erst duch die Assoziation zum komplexen Datentyp *WL_Typ*. Der Datentyp *WL_Werteliste* weist so alle Einträge jeder Werteliste bzw. jedes Wertelistentyps auf. Ein Wertelistenwert wird über die Attribute *Beschreibung*, *Name* und *ID* beschrieben. Über das Attribut *Sichtbarkeit* wird geregelt, ob der jeweilige Eintrag in dem bestimmten Projekt zutreffend ist oder nicht. Wertelisteneinträge können untereinander über eine SKOS Beziehungsart in Beziehung stehen.[18]

Beispiel: Wertelisteneinträge **Rot**, **Grün** und **Blau**

---

[15] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.57
[16] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.57
[17] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.56
[18] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.56

PT_FreeText

Der komplexe Datentyp *PT_FreeText* ist ein wichtiger Datentyp innerhalb des UML Anwendungsschemas, da alle Namen und Beschreibungen diesen Typ aufweisen. Mit Hilfe dieses Datentyps können Attributwerte mehrsprachig geführt werden. Dabei werden lokalisierte Zeichenketten in unterschiedlichen Sprachumgebungen über die Angabe von Ländercode, Sprachcode und Zeichenkodierung definiert.

2.2 Datenbank

Die Datenhaltung erfolgt innerhalb eines Datenbanksystems. Das Datenbanksystem ist objektrelational und beinhaltet mehrere Projektdatenbanken und eine Systemdatenbank. Dabei wird für jedes Projekt, sowie dessen Teilprojekte eine Projektdatenbank angelegt, die das unter Abbildung 1 und Abbildung 2 dargestellte Anwendungsschema vollständig abbilden. Daten, die nicht projektspezifisch sind werden in der Systemdatenbank gespeichert. Zur Umsetzung dieser Aufgabe ist das Create-Skript für die Erstellung einer Projektdatenbank in PostgreSQL/PostGIS vorliegend. Dabei werden für alle, im UML Anwendungsschema angegebenen Klassen entsprechende Tabellen erzeugt, mit Ausnahme der Klassen *Thema* und *AttributtypGuppe*. Für diese Klassen werden keine Tabellen generiert. Die Themen und AttributtypGruppen werden über ihren Wertelisteneintrag identifiziert. Diese Datenbank wurde lokal Implementiert. Neben den Klassen, die im UML Anwendungsschema dargestellt sind, weist diese Datenbank noch Klassen zur Umsetzung des Datentyps *PT_Freetext* auf. Diese Klassen sind in nachfolgendem Implementierungsschema abgebildet (Abbildung 3).[19]

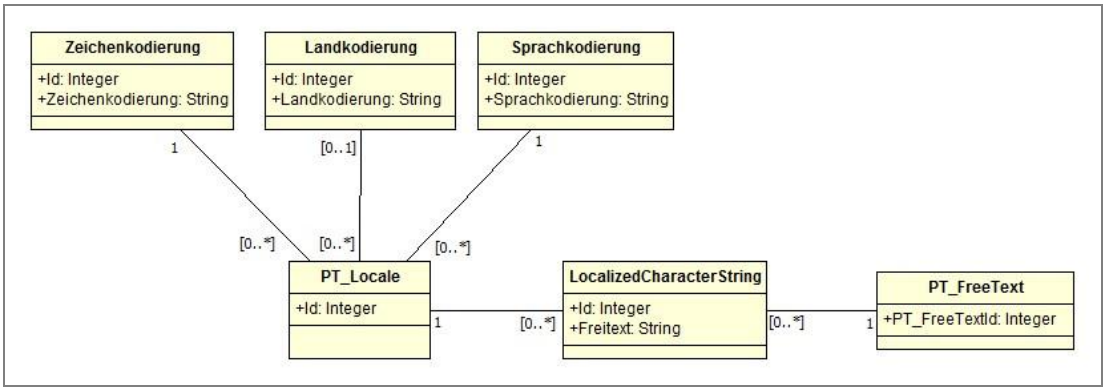


Abbildung 3: Implementierungsschema PT_FreeText[20]

[19] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.75, 76, 77

[20] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.77

Das oben aufgeführte Schema stellt eine Modifikation des Anwendungsschemas des Datentyps *PT_FreeText* aus der Norm ISO 19139 dar. Jeder lokalisierten Zeichenkette wird eine Sprachumgebung (PT_Locale) zugeordnet. Eine Instanz der Klasse *PT_FreeText* weist anschließend alle identischen Zeichenketten in unterschiedlichen Sprachumgebungen auf und fasst sie zusammen.[21] Weiterhin beinhaltet die Projektdatenbank eine Klasse zur Beschreibung von Geometrien. Diese Klasse weist jedoch keine Einträge auf, da Geometrieeigenschaften zu diesem Zeitpunkt nicht implementiert sind.

[21] Vgl. Grobkonzept OpenInfRA (23.07.2013) s.59,77

3 Ableitung XSchema

Aus dem zuvor erläuterten UML Anwendungsschema wird folglich ein XSchema abgeleitet, um den Austausch der Datenbankeinträge XML-basiert ermöglichen zu können. Zuvor ist es jedoch wichtig, den Begriff XSchema zu klären um anschließend die Möglichkeiten der Erstellung dieses Schemas darzustellen. Im nachfolgenden Kapitel werden alle, zur Umsetzung der Aufgabenstellung benötigten XML Techniken erläutert.

3.1 Definition XSchema

Ein XML Schema, kurz auch XSchema genannt, ist eine XML-basierte W3C Recommendation und beschreibt die Struktur eines XML Dokuments. Häufig wird statt XML Schema auch die Abkürzung XSD (XML Schema Definition) verwendet. Ein solches Schema deklariert Elemente und Attribute, die im zugehörigen XML Dokument, auch als Instanzdokument bezeichnet, vorhanden sein dürfen bzw. müssen. Weiterhin kann definiert werden, wie oft Elemente vorkommen können, welche Unterelemente bzw. Kinderelemente sie aufweisen und in welcher Reihenfolge diese aufgeführt werden. Zusätzlich kann festgelegt werden, welchen Inhalt ein Element besitzt. Ein weiterer wichtiger Punkt eines XML Schemas ist die Möglichkeit vordefinierte Datentypen für Attribute und Elemente auswählen bzw. eigene Datentypen definieren zu können. Ein XML Schema ist immer ein eigenständiges Dokument, auf welches das Instantdokument verweist. Dieses Instanzdokument kann gegen das entsprechende Schema validiert werden, um so die Gültigkeit der Datei zu überprüfen. Diese Verbindung zwischen Schema- und Instanzdokument wird anhand des nachfolgenden Beispiels erläutert.[22]

XML-Schema[23]

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.person.org">
   <xs:element name="Person" type="PersonType"/>
   <xs:complexType name="PersonType">
      <xs:sequence>
         <xs:element name="Vorname" type="xs:string"/>
         <xs:element name="Name" type="xs:string"/>
         <xs:element name="Alter" type="xs:short"/>
         <xs:element name="Geschlecht" type="xs:string"/>
```

[22] Vgl. Amrhein (2011) s.18

```
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

Instanzdokument[24]

```
<?xml version="1.0" encoding="UTF-8"?>
<Person xmlns="http://www.person.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.person.org person.xsd">
   <Vorname>Max</Vorname>
   <Name>Mustermann</Name>
   <Alter>44</Alter>
   <Geschlecht>Männlich</Geschlecht>
</Person>
```

Zunächst wird in beiden Dokumenten die XML Version und die entsprechende Kodierung angegeben. Nachfolgend wird im XSchema das Element `xs:schema` aufgeführt, welches das Wurzelelement jedes XML Schemas ist. Die in diesem Element definierten Eigenschaften gelten für alle Unterelemente und deren Attribute dieses Schemas. Alle direkten Unterelemente bzw. Kindelemente des `xs:schema` Elements sind global deklarierte Elemente, die anschließend als Wurzelelemente im Instanzdokument vorkommen können. Im obigen Beispiel ist das Element mit dem Namen `Person` global deklariert. Dieses Element verweist auf einen Datentyp, welcher nachfolgend definiert ist. Dieser `PersonType` Typ ist ein komplexer Datentyp. Diesem Typ ist es erlaubt Unterelemente und Attribute zu enthalten. Diese Unterelemente sind nicht global bekannt und sind somit lokal deklarierte Elemente. Weiterhin sind diese auch im Instanzdokument wieder als Unterelemente den globalen Elements vorzufinden, wie im obigen Beispiel ersichtlich ist. Es gibt verschiedene Möglichkeiten Bedingungen für diese Unterelemente in komplexen Datentypen zu spezifizieren. Dazu werden sogenannte Auswahlgruppen verwendet, die unbenannte Gruppen darstellen. Dabei stehen die drei Gruppen

- `sequence`,
- `all` und
- `choice`

zur Verfügung. Das `sequence` Element der Auswahlgruppe schreibt vor, dass alle

[23] Vgl. Stein (2013) s.173
[24] Vgl. Stein (2013) s.187

Unterelemente im Instanzdokument dieselbe Reihenfolge aufweisen müssen, wie im komplexen Typ des XML Schemas definiert wurde. Werden Unterelemente stattdessen über den Indikator `all` aufgeführt, können diese in Instanzdokument in beliebiger Reihenfolge auftretet, jedoch höchstens einmal. Bei Angabe des `choice` Tags kann nur eines der deklarierten Unterelemente vorkommen. Wie oft ein Element dabei auftreten darf, wird durch die Attribute `minOccurs` und `maxOccurs` bestimmt.[25]

```
<xs:element name="Vorname" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
```

Der Standardwert dieser Attribute beträgt „1“. Werden diese Attribute nicht aufgeführt, erscheint ein Element genau einmal im Instanzdokument.
In einem komplexen Datentyp können Elementen, neben Unterelementen, auch Attribute zugeordnet werden. Die Attribute werden dabei immer nach der Auswahlgruppe aufgeführt.

```
<xs:complexType name="PersonType">
   <xs:sequence>
       <xs:element name="Vorname" type="xs:string"/>
       <xs:element name="Name" type="xs:string"/>
       <xs:element name="Alter" type="xs:short"/>
       <xs:element name="Geschlecht" type="xs:string"/>
   </xs:sequence>
   <attribute name="id" type="ID"/>
</xs:complexType>
```

Neben der zuvor aufgeführten Möglichkeit Elementen entsprechende komplexe Datentypen zuzuordnen, besteht eine weitere Option nach der diese Zuordnung durchgeführt werden kann:

```
<xs:element name="Person">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Vorname" type="xs:string"/>
         <xs:element name="Name" type="xs:string"/>
         <xs:element name="Alter" type="xs:short"/>
         <xs:element name="Geschlecht" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

[25] Vgl. Amrhein (2011) s.24, 27, 28

Dabei wird ein Datentyp ohne namentliche Bezeichnung definiert, der sozusagen anonym ist und auf den andere Elemente nicht referenzieren können. Diese Struktur ist dann anzuwenden, wenn ein komplexer Typ nur von einem Element verwendet wird. Weisen jedoch mehrere Elemente auf denselben Typ ist es von Vorteil einen benannten Datentyp zu definieren, auf den referenziert werden kann, um so mehrfaches beschreiben desselben Datentyps zu vermeiden. Neben komplexen Datentypen können Elemente des XSchemas auch simple Datentypen aufweisen. Diesen Typen ist es nicht erlaubt Attribute oder Unterelemente zu deklarieren, jedoch kann der Wert, den die Elemente im Instanzdokument annehmen können, eingeschränkt werden. Das erfolgt über sogenannte Facetten. Dabei können u.a. Längen von Zeichenketten oder feste Werte für Zeichenketten vorgegeben werden, die angenommen werden dürfen.[26]

```
<xs:complexType name="PersonType">
   <xs:sequence>
      <xs:element name="Vorname" type="xs:string"/>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Alter" type="xs:short"/>
      <xs:element name="Geschlecht">
         <xs:simpleType>
            <xs:restriction base="xs:string">
               <xs:enumeration value="männlich"/>
               <xs:enumeration value="weiblich"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:element>
   </xs:sequence>
</xs:complexType>
```

Einige einfache, vordefinierte Datentypen, die zur Umsetzung der Aufgabenstellung benötigt werden, sind in nachfolgender Tabelle (Tabelle 1) dargestellt.

Tabelle 1: vordefinierte Datentypen eines XML Schemas[27]

Datentyp	Beschreibung
boolean	repräsentiert die logischen Werte *true* und *false*
ID	repräsentiert einen, innerhalb des Dokumentes eindeutigen, Identifikator für ein Element, der niht rein numerisch sein darf

[26] Vgl. Amrhein (2011) s.29
[27] Vgl. Jakobs (2011) s.7

IDREF	Verweis auf einen Identifikator innerhalb eines Dokumentes
integer	repräsentiert ganzzahlige numerische Werte (positiv und negativ)
string	repräsentiert Zeichenketten
unsignedInt	repräsentiert positive ganzahlige numerische Werte

Die deklarierten Elemente eines XML Schemas können, neben den zuvor beschriebenen unbenannten Gruppen, auch in benannten Gruppen angeordnet werden. Dabei wird innerhalb eines `group` Elements eine Auswahlgruppe definiert. Diesem `group` Element kann ein Name zugeordnet werden über den anschließend auf die definierte Gruppe referenziert werden kann. Wie in nachfolgendem Beispiel zu sehen, wird zunächst die Gruppe `Name` definiert.

```
<xs:group name="Name">
  <xs:sequence>
    <xs:element name="Vorname" type="xs:string"/>
    <xs:element name="Nachname" type="xs:string"/>
  </xs:sequence>
</xs:group>
```

Auf diese Gruppe kann nun referenziert werden.

```
<xs:complexType name="PersonType">
   <xs:sequence>
      <xs:group ref="Name"/>
      <xs:element name="Alter" type="xs:short"/>
      <xs:element name="Geschlecht" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

Ein weiterer wichtiger Bestandteil eines XML Schemas sind Namensräume. Diese werden im Wurzelelement `xs:schema` deklariert. Ein XML Schema ist dabei ein Vokabular von Elementdeklarationen und Typdefinitionen, die zu einem bestimmten Zielnamensraum gehören. Die global deklarierten Elemente sind dabei diesem Zielnamensraum zugeordnet und müssen im Instanzdokument diesem Namensraum zugeordnet sein. Das bedeutet sie müssen qualifiziert sein. Diese Zuordnung erfolgt durch Angabe eines Präfixes, welcher an den Namensraum gebunden ist. Lokal deklarierte Elemente

können entweder qualifiziert oder unqualifiziert sein. Diese Qualifizierung wird über das Attribut `elementFormDefault` im Wurzelelement festgelegt. Ist ein lokales Element qualifiziert, so muss auch dieses im Instanzdokument mit einen entsprechenden Präfix versehen werden. Durch diese Namensräume ist es möglich zwischen verschiedenen Vokabularen zu unterscheiden. Weiterhin ist es möglich durch Angabe des entsprechenden Namensraums, Elemente und Datentypen aus anderen Vokabularen in dem eigenen XML Schema zu verwenden.[28]

Die Verknüpfung vom Instanzdokument zum zugehörigen XML Schema erfolgt über das Attribut `schemaLocation` im Wurzelelement des Instanzdokumentes.

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.person.org person.xsd"
```

Dieses Attribut wird über das Präfix `xsi` qualifiziert und beinhaltet zwei Parameter. Der erste Parameter gibt den Namensraum des XML Schemas an und der zweite Parameter den absoluten oder relativen Pfad zu dessen Speicherort. Erst wenn einem XML Dokument das entsprechende XSchema zugeordnet ist, ist es möglich dieses Dokument gegen das XSchema zu validieren.[29]

Zusammenfassend ist zu sagen, dass eine XML Schema zum einen aus Elementdeklarationen und zum anderen aus Datentypdefinitionen besteht. Folgende Abbildung (Abbildung 4) stellt den Zusammenhang zwischen Elementen und Datentypen noch einmal grafisch dar.

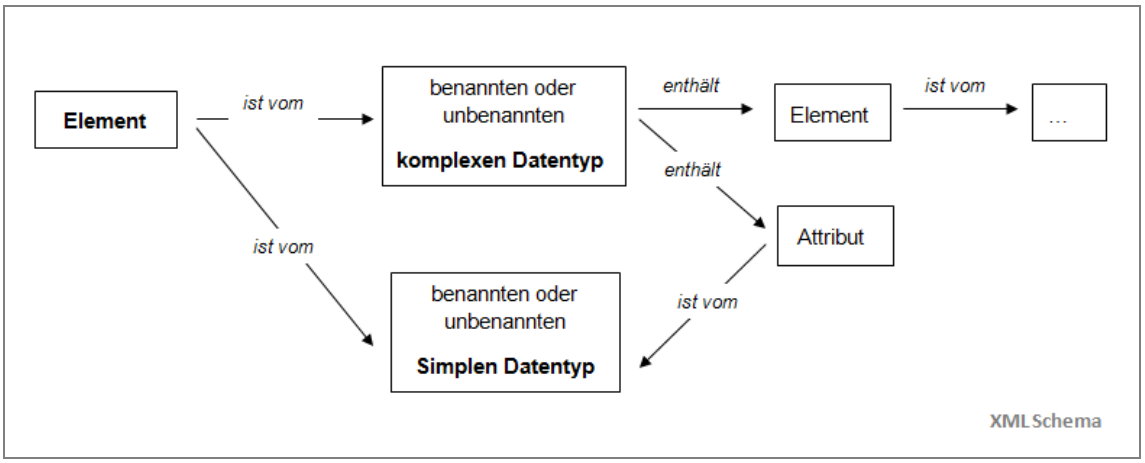


Abbildung 4: Element - Datentyp Beziehung

[28] Vgl. Stein (2013) s.186
[29] Vgl. Stein (2013) s.188

Die Vorteile eines XSchemas liegen also in der Möglichkeit eigene Datentypen definieren und auf eine Vielzahl von vordefinierten Datentypen zugreifen zu können, sowie in der Unterstützung von Namensräumen und dem Fakt, dass das Schema selbst in der XML Sprache geschrieben ist.

3.2 Implementierungsschema

Damit aus dem, in Kapitel 2.1 beschriebenen, UML Anwendungsschema ein XML Schema abgeleitet werden kann, muss dieses Schema entsprechend angepasst werden. Das ist nötig, da das UML Anwendungsschema eine Datensicht beschreibt, die unabhängig vom Format ist, in welchem diese anschließend implementiert wird. Das angepasste Schema wird als Implementierungsschema bezeichnet. Dieses Schema ist entsprechend des Formates dargestellt, in welches es implementiert wird. Dabei werden Beziehungen, Klassen und Datentypen abgeändert, da diese in der vorliegenden Form nicht im XML Schema umgesetzt werden können. Das vollständige Implementierungsschema ist unter Anlage A.2 aufgeführt.

3.2.1 Assoziationsklassen

Sowohl die Klasse *Attributwert*, als auch die Klassen *Beziehungstyp* und *EigenschaftenZurThemenausprägung* sind, wie in Kapitel 2.1 beschrieben, Assoziationsklassen. Die Implementierung dieser Klassen im XML Format ist nicht möglich. Aus diesem Grund ist es notwendig die Assoziationsklassen in eigenständige Klassen aufzulösen. In nachfolgender Abbildung (Abbildung 5) wird die Auflösung grafisch verdeutlicht.

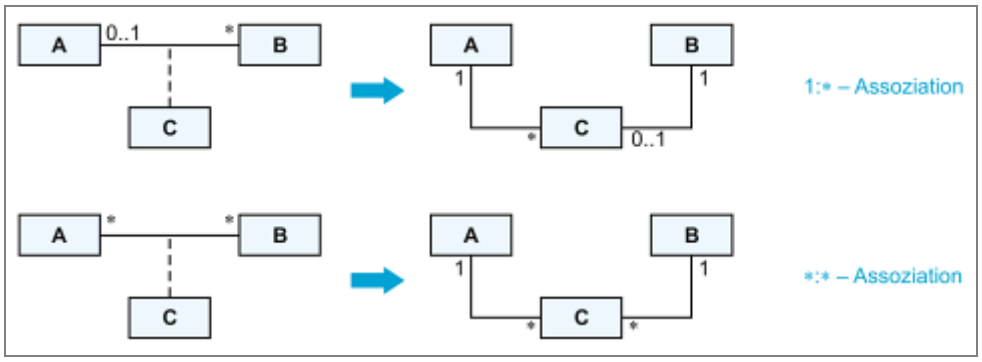


Abbildung 5: Auflösung einer Assoziationsklasse[30]

[30] Vgl. Balzert (2010) s.131

Diese Regeln zur Erzeugung eigenständiger Klassen werden zunächst auf die Klasse *Attributwert* angewendet. Dabei entsteht folgendes Ergebnis (Abbildung 7), dargestellt im Vergleich zur Ausgangssituation (Abbildung 6):

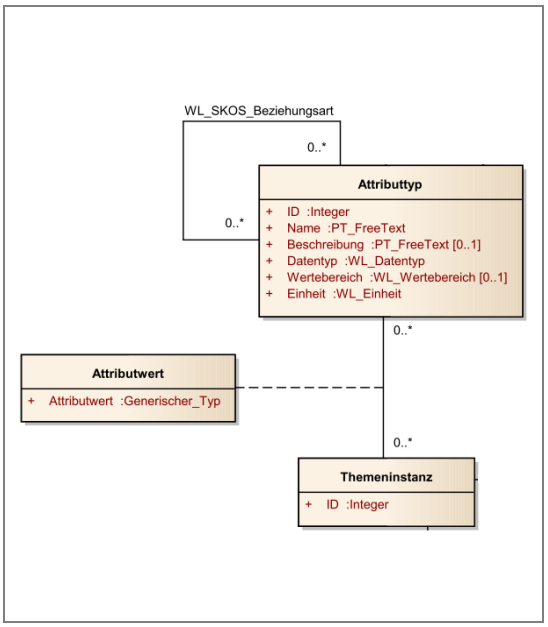


Abbildung 6: Assiziationsklasse

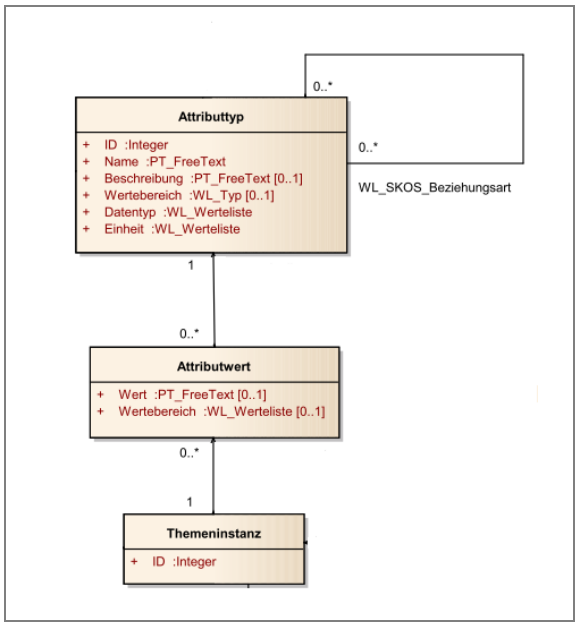


Abbildung 7: Aufgelöste Assoziationsklasse

Analog dazu könnte auch die Klasse *EigenschaftenZurThemenausprägung* in eine eigenständige Klasse aufgelöst werden. Aufgrund der vorliegenden Generalisierung muss zuvor jedoch ein weiterer Schritt zur Auflösung dieser Generalisierung durchgeführt werden. Dieser Schritt wird im nachfolgenden Kapitel erläutert.
Wegen der unterschiedlichen Aufgaben der drei Assoziationsklassen wird die Beziehungsklasse *Beziehungstyp* nicht nach oben dargestellten Methoden bearbeitet. Diese Klasse beschreibt die Beziehungen zwischen Themeninstanzen genauer und wird deshalb als Attribut dieser Beziehung implementiert. Genaueres ist in Kapitel 3.4.4 beschrieben.

3.2.2 Beziehungen

Assoziationen, Kompositionen und Aggregationen des UML Anwendungsschemas können im XML Schema dargestellt werden. Dabei ist es jedoch wichtig, dass diese Beziehungen Rollennamen besitzen, die die Beziehungsrolle beschreiben. Diese Rollennamen sind in dem XSchema nötig, da diese im Verlauf der Implementierung direkt als Elementnamen übernommen werden. Das gegebene UML weist größtenteils keine Rollenbezeichnungen auf. Deshalb ist es nötig eine Benennung vorzunehmen, bevor das Schema implementiert werden kann. Für die Rollennamen werden dabei Standartbezeichnungen bzw. allgemeine Namen vergeben um die Aussagen des Schemas

nicht zu verändern oder es zu verfälschen (Abbildung 8). Die Rollennamen beinhalten den Namen der Zielklasse. Weiterhin ist es möglich diese Bezeichnungen nachträglich anzupassen.

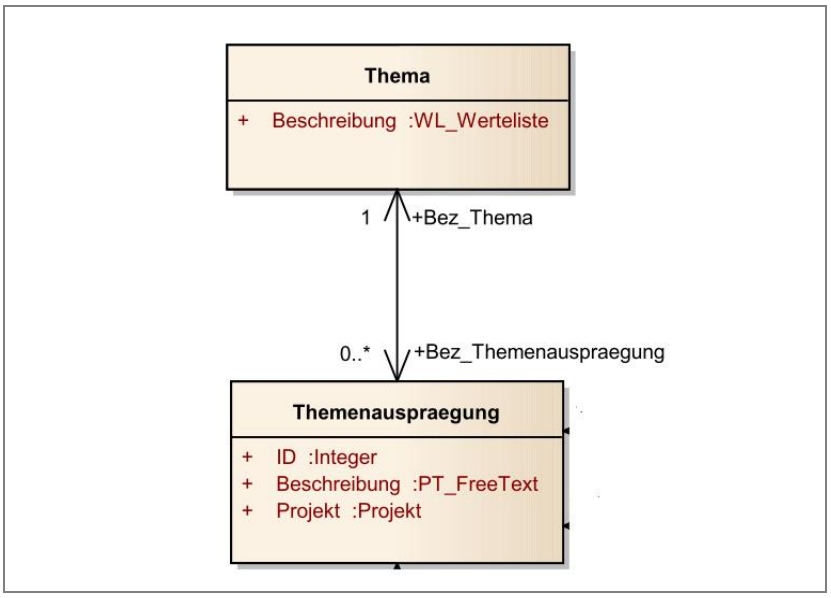


Abbildung 8: Assoziation mit Rollennamen

Innerhalb des gegebenen UML Anwendungsschemas ist weiterhin eine Generalisierung vorzufinden, bei der die Klassen *Beziehungstyp* und *Attributtyp* von der abstrakten Klasse *Eigenschaft* erben. Diese abstrakte Klasse vererbt dabei nicht nur das Attribut *id*, sondern auch die Beziehung zur Klasse *Themenausprägung* inklusive der daraus resultierenden Assoziationsklasse *EigenschaftenZurThemenausprägung*. Diese Klasse weist in Bezug auf die Klasse *Beziehungstyp* eine Einschränkung auf, da das Attribut Standartwert in Zusammenhang mit einem Beziehungstyp nicht auftritt. Um diese Struktur entsprechend im XML Schema abzubilden, wird die Generalisierung aufgelöst, indem die Eigenschaften der Superklasse in die Subklasse übertragen werden. Das ist auf die Umsetzung der Assoziationsklasse zurückzuführen. Diese muss, wie zuvor erwähnt, aufgelöst werden. Folglich würde die abstrakte Klasse *Eigenschaft* keine direkte Beziehung zu Klasse *Themenausprägung* mehr aufweisen. Diese weist dann nur eine Beziehung zur Klasse *EigenschaftenZurThemenausprägung* auf, die wiederum in einer Beziehung zur Klasse *Themenausprägung* steht (Abbildung 9).

Abbildung 9: Auflösung der Assoziationsklasse *EigenschaftenZurThemenausprägung*

Anschließend wird im XSchema, bei Anwenden der in Kapitel 3.4.4 beschriebenen Implementierungsregeln, nur das Attribut *ID*, sowie die Beziehung zur Klasse *EigenschaftenZurThemenausprägung* vererbt, jedoch nicht die Attribute dieser Klasse selbst. In Folge würden die Klassen *Attributtyp* und *Beziehungstyp* in Beziehung mit der gleichen Klasse (*EigenschaftenZurThemenausprägung*) stehen. Dadurch ist es nicht möglich das Auftreten des Attributes *Standartwert* im Zusammenhang mit der Klasse *Beziehungstyp* zu beschränken. Aus diesem Grund wird zuerst die Beziehung *Themenausprägung - Eigenschaft*, inklusive Assoziationsklasse, an die Subklassen vererbt. Dabei erfolgt eine Umbenennung der Assoziationsklassen, um die doppelte Existenz einer Klasse zu vermeiden (Abbildung 10).

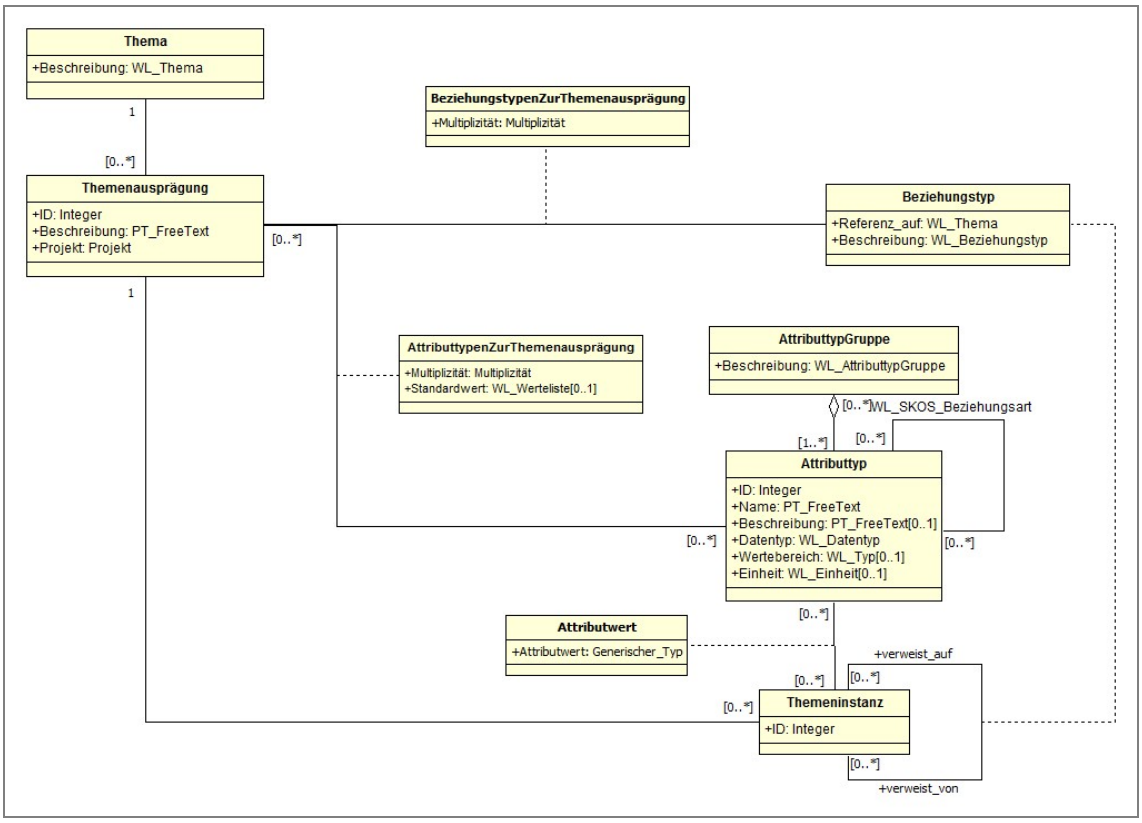


Abbildung 10: Vererbung von Assoziationsklassen

So entsteht zum einen eine Beziehung zwischen den Klassen *Themenausprägung* und *Attributtyp* mit der Assoziationsklasse *AttributtypenZurThemenausprägung* und zum anderen eine Beziehung zwischen den Klassen *Themenausprägung* und *Beziehungstyp* mit der Assoziationsklasse *BeziehungstypenZurThemenausprägung.* Erst an diesem Punkt werden die Assoziationsklassen, wie in Kapitel 3.2.1 beschrieben, aufgelöst (Abbildung 11).

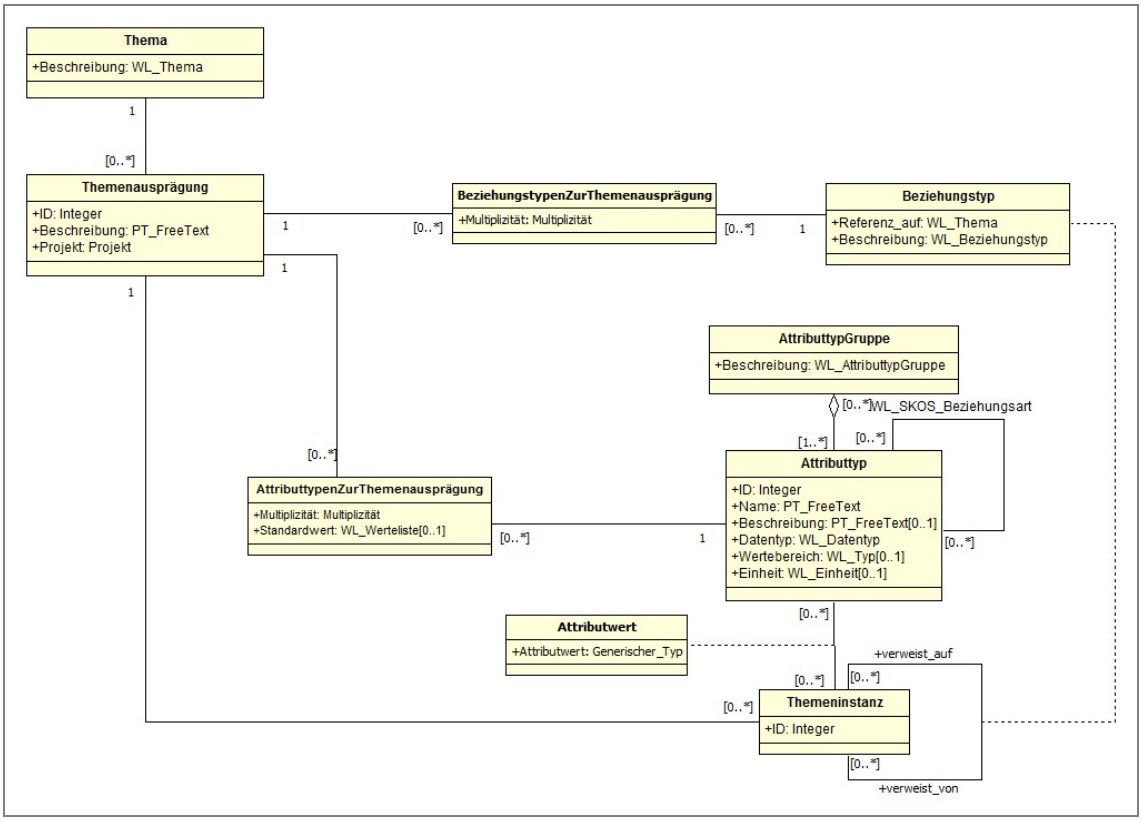


Abbildung 11: Aufgelöste Generalisierung

Die abstrakte Klasse *Eigenschaft* wird nicht im XSchema umgesetzt, da eine Implementierung von abstrakten Klassen nicht nötig ist, wenn die Generalisierung mittels Übertragung von Eigenschaften der Superklasse in die Subklasse vorgenommen wird.[31]

3.2.3 Datentypen

Die meisten Datentypen des UML Anwendungsschemas können ins XML Schema konvertiert werden. Die Attribute der Klasse Multiplizität weisen den Datentyp *UInteger* auf. Für diesen Typ gibt es keine mögliche Konvertierung. Aus diesem Grund ist es nötig einen Datentyp zu wählen der durch das XML Schema unterstützt wird. Dabei ist es wichtig einen Typ zu verwenden, der den gleichen Wertebereich definiert, um Probleme beim späteren Datenexport aus der Datenbank zu vermeiden und um ein XSchema zu generieren das mit dem UML Anwendungsschema übereinstimmt. Der numerische Datentyp *UInteger* beschreibt 32-Bit Ganzzahlen ohne Vorzeichen. Das heißt, er unterstützt alle Zahlen im Bereich von 0 bis 4.294.967.295.[32]

[31] Vgl. Norm ISO 19118 (2006) s.44

[32] Vgl. Microsoft Corporation (2013)

Nachfolgend sind alle vordefinierten Datentypen eines XML Schemas dargestellt (Abbildung 12).

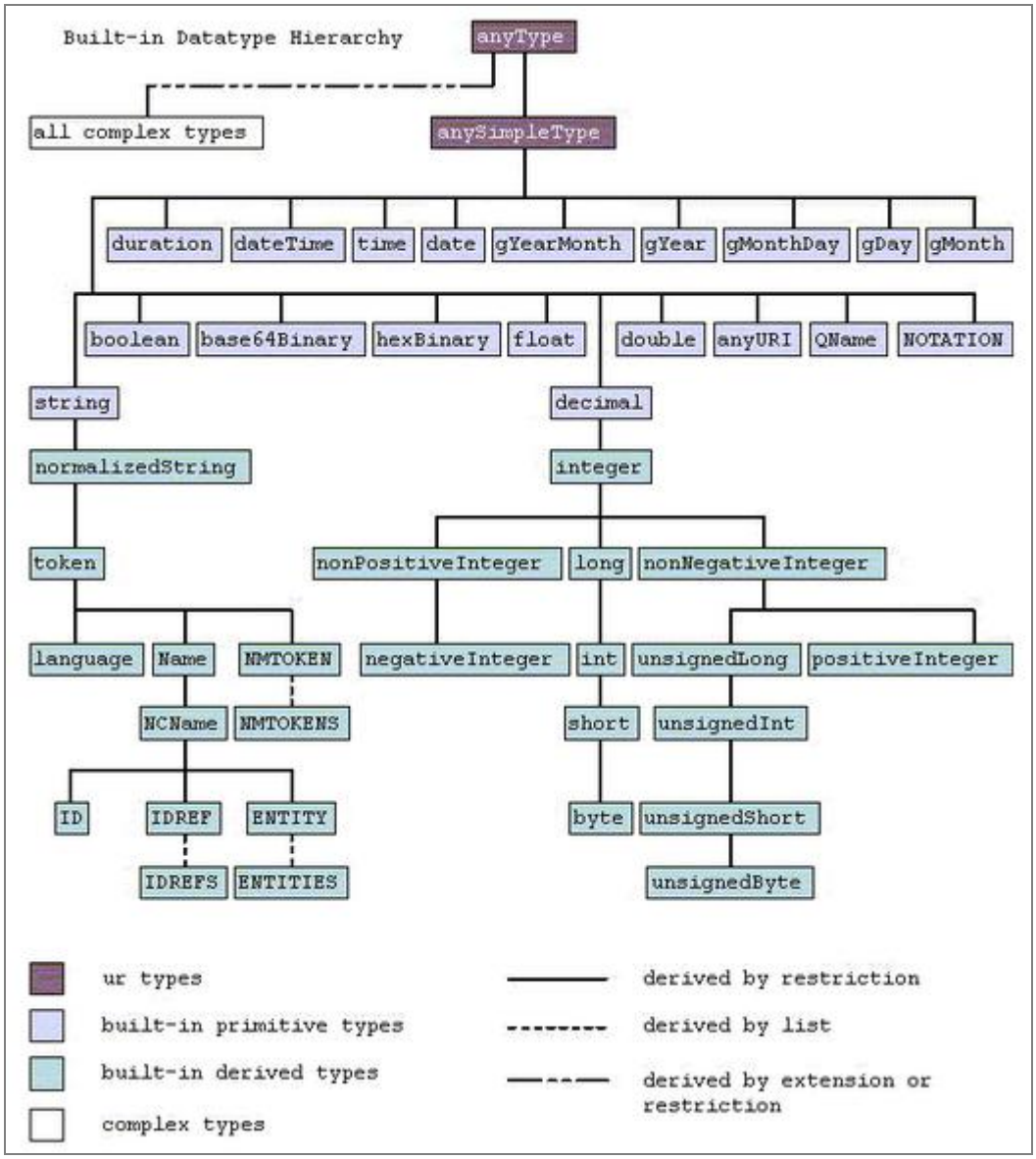


Abbildung 12: vordefinierte Datentypen des XML Schemas [33]

Wie anhand der Abbildung zu erkennen ist, stehen im Bereich der Ganzzahlen (*engl*. integer) mehrere Datentypen zur Verfügung. Da die oben genannten Bedingungen erfüllt sein müssen, kommt der Datentyp *unsignedInt* in Frage. Dieser Typ beschreibt Ganzzahlen in dem benötigten Wertebereich. Der übergeordnete Datentyp *unsignedLong* definiert hingegen einen zu großen Wertebereich, da er 64-Bit Ganzzahlen speichert. Der untergeordnete Datentyp *unsignedShort* beschreibt wiederum einen

[33] Vgl. W3C Recommendation (2004)

geringeren Wertebereich, da er 16-Bit Ganzzahlen speichert.[34] Aus diesen genannten Gründen wird der Datentyp *unsignedInt* im Implementierungsschema angewendet.

Ein weiterer Datentyp, der nicht direkt in das XML Schema übertragen werden kann, ist der *generische Typ*, den das Attribut *Attributwert* der Klasse *Attributwert* aufweist. Ziel dieses Datentyps ist das Anpassen des Typs an den gewählten Datentyp in der Klasse *Attributtyp*. Dabei soll der generische Datentyp sich entsprechend des Datentyps, des Attributtyps ändern. Eine solche Änderung ist innerhalb des XSchemas nicht möglich. Eine Option unterschiedliche Datentypen zu ermöglichen ist das Nutzen des Typs *anyType.* Wie anhand der obigen Abbildung (Abbildung 12) deutlich wird, umfasst dieser Datentyp alle komplexen und simplen Datentypen. In Anlehnung an die Datenbank ist es auch möglich diesen generischen Typ als mehrspracheigen Text oder Wertelisteneintrag darzustellen.

Weiterhin weisen viele Attribute den Datentyp *WL_Werteliste* auf. Dabei sind die Datentypen unterschiedlich bezeichnet (z.B *WL_Thema*, *WL_Einheit*), weisen jedoch alle auf den definierten Datentyp *WL_Werteliste*. Diese Bezeichnung der Typen definiert zusätzlich den Wertelistentyp, dem die einzelnen Werte zugeordnet sind. Eine Implementierung dieser unterschiedlich benannten Datentypen würde im XSchema bewirken, dass die Elemente nicht auf den Datentyp *WL_Werteliste* verweisen, sondern z.B. auf die Datentypen *WL_Thema*, *WL_Einheit*, die jedoch nicht implementiert sind. Aus diesem Grund ist es nötig die Datentypen auf *WL_Werteliste* umzubenennen. Nachfolgende Abbildung (Abbildung 13) zeigt diese Änderungen am Beispiel der Klasse *Attributtyp*.

Abbildung 13: Datentyp *WL_Werteliste*

[34] Vgl. Jakobs (2011) s.7

3.3 Implementierung mit Enterprise Architect und ShapeChange

Das zuvor beschriebene Implementierungsschema muss anschließend in ein XML Schema überführt werden. Dazu stehen zum einen das Programm *Enterprise Architect* und um anderen das Tool *ShapeChange* zur Verfügung, wobei eine Kombination aus beiden Werkzeugen zur Generierung eines XSchemas gemäß der Norm ISO 19118 erreicht wird. Das Werkzeug *Enterprise Architect* wird in nachfolgender Definition kurz dargestellt.

> *„Enterprise Architect ist ein umfangreiches, stabiles und performantes UML Analyse und Design-Werkzeug. EA unterstützt das Modellieren aller in der UML 2.4.1 spezifizierten Modelle. Darüber hinaus bietet EA weitere Features um den Softwareentwicklungsprozess zu unterstützen, wie das Sammeln von Requirements und das Erstellen von test und maintenance Modellen.“*[35]

ShapeChange hingegen ist ein Java Tool, welches UML Anwendungsschemata, die der ISO 19109 Norm entsprechen, in verschiedene Zielformate implementiert (Abbildung 14). Das am häufigsten genutzte Zielformat ist dabei das XML Schema Dokument. Weiterhin unterstützt es auch Zielformate wie Feature Catalogues, KML oder Codelisten Bibliotheken. Die Implementierung erfolgt über Implementierungsregeln.[36]

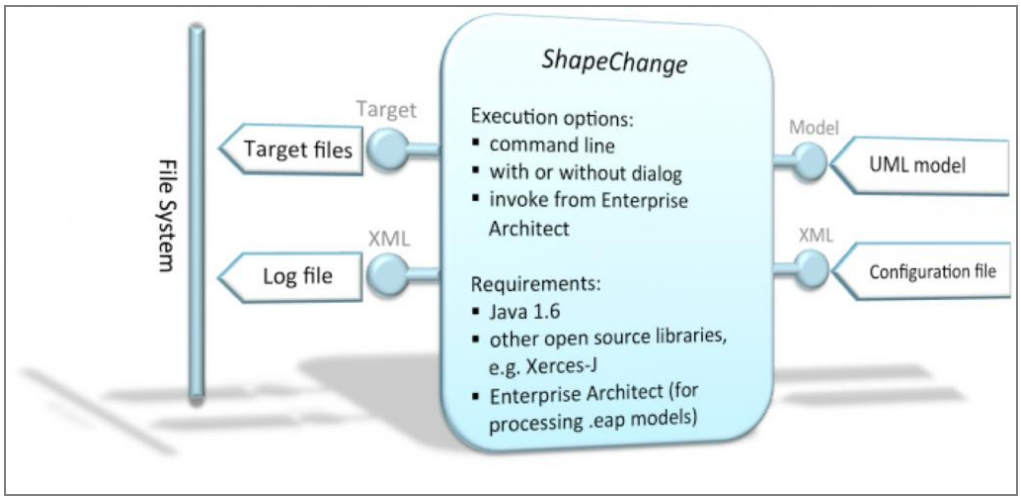


Abbildung 14: ShapeChange[37]

[35] Vgl. SparxSystems GmbH (2013)
[36] Vgl. interactive instruments GmbH u. The MITRE Corporation (2013)
[37] Vgl. interactive instruments GmbH u. The MITRE Corporation (2013)

Um mit Hilfe dieser beiden Werkzeuge ein XSchema entsprechend des gegebenen UML Anwendungsschemas generieren zu können, werden folgende Schritte durchgeführt:

1. Modellieren des Implementierungsschemas in *Enterprise Architect*
2. Export im XMI Format
3. Anpassen der *ShapeChange* Konfigurationsdatei
4. Generieren des XSchemas mittels *ShapeChange*
5. Analysieren des Ergebnisses

Jeder dieser Punkte wird nachfolgend erläutert.

1. Modellieren des Implementierungsschemas in Enterprise Architect

Das abgeleitete Implementierungsschema muss zunächst in *Enterprise Architekt* modelliert werden, da die Nutzung von *ShapeChange* darauf aufbaut. Die Klassen und Datentypen werden dabei im selben Modell erstellt. Dabei ist zu beachten, dass den Attributen, die einen benutzerdefinierten Datentypen aufweisen, dieser Datentyp direkt über die Datentypliste des Optionsmenü hinzugefügt werden. Wird hingegen nur der Name des Datentyps angegeben und keine direkte Verknüpfung über die Liste hergestellt, wird der Datentyp währen des Generierens des XSchemas nicht erkannt. Anschließend können den Typen und Klassen die Beziehungen zugeordnet werden. Die Navigierbarkeit der einzelnen Beziehungen sollte dabei angegeben werden, ansonsten werden bidirektionale Beziehungen nicht korrekt umgesetzt.

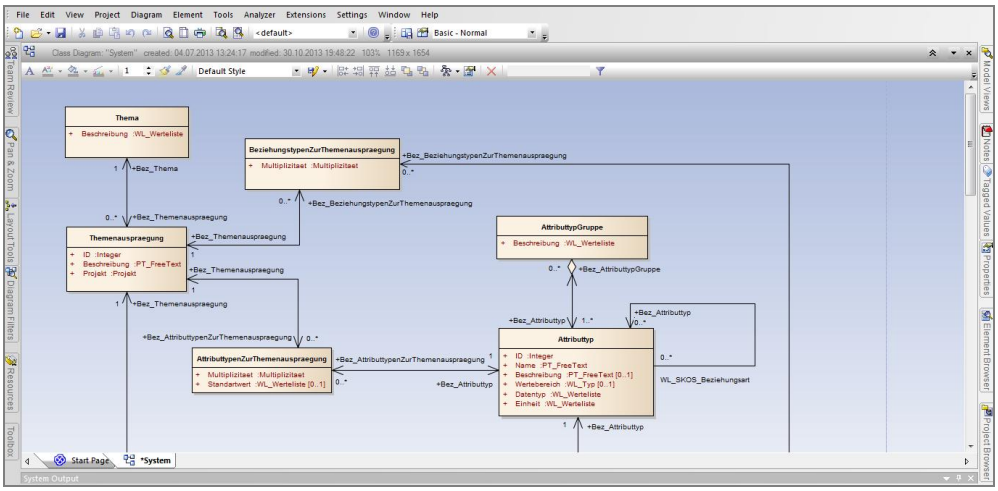


Abbildung 15: Auszug der Modellierung in *Enterprise Architect*

Nachdem das Implementierungsschema komplett modelliert wurde (Abbildung 15) ist es weiterhin nötig unter den Einstellungen des Modells den Namensraum zu definieren. Andernfalls kommt es während der Generierung des XSchemas zu Fehlern und die XSchema Datei wird nicht generiert.

2. XMI Export

Das Format XML Metadata Interchange (XMI) ist ein von der OMG (Object Management Group) verabschiedeter Standard und ermöglicht den Austausch von Modellen, die auf UML basieren.[38] Damit aus dem XMI Dokument anschließend das XSchema erzeugt werden kann, müssen folgende Vorrausetzungen erfüllt sein[39]:

- Das XMI Dokument muss der Version XMI 1.0 entsprechen
- Das XMI Dokument muss wohlgeformt sein.
- Das XMI Dokument muss gültig sein.

Das Tool *Enterprise Architect* bietet die Möglichkeit das modellierte UML Anwendungsschema als XMI Dokument zu exportieren. Dabei kann die zu verwendende XMI Version angegeben, sowie ein entsprechendes DTD Dokument erzeugt werden. Wähend des Exports wird allen Beziehungen, die keine Kardinalität aufweisen, die Multiplizität 0..* zugewiesen. Weiterhin werden Beziehungen, bei denen keine Navigierbarkeit angegeben ist, als Beziehungen mit der Richtung Quelle - Ziel interpretiert. Die Informationen der Richtung Ziel - Quelle gehen verloren. Aus diesen Gründen ist es wichtig das Modell hinsichtlich Multiplizität und Navigierbarkeit vollständig darzustellen.[40]

3. Anpassen der Konfigurationsdatei

Die Durchführung der XSchema Generierung setzt das Vorhandensein einer Konfigurationsdatei voraus. In dem Internetauftritt des Werkzeugs *ShapeChange* ist eine Beispieldatei[41] vorzufinden, die entsprechend angepasst werden kann. Diese Konfigurationsdatei beinhaltet Angaben zum Ausgansformat, sowie den entsprechenden Speicherort. Weiterhin wird der Speicherort der Log-Datei festgelegt. Desweiteren müssen Informationen (z.B. Speicherort) zum Zielformat angegeben werden. Zusätzlich ist die Angabe der Implementierungsregeln nötig. Mögliche Implementierungsregeln, die Unterstützt werden, sind:

---

[38] Vgl. Jeckle (2004) s.13
[39] Vgl. http://shapechange.net/app-schemas/xmi/
[40] Vgl. http://shapechange.net/app-schemas/xmi/
[41] http://shapechange.net/resources/test/testXMI.xml

- GML 3.2 (ISO 19136:2007)
- GML 3.2 (ISO 19136:2007) mit GML 3.3 Erweiterung
- ISO/TS 19139

Die Informationen innerhalb der Konfigurationsdatei sind entsprechend anzupassen. Für die Umsetzung des gegebenen UML Anwendungsschemas werden die Implementierungsregeln nach ISO/TS 19139 gewählt, da keine GML nach ISO 19136 generiert werden soll.

4. Generieren des XSchemas mittels ShapeChange

Nach dem XMI Export des UML Anwendungsschemas und der Anpassung der Konfigurationsdatei kann das XML Schema generiert werden. Das Ausführen des Werkzeugs *ShapeChange* wird über einen Befehl in der MS-DOS-Eingabeaufforderung bewirkt. Dieser Befehl setzt sich wie folgt zusammen:

```
java -jar ShapeChange-2.0.0-SNAPSHOT.jar -Dfile.encoding=UTF-8 -c
MA\configXMI.xml
```

Mit absetzten dieses Befehls wird die Java Datei aus dem *ShapeChange* Verzeichnis aufgerufen und die angegebene Kodierung der Zieldatei sowie die entsprechende Konfigurationsdatei übergeben. Nachdem das XSchema erzeugt wurde, kann dem Log-Dokument entnommen werden, ob Fehler aufgetreten sind. Dabei ist bereits zu erkennen, dass der Datentyp unsignedInt nicht erkannt und so nicht generiert wird. Weitere Informationen sind dabei dem XSchema Dokument (Anlage A.3) zu entnehmen.

5. Ergebnisanalyse

Als Ergebnis wird eine wohlgeformte und gültige XSchema Datei generiert, was den formalen Anforderungen entspricht. Anschließend ist zu prüfen ob die XSchema Datei auch inhaltlich die Voraussetzung erfüllt und das gegebene UML Anwendungsschema richtig im XSchema umgesetzt wurde. Das vollständige XSchema ist unter Anlage A.3 aufgeführt. Zunächst ist zu erkennen, dass alle Elemente, die eine Klasse wiederspiegeln, als globale Elemente des XSchemas generiert werden. So kann jedes dieser Elemente das Wurzelelement des Instanzdokumentes sein. Jedoch ist es nicht möglich, alle Elemente innerhalb der Datei zu führen bzw. eines dieser Elemente mehrfach anzugeben, da nur ein Wurzelelement existieren darf. Demnach ist in diesem XSchema kein Wurzelelement deklariert, welches alle diese Elemente als lokale Elemente bzw. Unteremente führt. Jedes Element weist einen entsprechenden komplexen Datentyp auf, innerhalb dem weitere Elemente bzw. Attribute der Klasse deklariert werden. Die Benennung des Datentyps erfolgt nach folgendem Muster:

```
Name Datentyp = Klassenname+'_Type'
```

Jedes Element weist weiterhin einen komplexen Datentyp auf, der Elemente für mögliche Referenzen deklariert. So werden zum einen Attribute hinzugefügt, die aus dem OGC Namensraum implementiert werden und so Referenzen über UUIDs oder URIs, jedoch nicht über IDs[42], ermöglichen. Zum anderen kann auch das Zielelement auf welches referenziert wird, direkt angegeben werden. Diese Möglichkeit führt im Instanzdokument jedoch zu tiefen Verschachtelungen und redundantem Auftreten von Elementen und ist daher zu vermeiden. Eigens definierte, komplexe Datentypen werden auf die gleiche Art und Weise umgesetzt wie die Klassen. Der Datentyp *PT_FreeText* wird aus dem Namensraum GMD importiert. Alle einfachen Datentypen werden aus dem OGC Namensraum importiert (z.B. *Integer*). Jedoch wird der Datentyp *unsignedInt* nicht erkannt, da dieser nicht im OCG Namensraum implementiert ist. Dieser Datentyp ist, wie in Abbildung 12 (Kapitel 3.2.3) erkennbar, ein vordefinierter Datentyp des XSchemas, ebenso wie die anderen einfachen Datentypen wie z.B. *Integer*. Da diese Datentypen jedoch beim Generieren des XSchemas über *ShapeChange* aus dem OCG Namensraum geladen werden, können voreingestellte Datentypen des XSchemas nicht umgesetzt werden. Eine Einstellungsmöglichkeit hinsichtlich dieses Problems ist in der Konfigurationsdatei nicht enthalten. Im Umgang mit den Wertelisten ist es wichtig, dass die Wertelisteneinträge gespeichert werden. Eine Speicherung im XSchema Dokument selbst ist nicht möglich da es sich bei den Wertelisten nicht um eine Aufzählung (*engl*. enumeration) handelt. Deshalb ist das Erzeugen einer Werteliste in einem externen Listendokument nötig. Ein solches Listendokument ist in dem, durch *ShapeChange* generierten, XSchema nicht vorgesehen.

Alternativ kann der oben aufgeführt zweite Schritt wegelassen und das XSchema direkt aus dem Enterprise Architect Modell erzeugt werden. Das Ergebnis welches erreicht wird ist dabei jedoch von der anderen Option nicht zu unterscheiden.[43]

Aus oben aufgeführten Gründen und aufgrund der hohen Abstraktheit des UML Anwendungsschemas ist es nicht möglich das XML Schema mit Hilfe der beiden Werkzeuge *Enterprise Architect* und *ShapeChange* vollständig zu erzeugen. Jedoch bietet das Anwenden dieser Tools eine gute Grundlage, auf der das XSchema weiterhin bearbeitet und angepasst werden kann. Dazu sind nachfolgend Regeln angegeben, mit denen die Implementierung des UML Anwendungsschemas im XSchema ermöglicht wird.

---

[42] Vgl. http://www.schemacentral.com/sc/niem21/ag-gco_ObjectReference.html

[43] Vgl. http://shapechange.net/get-started/

3.4 Implementierungsregeln

Wie im vorherigen Kapitel beschrieben, wird bei der Implementierung mittels *Enterprise Architect* und *ShapeChange* ein Ergebnis erzielt, das die Aussagen des UML Anwendungsschema nicht ausreichend darstellt. Aus diesem Grund werden Implementierungsregen erstellt, die es ermöglichen das XML Schema aus dem UML Anwendungsschema abzuleiten. Als Grundlage dient dazu die Norm ISO 19118. Die nachfolgend erläuterten Regeln sind in zusammengefasster Form in Anlage A.4 vorzufinden. Das komplette XSchema ist unter Anlage A.5 aufgeführt.

3.4.1 Wurzelelement

Wurzelelemente eines Instanzdokumentes können alle Elemente sein, die global im XSchema deklariert sind, also als direktes Unterelement des `xs:schema` Elementes auftreten. Als Wurzelelement eines Instanzdokumentes wird ein Element mit dem Namen `Datensatz` deklariert. Innerhalb dieses Elementes werden alle Unterelemente lokal deklariert, die anschließend im Wurzelelement des Instanzdokumentes enthalten sein können bzw. müssen. Diese Zuordnung wird im XSchema Dokument über einen unbenannten komplexen Datentyp durchgeführt. Aufgrund des einmaligen Auftretens dieses Typs ist es nicht nötig einen benannten Datentyp zu wählen. Die Unterelemente spiegeln dabei alle Klassen des UML Anwendungsschemas wieder. Genauere Erläuterungen zu dieser Darstellung sind im nachfolgenden Kapitel aufgeführt. Innerhalb dieser Datentypdefinition wird ebenfalls sichergestellt, dass alle diese Unterelemente mindestens einmal im Instanzdokument aufgeführt werden müssen. Die Reihenfolge ist dabei alphabetisch. Neben dem `Datensatz` Element wird ein weiteres Wurzelelement (`Wertelisten`) deklariert. Dieses wird zur Beschreibung der Wertelisten verwendet. Innerhalb dieses Elementes können zwei Unterelemente auftreten. Zum einen die Werteliste, welche alle Wertelistentypen beinhaltet und zum anderen die Werteliste, welche alle Wertelisteneinträge aufweist. Jede Werteliste kann dabei nur einmal vertreten sein.

Als Zielnamensraum wird die Abkürzung OpenInfRA verwendet. Diese Angabe kann jedoch entsprechend angepasst werden.

3.4.2 Klassen

UML Klassen können auf unterschiedliche Arten im XML Schema implementiert werden. Diese Implementierung richtet sich dabei nach dem Stereotyp der einzelnen Klasse. Die möglichen Stereotypen sind dabei in nachfolgender Tabelle aufgelistet (Tabelle 2).

Tabelle 2: Stereotypen von UML Klassen[44]

Stereotyp	**Bemerkung**
<<BasicType>>	Wird im XML Schema als simpler Datentyp definiert.
<<DataType>>	Beschreibt einen strukturierten Datentyp. Wird im XML Schema als komplexer Datentyp definiert.
<<Union>>	Beschreibt eine Liste von Attributen, wobei nur ein Attribut ausgewählt werden kann. Wird im XML Schema als komplexer Datentyp definiert mit der Auswahlgruppe `choice`.
<<Enumeration>>	Beschreibt eine Aufzählung. Wird im XML Schema als simpler Datentyp definiert.
<<CodeList>>	Beschreibt eine erweiterte Aufzählung. Wird als externe Codeliste bzw. Wörterbuch im XML Format definiert.
<<Interface>>	Wird nicht im XML Schema umgesetzt.
<<Type>>	Beschreibt einen Objekttyp. Wird im XML Schema als komplexer Datentyp definiert.
kein Stereotyp	Beschreibt einen Objekttyp. Wird im XML Schema als komplexer Datentyp definiert.

Die Klassen des vorliegenden UML Anwendungsschemas weisen zum einen den Stereotyp <<DataType>> und zum anderen keinen Stereotyp auf. Die Umsetzung der Datentypen wird im Kapitel 3.4.4 beschrieben. Klassen ohne Stereotyp definieren Objekttypen. Um diese Typen vollständig im XML Schema abzubilden, wird zunächst ein Element deklariert, welches den Klassennamen aufweist.

```
<xs:element name="Themenauspraegung" type="ThemenauspraegungType"/>
```

Dieses Element verwendet dabei einen benannten Datentyp, der als komplexer Typ definiert ist.

[44] Vgl. Norm ISO 19118 (2006) s.24

```
<xs:complexType name="ThemenauspraegungType">
   …
   <!--Deklaration von Unterelementen-->
   …
   <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>
```

Dieser Datentyp erhält im konkreten Beispiel den Namen `ThemanauspraegungType`. Der Name setzt sich aus dem Klassennamen und dem Wort „Type“ zusammen. Durch diese Regeln der Benennung sind Elementnamen und Namen von Datentypen klar voneinander getrennt und sichert die Übersichtlichkeit des XSchema Dokumentes. Weiterhin werden dem Element Attribute zur Identifikation hinzugefügt. Diese Zuordnung erfolgt, wie in obigen Beispiel ersichtlich, im komplexen Datentyp. Dabei wird das Attribut mit dem Namen `id` deklariert, welches den Datentyp `ID` aufweist. Zusätzlich erhält dieses Attribut den Hinweis, wie es genutzt werden muss. In diesem Fall ist es notwendig dieses Attribut im Instanzdokument anzugeben. Die Nutzung wird über das Attribut *use* festgelegt. Standartmäßig besagt es, dass die Angabe des deklarierten Attributes im Instanzdokument optional ist. Der Attributwert `required` bewirkt hingegen, dass es notwendig ist das Attribut `id` anzugeben um ein gültiges XML Dokument zu erhalten. Die Angabe des `id` Attributes ist verpflichtend, weil über diese ID im Instanzdokument auf das Element referenziert wird. Andererseits könnten diese Referenzen nicht aufgebaut werden.

3.4.3 Attribute

Die Klassen des UML Anwendungsschemas weisen Attribute auf. Diese Attribute werden im XML Schema innerhalb des komplexen Datentyps des Klassenelements deklariert. Ein Attribut des UML Anwendungsschemas wird dabei als XSchema Element implementiert. Der Name dieser Elemente ist identisch zu den Namen der Attribute des UML Anwendungsschemas. Wie im nachfolgenden Beispiel zu sehen, werden die Attributelemente innerhalb der Auswahlgruppe `sequence` deklariert.

```
<xs:complexType name="ThemenausprägungType">
   <xs:sequence>
      <xs:element name="ID" type="xs:integer"/>
      <xs:element name="Beschreibung"
       type="gmd:PT_FreeText_PropertyType"/>
      <xs:element name="Projekt" type="ProjektType"/>
      …
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>
```

Durch die Sequenz wird festgelegt, dass die Elemente die gleiche Reihenfolge aufweisen wie die Attribute im UML Anwendungsschema. Weiterhin ermöglicht sie, dass die Elemente entsprechend der angegebenen Multiplizität auftreten. Die Kardinalität der einzelnen UML Attribute kann im XSchema über die Attribute `maxOccurs` und `minOccurs` umgesetzt werden. Die unterschiedlichen Möglichkeiten sind in nachfolgender Tabelle dargestellt (Tabelle 3).

Tabelle 3: Multiplizität[45]

UML	minOccurs	maxOccurs	Nötige Elementdeklarationen
1 (default)	1 (default)	1 (default)	
0..1	0	1 (default)	minOccurs="0"
0..*	0	unbounded	minOccurs="0" maxOccurs="unbounded"
1..*	1 (default)	unbounded	maxOccurs="unbounded"
2..8	2	8	minOccurs="2" maxOccurs="8"

3.4.4 Beziehungen

Beziehungen des UML Anwendungsschemas werden im XML Schema wie Attribute der entsprechnden Klassen gehandhabt. Dabei werden die Beziehungen ebenfalls als Elemente innerhalb des komplexen Datentyps des Klassenelementes deklariert. Die Namen der Elemente richten sich nach den Rollennamen, die die Beziehung zwischen Ausgangsklasse und Zielklasse näher beschreiben. Die Multiplizität der Beziehung wird analog zu der Kardinalität der UML Attribute umgesetzt und richtet sich ebenfalls nach den in Tabelle dargestellten Deklarationen. Die Datentypdefinition des einzelnen Elementes steht dabei im Zusammenhang mit der Art der Beziehung. Eine Beziehung kann

- eine Assoziation,
- eine Aggregation,
- eine Komposition oder
- Generalisierung

sein.

[45] Vgl. Norm ISO 19118 (2006) s.45

Assoziation

Stellt die Beziehung eine Assoziation dar (Abbildung 16), verweist das Element auf einen benannten komplexen Datentyp.

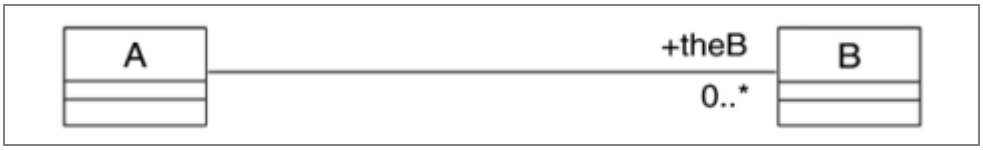


Abbildung 16: Beispiel Assoziation[46]

Dieser komplexe Datentyp soll entweder auf einer Objektreferenz basieren oder Attribute, die eine Objektreferenz definieren, beinhalten. Das nachfolgende Beispiel zeigt den implementierten XML Code der oben dargestellten Assoziation:[47]

```
<xs:complexType name="A">
  <xs:sequence>
    <xs:element name="theB" type="ref_B" minOccurs="0"
     maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ref_B">
  <xs:attributeGroup ref="IM_ObjectReference"/>
</xs:complexType>
```

Aus diesem Beispiel wird ersichtlich, dass die Ausgangsklasse (Klasse A) die Beziehung zur Zielklasse (Klasse B) als Element deklariert. Der zugehörige Datentyp `ref_B` referenziert auf eine Attributgruppe `IM_ObjectReference`. Diese Attributgruppe definiert eine Liste von drei Attributen, die zum Referenzieren anderer Objekte genutzt wird.

```
<xs:attributeGroup name="IM_ObjectReference">
    <xs:attribute name="idref" type="xs:IDREF"/>
    <xs:attribute name="uuidref" type="xs:string"/>
    <xs:attribute name="uriref" type="xs:anyURI"/>
</xs:attributeGroup>
```

Das `idref` Attribut ermöglicht es auf ein Element zu referenzieren, welches innerhalb des XML Dokumentes befindlich ist und durch eine ID identifiziert wird. Dokumentüber-

[46] Vgl. Norm ISO 19118 (2006) s.46
[47] Vgl. Norm ISO 19118 (2006) s.46

greifende Referenzen können mit diesem Attribut nicht durchgeführt werden. Zur Umsetzung dieses Falls kann das Attribut `uriref` genutzt werden, da es ermöglicht auf Objekte anderer Datensätze zu referenzieren. Das dritte Attribut `uuidref` stellt Verweise auf Objekte, die über eine UUID identifiziert werden, bereit. Die obige Abbildung (Abbildung 16) definiert eine Assoziation, die nur in eine Richtung navigierbar ist. Das heißt, dass die Klasse B über keinerlei Informationen über die Klasse A verfügt. Jedoch kann diese Implementierung auch auf bidirektionale Beziehungen angewendet werden. Dabei wird der Zielklasse ein Element, welches die Beziehung zur Klasse A darstellt, zugeordnet. Der Elementname ist wiederum der Rollenname. In obiger Abbildung (Abbildung 16) wäre das beispielsweise der Name *theA.* Die zugehörige Implementierung ist folgende:

```
<xs:complexType name="B">
  <xs:sequence>
    <xs:element name="theA" type="ref_A" minOccurs="0"
     maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ref_A">
  <xs:attributeGroup ref="IM_ObjectReference"/>
</xs:complexType>
```

Angewendet auf das gegebene UML Anwendungsschema ist die Implementierung der Assoziationen am Beispiel der Klasse Themenausprägung dargestellt:

```
<xs:complexType name="ThemenausprägungType">
      <xs:sequence>
         <xs:element name="ID" type="xs:integer"/>
         <xs:element name="Beschreibung"
            type="gmd:PT_FreeText_PropertyType"/>
         <xs:element name="Projekt" type="ProjektType"/>
         <xs:element name="Bez_Themeninstanz"
            type="ReferenzType" minOccurs="0"
            maxOccurs="unbounded"/>
         <xs:element name="Bez_AttributtypenZurThemenausprägung"
            type="ReferenzType"
            minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="Bez_Thema" type="ReferenzType"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="required"/>
   </xs:complexType>
```

```
<xs:complexType name="ReferenzType">
   <xs:attribute name="idref" type="xs:IDREF" use="required"/>
</xs:complexType>
```

Zur Realisierung der Referenz zu anderen Objekten wird das Attribut `idref` mit dem Datentyp `IDREF` deklariert, da sich die Elemente, auf die referenziert wird, innerhalb desselben Instanzdokumentes befinden. Alle Assoziationen verweisen dabei auf denselben Datentyp `ReferenzType`. Alternativ zu den zuvor genannten Referenzen könnten die Elemente der Zielklasse auch als Unterelemente des Beziehungselements dargestellt werden. Auf diese Möglichkeit wird jedoch verzichtet, da so eine Verschachtelung entsteht, die das Instanzdokument unübersichtlich gestaltet und zusätzlich die Größe dieses Dokumentes negativ beeinflusst. Grund dafür ist, dass Elemente, die die Zielklasse darstellen, bei jeder Beziehung erneut eingefügt und so redundant geführt werden.

Aggregation

Aggregationen werden auf ähnliche Art und Weise wie Assoziationen implementiert. Anhand der nachfolgenden Abbildung (Abbildung 17) wird die Umsetzung im XSchema erläutert.

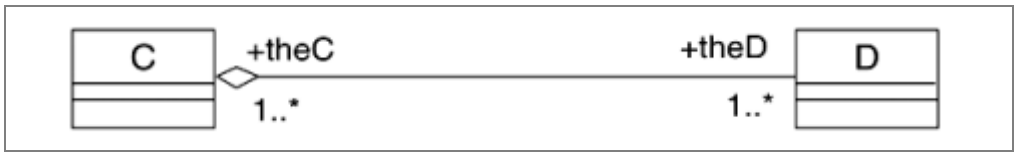


Abbildung 17: Beispiel Aggregation[48]

Der definierte Datentyp unterscheidet sich dabei ob es sich bei der Klasse um das Aggregat (Klasse C) oder einen Teil dieses Aggregats (Klasse D) handelt. Die Beziehung wird dabei in der Teilklasse nach den gleichen Regeln implementiert, wie die zuvor beschriebe Assoziation:[49]

```
<xs:complexType name="D">
  <xs:sequence>
    <xs:element name="theC" type="IM_ObjectReference" minOccurs="1"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

[48] Vgl. Norm ISO 19118 (2006) s.47
[49] Vgl. Norm ISO 19118 (2006) s.47

In der Aggregatklasse wird die Beziehung nach folgenden Punkten implementiert:[50]

```
<xs:complexType name="C">
  <xs:sequence>
    <xs:element name="theD" minOccurs="1" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="D" type="D" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attributeGroup ref="IM_ObjectReference"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Das Element, welches die Beziehung beschreibt, weist dabei einen unbenannten komplexen Datentyp auf, innerhalb dem wiederum ein Element deklariert ist, das auf den Datentyp der Teilklasse referenziert. Weiterhin wird der definierten Beziehung ein Referenzattribut zugeordnet. Das bewirkt, dass das Aggregatelement über Referenzen auf das entsprechende Teilelement verweist. Aus anwenden des unbenannten komplexen Datentyp wird beim Umsetzen der Aggregation des UML Anwendungsschema verzichtet, da die verwendung von Referenzattributen ausreichend ist.

```
<xs:complexType name="AttributtypGruppeType">
   <xs:sequence>
      <xs:element name="Beschreibung" type="WL_ReferenzType"/>
      <xs:element name="Bez_Attributtyp" type="ReferenzType"
       maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>
```

Komposition

Weiterhin kann eine Beziehung auch eine Komposition (Abbildung 18) sein, die eine Kompositionsklasse (Klasse E) und ein Teilklasse dieser Komposition (Klasse F) darstellt.

[50] Vgl. Norm ISO 19118 (2006) s.47

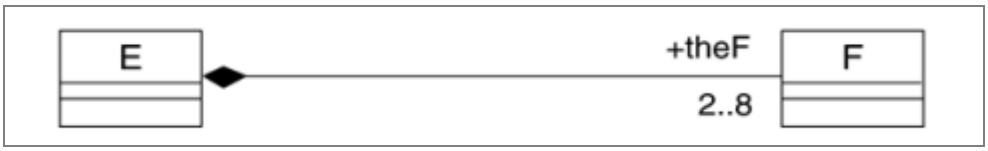


Abbildung 18: Beispiel Komposition[51]

Diese Beziehung wird im Vergleich zu den zuvor beschriebenen Beziehungen nur in der Kompositionsklasse implementiert, unabhängig davon, ob es sich um eine bidirektionale Beziehung handelt oder nicht. Innerhalb des komplexen Typs des Kompositionselements wird die Beziehung als Element deklariert, welches den Datentyp des Teilelements aufweist:[52]

```
<xs:complexType name="E">
  <xs:sequence>
    <xs:element name="theE" type="F" minOccurs="2" maxOccurs="8"/>
  </xs:sequence>
</xs:complexType>
```

Das UML Anwendungsschema weist keine Komposition auf. Die Implementierungsregel für diese Beziehung wird dennoch aufgeführt, da das UML Anwendungsschema keine finale Version aufweist und so einer weiteren Überarbeitung unterliegt. Ergibt sich im Verlauf der Weiterverarbeitung die Notwendigkeit einer Komposition, so kann diese nach oben genannter Regel implementiert werden.

Generalisierung

Schlussendlich kann eine Beziehung auch eine Generalisierung sein. Bei der Umsetzung einer Generalisierung in das XML Schema wird unterschieden, ob eine Klasse von nur einer oder von mehreren Klassen erbt. Die erbende Klasse wird als Subklasse bezeichnet und die Klasse, von der geerbt wird, stellt die Superklasse dar. Wie in nachfolgender Abbildung (Abbildung 19) ersichtlich wird, erben die Klassen *H* und *I* von der Klasse *G*.

---

[51] Vgl. Norm ISO 19118 (2006) s.47

[52] Vgl. Norm ISO 19118 (2006) s.47

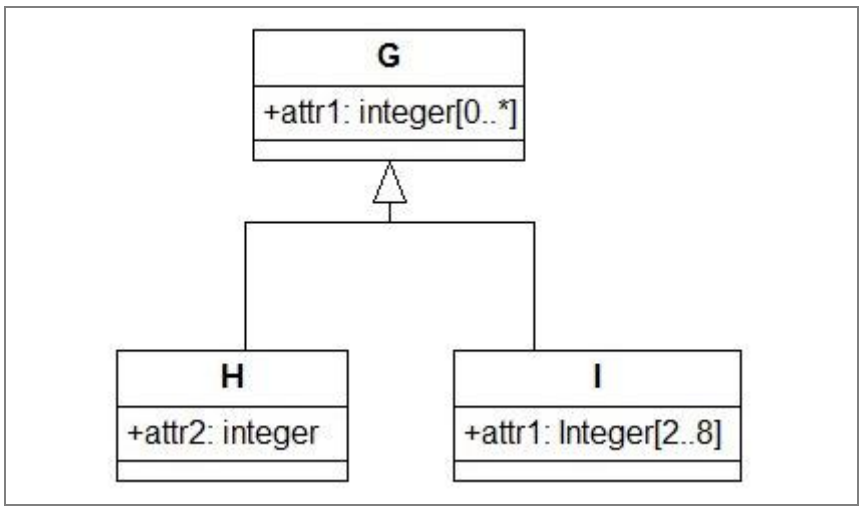


Abbildung 19: Beispiel Generalisierung[53]

Die Generalisierung wird dabei in der erbenden Klasse (Klasse *H*) über Erweiterungen (*engl.* extensions) oder Beschränkungen (*engl*. restrictions) implementiert. Die Datentypen *H* und *I* werden so auf Grundlage des schon bestehenden Datentyps *G* gebildet. Der Datentyp der Klasse *H* stellt eine Erweiterung des Datentyp *G* dar, indem ihm ein neues Element hinzugefügt wird. Das neu hinzugefügte Element implementiert das Attribut der Klasse *H*.[54]

```
<xs:complexType name="H">
  <xs:complexContent>
    <xs:extension base="G">
      <xs:sequence>
        <xs:element name="attr2" type="integer"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Die Klasse *I* hingegen begrenzt die Häufigkeit des Attributes *attr1*. Somit muss diese Generalisierung über eine Einschränkung des Datentyps *G* implementiert werden:[55]

```
<xs:complexType name="I">
  <xs:complexContent>
    <xs:restriction base="G">
      <xs:sequence>
```

[53] Erstellt mit *StarUML*
[54] Vgl. Norm ISO 19118 (2006) s.41
[55] Vgl. Norm ISO 19118 (2006) s.41

```
        <xs:element name="attr1" type="Integer" minOccurs="2"
            maxOccurs="8"/>
      </xs:sequence>
    </xs:restricton>
  </xs:complexContent>
</xs:complexType>
```

Eine Weitere Möglichkeit eine Generalisierung zu implementieren, ist das Übertragen der Attribute der Superklasse in die Subklasse. Diese Möglichkeit wird häufig bei mehrfacher Vererbung (Abbildung 20) umgesetzt, da das XSchema diese Form der Vererbung nicht unterstützt.[56]

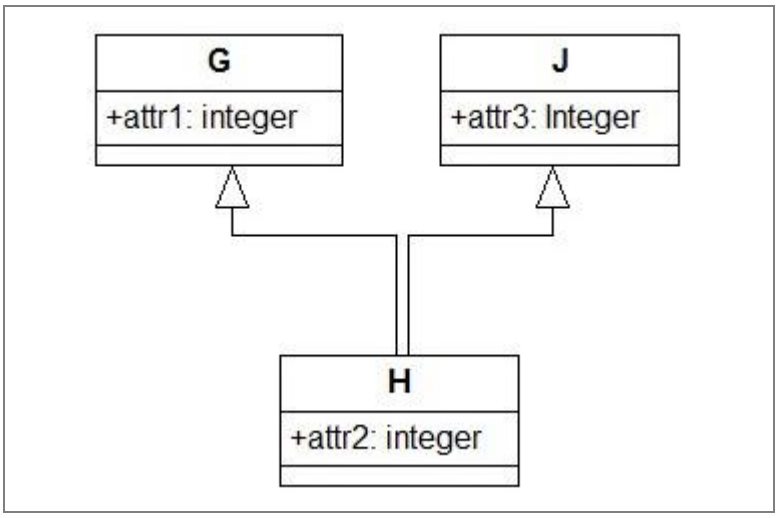


Abbildung 20: Beispiel mehrfache Vererbung[57]

Die mögliche Implementierung ist nachfolgend aufgeführt:[58]

```
<xs:complexType name="H">
  <xs:complexContent>
      <xs:sequence>
        <xs:element name="attr1" type="Integer"/>
        <xs:element name="attr2" type="Integer"/>
        <xs:element name="attr3" type="Integer"/>
      </xs:sequence>
  </xs:complexContent>
</xs:complexType>
```

[56] Vgl. Norm ISO 19118 (2006). *Geographic information – Encoding*.

[57] Erstellt mit *StarUML*

[58] Vgl. Norm ISO 19118 (2006) s.42

Innerhalb des gegebenen UML Anwendungsschemas erben die Klasse *Beziehungstyp* und *Attributtyp* von der abstrakten Klasse *Eigenschaft*. Diese Generalisierung wurde wie in Kapitel 3.2.2 beschrieben aufgelöst. Die daraus resultierenden Klassen und Assoziationen werden nach den zuvor beschriebenen Regeln umgesetzt.

SKOS Beziehungen

Ein weiterer wichtiger Punkt bei der Implementierung von Beziehungen sind die SKOS Beziehungsarten. Diese werden auf ähnliche Art und Weise wie die zuvor beschriebenen Assoziationen umgesetzt. Dabei wird die Beziehung zunächst als Element des komplexen Datentyps der entsprechenden Klasse implementiert.

```
<xs:element name="Bez_Attributtyp" type="SKOSReferenzType"
minOccurs="0" maxOccurs="unbounded"/>
```

Das Element verweist im Vergleich zu den Assoziationen nicht auf den Datentyp `ReferenzTyp`, sondern auf den komplexen Datentyp `SKOSReferenzType`. Dieser deklariert neben dem Attribut `idref` noch ein zusätzliches Attribut `SKOS_Beziehungsart`.

```
<xs:complexType name="SKOSReferenzType">
   <xs:attribute name="idref" type="xs:IDREF" use="required"/>
   <xs:attribute name="SKOS_Beziehungsart" type="xs:anyURI"
   use="required"/>
</xs:complexType>
```

Das `SKOS_Beziehungsart` Attribut ermöglicht dabei die Referenz auf den entsprechenden Wertelsteneintrag, der die SKOS Beziehungsart beschreibt.

Beziehung zwischen Themeninstanzen

Themeninstanzen weisen untereinander Beziehungen auf. Diese Beziehungen sind durch die Assoziationsklasse näher beschrieben. Zunächst werden diese Beziehungen entsprechend der Implementierungsregeln von Assoziationen umgesetzt, indem diese als Element des komplexen Datentyps der Klasse *Themeninstanz* deklariert wird.

```
<xs:element name="BeziehungZuAnderenThemeninstanzen" ty-
pe="BeziehungZuAnderenThemeninstanzenType" minOccurs="0"
maxOccurs="unbounded"/>
```

Dieses Element verweist, im Gegensatz zu den zuvor beschriebenen Assoziationen, auf den komplexen Datentyp `BeziehungZuAnderenThemeninstanzenType`.

```
<xs:complexType name="BeziehungZuAnderenThemeninstanzenType">
   <xs:attribute name="idref" type="xs:IDREF" use="required"/>
   <xs:attribute name="Beziehungstyp" type="xs:IDREF" use="required"/>
</xs:complexType>
```

Dieser Datentyp definiert neben dem Attribut zur Referenz auf das Zielelement der Beziehung auch ein Attribut, welches es ermöglicht, dass auf den Beziehungstyp verwiesen werden kann. Dieser Verweis wird ebenfalls über eine ID Referenz umgesetzt, da sich das Element des Beziehungstyps im selben Dokument befindet.

3.4.5 Datentypen

Die Datentypen der Attribute des UML Anwendungsschemas können teilweise mit Hilfe von vordefinierten Datentypen des XML Schemas (*engl*. build-in Datatyps), die bereits in Kapitel 3.2.3 unter Abbildung 12 aufgeführt wurden, implementiert werden. Einige Datentypen müssen aus anderen Namensräumen importiert oder innerhalb des XML Schemas definiert werden. Das Nutzen von vordefinierten Datentypen kann dabei auf einfachem Weg realisiert werden.

```
<xs:element name="ID" type="xs:integer"/>
```

Der Datentyp erhält dabei das Präfix `xs`, welches besagt, dass der Datentyp (z.B. `integer`) aus dem Namensraum XMLSchema verwendet wird. Datentypen die innerhalb des XSchemas definiert werden müssen, sind im UML Anwendungsschema über den Stereotyp <<DataType>> gekennzeichnet. Diese Typen werden zu einem komplexen Datentyp konvertiert. Die UML Attribute und Beziehungen können entweder als Attribute oder lokale Elemente im komplexen Datentyp implementiert werden. Das resultierende XML Schema weist diese UML Attribute als lokal deklarierte Elemente auf, da einige UML Attribute wiederum auf weitere komplexe Datentypen referenzieren. Diese Referenz kann mit XML Attributen nicht umgesetzt werden, da Attribute innerhalb eines XML Schemas nur simple Datentypen aufweisen können. Implementierte Datentypen benötigen im Gegensatz zu Klassen keine Attribute zur Identifizierung, da im Instanzdokument in der Regel nicht über Referenzen auf Datentypen verwiesen wird, sondern die lokalen Elemente des Datentyps direkt als Unterelemente eingefügt werden.

```
<xs:element name="Multiplizität" type="MultiplizitätType"/>

<xs:complexType name="MultiplizitätType">
   <xs:sequence>
      <xs:element name="Min-Wert" type="xs:unsignedInt"/>
```

```
      <xs:element name="Max-Wert" type="xs:unsignedInt"
       minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
```

Im Verlauf hat sich jedoch gezeigt, dass bei der Anwendung dieser Regeln auf den Datentyp *Projekt* unnötige Verschachtelungen auftreten, da jedes Projekt die Möglichkeit bietet ein Teilprojekt aufzuweisen. So wird im Instanzdokument folgendes Ergebnis erzielt:

```
<Projekt>
   <ID>0</ID>
   <Name>…</Name>
   <Beschreibung>…</Beschreibung>
   <Teilprojekt_von>
      <ID></ID>
      <Name></Name>
   </Teilprojekt_von>
</Projekt>
```

Wie obiges Beispiel zeigt, ist aufgrund der Verschachtelung eine Unübersichtlichkeit gegeben, da die Elemente `ID` und `Name` mehrfach auftreten. Weiterhin werden Projekte redundant im XML Dokument geführt, da mehrere Teilprojekte zu einem Projekt gehören können. Dieses Problem wird umgangen, indem alle Projekte wie die bereits beschriebenen Klassen implementiert werden. So werden die Projekte ebenfalls als erstes Kindelement des Wurzelelementes geführt und weisen ein `id` Attribut auf. Anschließend kann auf die entsprechenden Projekte referenziert werden.

```
<Projekt idref="ProjektID_1"/>
```

Wertelisten

Wird die zuvor beschriebene Implementierung von komplexen Datentypen auf die Wertelisten angewendet, entsteht im Instanzdokument das folgende, am Beispiel des *Einheit* Attributs beschriebene Ergebnis:

```
<Einheit>
   <ID>2002</ID>
   <Name>…</Name>
   <Beschreibung>…</Beschreibung>
   <Sichtbarkeit>1</Sichtbarkeit>
   <gehörtZu_Typ>
      <ID>3003</ID>
      <Name>…</Name>
```

```
      <Beschreibung>…</Beschreibung>
   </gehörtZu_Typ>
</Einheit>
```

Dieses Ergebnis ist unübersichtlich, da die Elemente `ID`, `Name` und `Bezeichnung` mehrfach untereinander auftreten. Weiterhin sind die Wertelisten nicht komplett im Schemadokument enthalten. Da das XML Dokument jedoch auf diesen Wertelisten aufbaut, ist das Speichern der kompletten Wertelisten im XML Dokument eine wichtige Bedingung. Aus diesem Grund werden alle Wertelisteeinträge und Wertelistentypen in einem externen XML Dokument geführt. Dabei wird das Element `Wertelisten` als Wurzelelement eingeführt. Dieses Element weist die Unterelemente `WL_Typ` und `WL_Werteliste` auf, welche die entsprechenden Wertelistentypen und Wertelisteneinträge beinhalten.

```
<xs:element name="Wertelisten">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="WL_Typ">
            …
         </xs:element>
         <xs:element name="WL_Werteliste">
            …
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Die Umsetzung erfolgt dabei über einen unbenannten komplexen Datentyp. Das Element `WL_Typ` weist ebenfalls einen solchen Datentyp auf, innerhalb dem ein Element *Typ* deklariert wird, das wiederum auf den Datentyp `WL_TypType` referenziert. Über das `Typ` Element werden anschließend alle WL_Typen im Instanzdokument beschrieben. Dabei ist festgelegt, dass mindestens ein Typ enthalten sein muss, da innerhalb des UML Anwendungsschemas definiert ist, dass ein Wertelisteneintrag immer einem Wertelistentypen zugeordnet ist.

```
<xs:element name="WL_Typ">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Typ" type="WL_TypType"
             maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
</xs:element>
```

Analog wird auch der komplexe Datentyp für das Element `WL_Werteliste` implementiert. Um anschließend im Instanzdokument auf die entsprechenden Einträge und Typen referenzieren zu können, ist es nötig den Wertelisteneinträgen und –typen ebenfalls einen Identifikator in Form eines `id` Attributes zuzuweisen. Anschließend kann innerhalb des Wertelistendokumentes, wie bereits in einem der vorherigen Kapiteln beschrieben, über das `idref` Attribut auf die nötigen Einträge und Typen referenziert werden. Um innerhalb des Instanzdokumentes auf die Einträge der Wertelisten verweisen zu können, wird mittels eines Attributes des Datentyps `anyURI` auf das konkrete Element der Werteliste referenziert.

```
<xs:complexType name="WL_ReferenzType">
   <xs:attribute name="WertelistenEintrag" type="xs:anyURI"
    use="required"/>
</xs:complexType>
```

Mit Hilfe des Datentyps *anyURI* ist es möglich dokumentübergreifende Referenzen umzusetzen. Auf die dargestellte Art und Weise kann die komplette Werteliste in einem XML Dokument geführt werden, sodass anschließend auf die einzelnen Einträge referenziert werden kann, ohne das die zuvor genannte Unübersichtlichkeit und Redundanz auftritt.

Nachfolgende Abbildung (Abbildung 21) zeigt das Diagramm, das die Abbildung Implementierung der Wertelisten im XSchema Dokument repräsentiert.

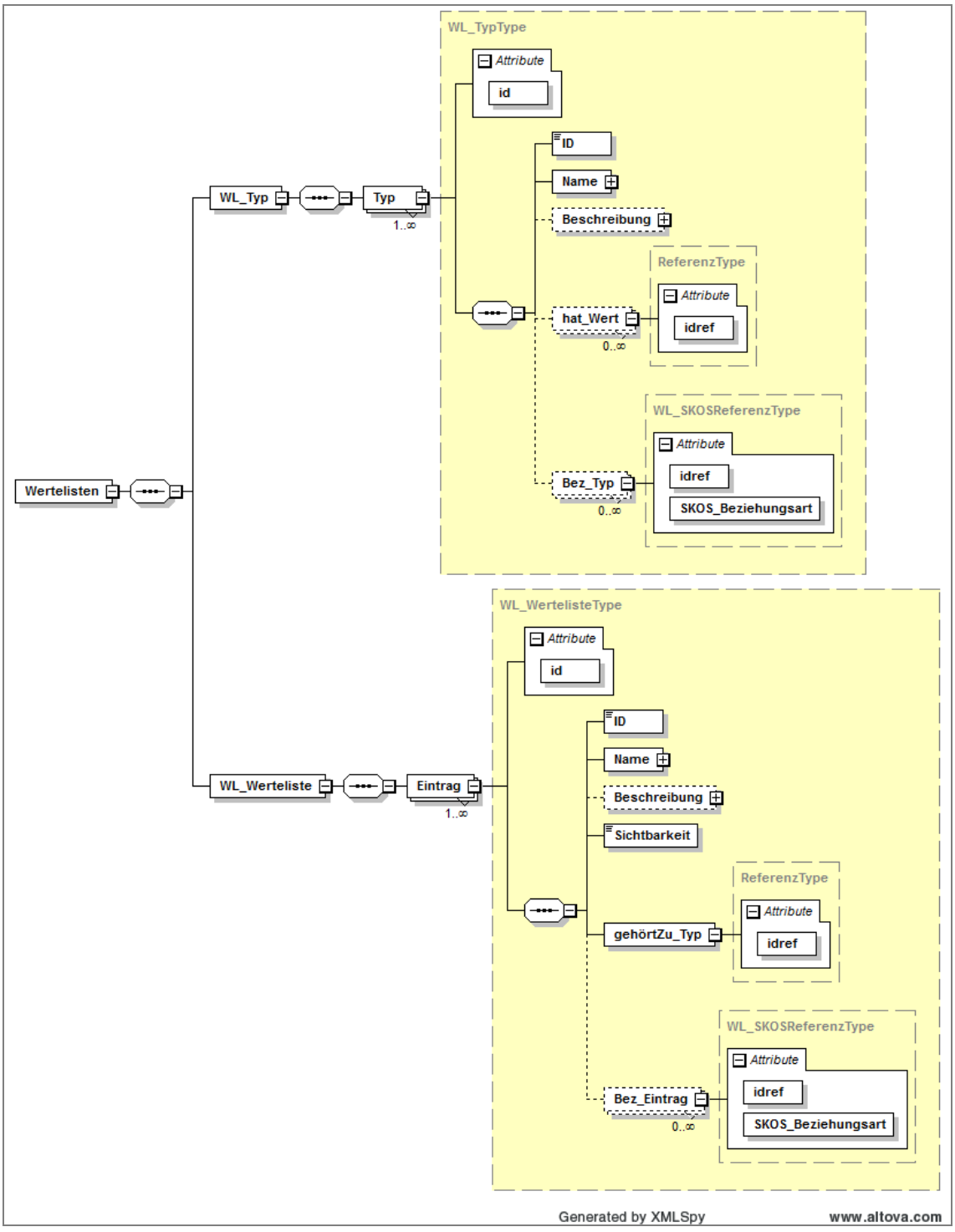


Abbildung 21: Abbildung der Wertelisten im XSchema[59]

[59] Erstellt mit *Altova XMLSpy® 2013 Professional Edition*

Mehrsprachigkeit

Die Mehrsprachigkeit bestimmter Attribute wird, wie bereits erläutert, über den Datentyp *PT_Freetext* realisiert. Sowohl in ISO 19115 also auch in ISO 19139 wird dieser Datentyp definiert, wobei die Implementierung von ISO 19139 eine Modifizierung der Beschreibung aus ISO 19115 darstellt. Zur Implementierung dieses Datentyps im XSchema wird die Definition aus ISO 19139 verwendet, da diese ein konzeptuelles Schema (Abbildung 22), sowie ein eigenes XSchema bietet.

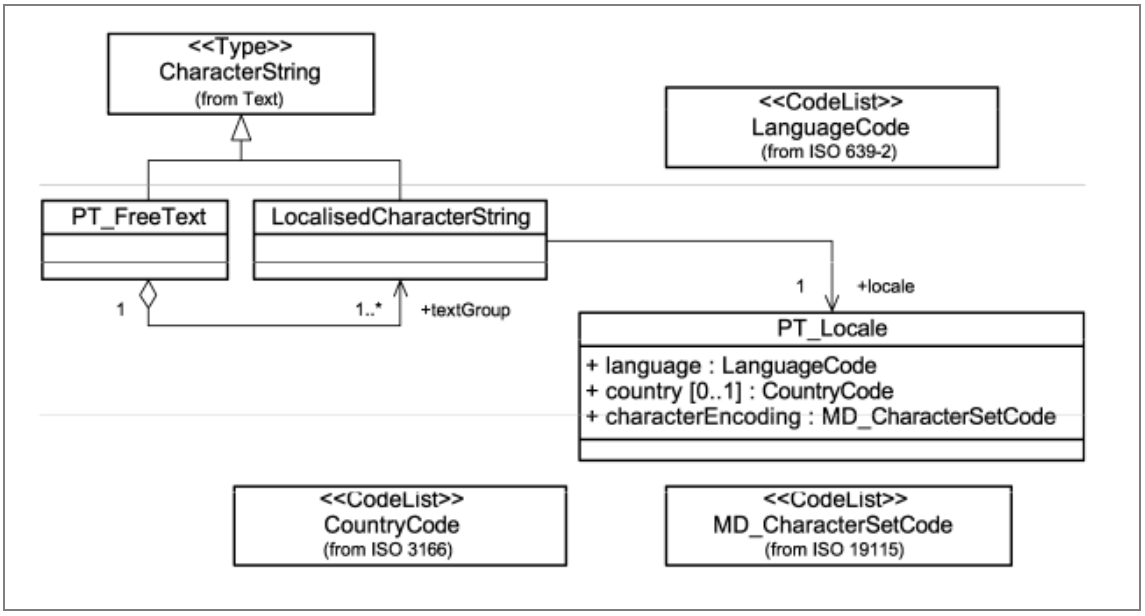


Abbildung 22: Konzeptuelles Schema zu PT_Freetext aus ISO 19139[60]

Das eigene XSchema definiert bzw. deklariert alle Datentypen und Elemente, die nötig sind, um Freitext entsprechend des konzeptuellen Schemas darzustellen und gehört dem GMD Namensraum an. Dieser Namensraum beinhaltet die Implementierung von ISO 19115 und folgt den Kodierungsregeln von ISO 19139.[61] Die Inhalte dieses XSchemas müssen in das XSchema des OpenInfRA Informationssystems importiert werden, um den entsprechenden Elementen den `PT_Freetext` Datentypen zuweisen zu können. Dabei muss zunächst Datei gmd.xsd importiert werden, da diese das Wurzelelement des GMD Namensraums darstellt.

```
<xs:import namespace="http://www.isotc211.org/2005/gmd"
schemaLocation="http://schemas.opengis.net/iso/19139/20070417/gmd/gmd.
xsd"/>
```

[60] Vgl. Norm ISO 19139 (2010) s.14
[61] Vgl. ISO/TC 211 (2013)

Anschließend kann auf die benötigten Datentypen referenziert werden. Dabei weist jedes Attribut mit dem Datentyp `PT_Freetext` auf den Datentyp `PT_FreeText_PropertyType` des GMD Namensraums.

```
<xs:element name="Name" type="gmd:PT_FreeText_PropertyType"/>
```

Die Definition dieses Datentyps, sowie weitere nötige Elemente und Datentypen sind unter Anlage A.4 aufgeführt. Wie in Abbildung 22 ersichtlich ist, weist jede *PT_FreeText* Klasse eine Beziehung *textGroup* zur Klasse *LocalisedCharacterString* auf. Diese Beziehung besagt, dass jede *PT_FreeText* Instanz mehrere lokalisierte Zeichenketten (LocalisedCharacterString) aufweisen kann. Wobei jeder Zeichenkette eine Sprachumgebung (PT_Locale) zugeordnet ist, in der sich diese Zeichenkette befindet. Diese Sprachumgebung wird wiederum durch die Angabe eines Sprachcodes, Ländercodes und einer Zeichenkodierung definiert. Diese Codes basieren dabei ebenfalls auf ISO Normen und werden in einer Codeliste geführt. Da jede XML Datei immer nur Daten desselben Zeichensatzes unterstützen kann, ist es möglich alle lokalisierten Zeichenketten einer Sprachumgebung zu gruppieren. Dabei steht das Konzept eines Übersetzungscontainers aus der ISO 19139 zur Verfügung (Abbildung 23).

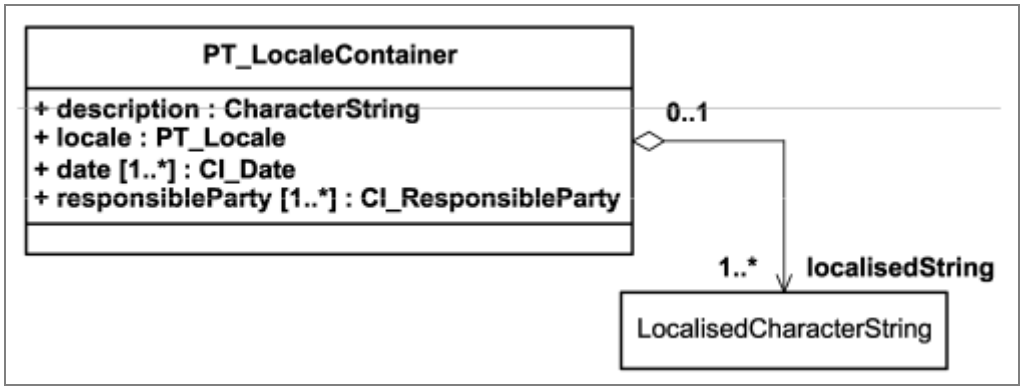


Abbildung 23: Übersetzungscontainer[62]

Jeder Übersetzungscontainer bildet eine separate XML Datei. Innerhalb jeder Datei sind eine Beschreibung, eine Sprachumgebung, das Entstehungsdatum und die Verantwortliche Stelle gegeben. Weiterhin beinhaltet jedes Dokument die entsprechenden Zeichenketten.

Auf Grundlage der Konzepte des PT_FreeText und des Übersetzungscontainers werden alle Attribute des UML Anwendungsschemas implementiert, die den Datentyp *PT_FreeText* aufweisen. Der entsprechende Auszug aus dem XSchema Dokument des Freitextes nach ISO 19139 ist unter Anlage A.4 aufgeführt.

---

[62] Vgl. Norm ISO 19139 (2010) s.15

3.5 Instanzdokument

In diesem Kapitel wird ein Beispielinstanzdokument erläutert. Um das gegebene UML Anwendungsschema in vollem Umfang umsetzen zu können, ist es nötig mehrere XML Dokumente zu erzeugen. Zum einen das Hauptdokument, welches die Struktur der UML Klassen umsetzt. Zum anderen ist ein XML Dokument nötig, in dem die Wertelistentypen und –einträge geführt werden. Weiterhin sind XML Dokumente nötig um das PT_FreeText Konzept umzusetzen. Dabei enthält ein Dokument alle Codelisten die nötig sind um die Sprachumgebung einer Zeichenkette zu definieren. Zusätzlich sind weitere Dateien nötig, welche die Übersetzungscontainer beinhalten. Dabei wird für jede Sprachumgebung ein separater Container und somit auch eine separate XML Datei erzeugt.

Bevor die Instanzdokumente erläutert werden, ist es jedoch nötig den Begriff XPointer zu klären, da diese XML Sprache in den Dokumenten häufig Anwendung findet. XPointer (XML Pointer Language) wurde vom W3C entwickelt und ermöglicht das Adressieren von Teilen eines XML Dokumentes und verwendet XPath Ausdrücke um innerhalb dieser Dokumente navigieren zu können (Abbildung 24).[63]

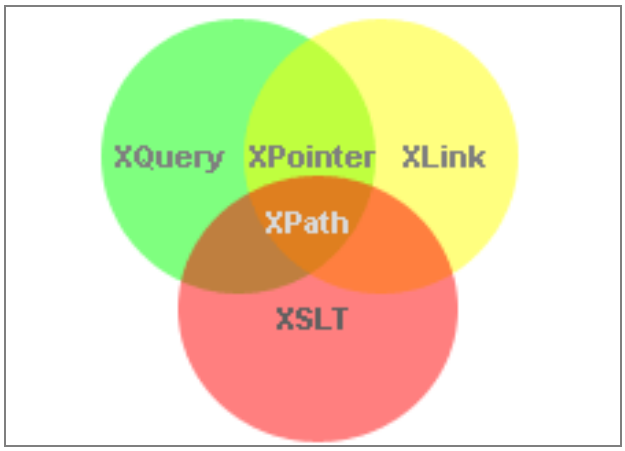


Abbildung 24: Beziehung XPointer – Xlink – XPath[64]

Dabei werden im Vergleich zu XLinks die URI Verweise erweitert, da XLinks lediglich auf das komplette Dokument referenzieren können und XPointer hingegen auf bestimmte Teile dieses Dokuments (Abbildung 25).

[63] Vgl. Refsnes Data (2013)
[64] Vgl. Refsnes Data (2013)

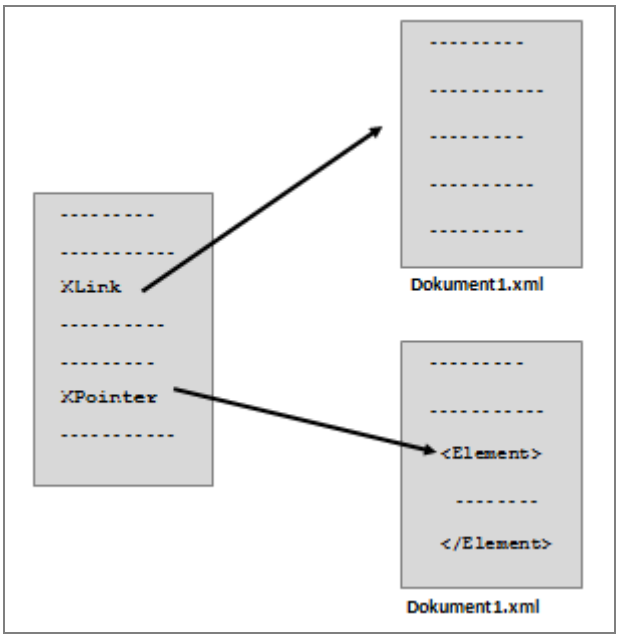


Abbildung 25: Unterschied XLink – Xpointer

XPointer stellt dabei die drei folgenden Möglichkeiten der Adressierung bereit:[65]

- Bare-Names
- Child Sequence
- Full XPointer

Die Verwendung von **Bare-Names** stellt dabei die einfachste Option dar. Die Elemente werden über ihre angegebene ID identifiziert. Die ID des zu adressierenden Elementes wird direkt am Ende der URI angegeben durch Verwendung des „#“ Zeichens. Beispielsweise kann das Element

```
<root>
   <element id="Titel">
      <unterelement1>Titel neu</unterelement1>
      <unterelement2>Titel alt</unterelement2>
   </element>
</root>
```

aus dem Dokument `Dokument.xml` mit folgender URI

`Dokument.xml`**`#Titel`**

referenziert werden. Dabei wird im Dokument direkt auf das Element mit der angegeben ID gezeigt (*engl*. to point).[66]

[65] Vgl. Wruck (2000)
[66] Vgl. Wruck (2000)

Bei der Nutzung der **Child Sequence** Option werden Elemente hingegen über Indexzahlen adressiert. Die einzelnen Zahlen werden dabei durch das „/“ Zeichen getrennt. Auf das `<element>` Element aus obigem Beispiel wird dabei durch folgende URI referenziert:

`Dokument.xml`**`#/1/1/2`**

Sozusagen wird das zweite Unterelement des ersten Kindelementes des Root Elements adressiert (Titel alt). Weiterhin ist es möglich Child Sequences und Bare-Names zu kombinieren. Demzufolge könnte auf das zweite Unterelement mittels der URI

`Dokument.xml`**`#Titel/2`**

referenziert werden.[67] Bei der Verwendung der **Full XPointer** Möglichkeit werden komplexe Verweise mittels des xpointer()-Schemas erstellt mit denen auf bestimmte Teile des Dokumentes referenziert werden kann. Die erweiterte URI wird dabei wie folgt zusammen gesetzt:

`Dokument.xml`**`#xpointer(Ausdruck)`**

Der genannte Ausdruck kann dabei ein XPath Ausdruck sein, mit dem der gewünschte Teil adressiert wird oder der Verweis auf Punkte oder Bereiche im XML Dokument. Mit der erweiterten URI

`Dokument.xml`**`#xpointer(xpointer(start-point(//unterelement1)))`**

kann so auf den ersten Punkt innerhalb des ersten `unterelement` Elements verwiesen werden. Weiterhin ist es z.B. möglich auf den Bereich unmittelbar vor dem Start-Tag des ersten Unterelements bis unmittelbar nach dem End-Tag dieses Elementes zu adressieren über folgende URI: [68]

`Dokument.xml`**`#xpointer(xpointer(range(//unterelement1)))`**

Innerhalb der nachfolgend dargestellten Dokumente wird nur die Bare-Names Option verwendet, da alle adressierten Elemente Identifikatoren aufweisen und so das Nutzen dieser einfachen Option ausreichend ist.

3.5.1 Hauptdokument

Innerhalb des Hauptdokumentes wird der Großteil der Struktur des UML Anwendungsschemas umgesetzt. Dabei wird zunächst das Wurzelelement `Datensatz` aufgeführt. Als Namensraumpräfix wird die Abkürzung OpenInfRA verwendet. Innerhalb des öff-

[67] Vgl. Wruck (2000)
[68] Vgl. Klippstein (2007)

nenden Tags des Wurzelelementes müssen alle Namensräume angegeben werden, die zur Verwendung von Freitexten, Links usw. benötigt werden. Weiterhin muss an dieser Position ebenfalls der Speicherort des XSchemas (`schemaLocation`) angegeben werden, um eine Validierung der XML Datei gegen das XSchema vornehmen zu können. Der Speicherpfad wird dabei als relativer Pfad angegeben.

```
<OpenInfRA:Dataset xmlns:OpenInfRA="OpenInfRA"
xmlns:gmd="http://www.isotc211.org/2005/gmd"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="OpenInfRA ../XSD/OpenInfRA_xsd.xsd">
```

Innerhalb des Wurzelelementes werden folgend alle UML Klassen und der Datentyp *Projekt* als Elemente aufgeführt. Die Reihenfolge wurde dabei im XSchema Dokument festgelegt. Jedes dieser Elemente weist ein Identifikator in Form eines Attributes (`id` Attribut) auf, um Referenzen auf dieses Element zu ermöglichen. Der Attributwert des `id` Attributes setzt sich dabei aus dem Namen der Klasse, dem Wort „ID“ und dem Wert des ID Elementes bzw. des in der Datenbank enthaltenen `id` Wertes zusammen.

```
<Attributtyp id="AttributtypID_1">
```

Klassen, die im UML Anwendungsschema und auch innerhalb der Datenbank kein *Id* Attribut aufweisen, wird während der Stylesheet Transformation eine ID zugewiesen. Im speziellen betrifft das die Klassen *AttributtypGruppe*, *Thema*, *AttributtypenZurThemenausprägung* sowie *BeziehungstypenZurThemenausprägung*. Die Instanzen der Klassen *AttributtypGruppe* und *Thema* erhalten dabei als ID eine fortlaufende Nummer beginnend bei Eins. Den anderen beiden Klassen wird eine, aus den IDs der *Themenausprägung* und des *Attributtyps* bzw. *Beziehungstyps* zusammengesetzte, ID zugewiesen, um so einen eindeutigen Identifikator zu erhalten. Dem Klassenelement werden anschließend alle Attribute und Beziehungen, die im UML Anwendungsschema definiert sind, als Unterelemente hinzugefügt. Dabei wird unterschieden, ob ein Element einen einfachen vordefinierten Datentyp wie z.B. *integer*, einen komplexen Datentyp wie z.B. *Multiplizität*, den Datentyp *PT_FreeText* oder die Datentypen *Wertelisteneintrag* bzw. *Wertelistentyp* aufweist oder ob ein Elemente eine Beziehungen darstellt. Einfache Datentypen werden im Instanzdokument auch auf einfache Art und Weise umgesetzt, indem den Elementtags ein Attributwert des entsprechenden Datentyps zugewiesen wird.

```
<ID>1</ID>
```

Elementen, die einen komplexen Datentyp aufweisen, werden alle in diesem Datentyp definierten Attribute und Unterelemente hinzugefügt. Diese treten wiederum als Unterelemente des Klassenattributs auf. Nachfolgend wird diese Umsetzung anhand des Attributes *Multiplizität* der Klasse *AttributtypenZurThemenausprägung* dargestellt.

```
<Multiplizitaet>
   <Min-Wert>0</Min-Wert>
   <Max-Wert>1</Max-Wert>
</Multiplizitaet>
```

Die Implementierung des Datentyps *PT_FreeText* wird im Insatnzdokument wie folgt vorgenommen:

```
<Name>
   <gmd:PT_FreeText>
      <gmd:textGroup xlink:href="locale_1.xml#41"/>
   </gmd:PT_FreeText>
</Name>
```

Jedem Element, welches diesen Datentyp aufweist, wird dabei ein Unterelement `PT_FreeText` mit Präfix `gmd` zugewiesen. Die Angabe des Präfixes ist notwendig, da andernfalls die Definition des `PT_Freetext` Datentyps, statt im `gmd` Namensraum, innerhalb des `OpenInfRA` Namensraums gesucht wird. Das Entfernen dieses Präfix würde anschließend dazu führen, dass die XML Datei nicht gültig ist. Das `FreeText` Element weist wiederum ein `textGroup` Element auf, welches dafür zuständig ist, die entsprechende lokalisierte Zeichenkette innerhalb eines Übersetzungscontainers zu adressieren. Diese Adressierung wird mit Hilfe der zuvor erläuterten XPointern vorgenommen.
Elemente vom Datentyp `Werteliste` bzw. `Wertelistentyp` verweisen auf Einträge aus der gegebenen Werteliste. Da diese Wertelisten in einem separaten Dokument geführt werden, ist es auch hier nötig über zuvor erwähnte XPointer die gewünschten Inhalte dieses Dokumentes zu adressieren. Die XPointer Angabe wird innerhalb eines Attributs vorgenommen:

```
<Datentyp WertelistenEintrag="Wertelisten.xml#EintragID_10"/>
```

oder

```
<Wertebereich WertelistenTyp="Wertelisten.xml#TypID_11"/>
```

Der Name des Attributes ist dabei abhängig davon, ob auf einen Wertelisteneintrag oder einen Wertelistentyp verwiesen wird.

Schlussendlich können Elemente auch Beziehungen repräsentieren. Dabei wird auf das Zielelement referenziert, indem dem Attribut `idref` der Identifikator des entsprechenden Zielelements hinzugefügt wird.

```
<Bez_AttributtypGruppe idref="AttributtypGruppeID_15"/>
```

Hinzu kommt, dass SKOS Beziehungen und Beziehungen, die einen bestimmten Beziehungstyp aufweisen, ein weiteres Attribut benötien. Dieses adressiert zum einen den entsprechenden Wertelisteneintrag der SKOS Beziehung und zum anderen einen bestimmten Beziehungstyp. Dabei ist anzumerken, dass SKOS Beziehungsarten über XPointer externe Elemente adressieren und Beziehungstypen über ID Referenzen auf interne Elemente verweisen:

```
<Bez_Attributtyp idref="AttributtypID_3"
SKOS_Beziehungsart="Wertelisten.xml#EintragID_31"/>
```

oder

```
<BeziehungZuAnderenThemeninstanzen idref="ThemeninstanzID_2" Bezie-
hungstyp="BeziehungstypID_1"/>
```

3.5.2 Wertelisten

Die Wertelisten werden separat zu dem Hauptdokument in einer externen XML Datei gespeichert. So bietet sich zum einen die Möglichkeit die Wertelisten aus der Datenbank auszulesen und in das geforderte XML Format zu bringen oder zum anderen das Definieren der Wertelisten direkt im XML Dokument. Bei der Nutzung der zweiten Möglichkeit werden die Wertelisten redundant geführt, da sie neben dem XML Format auch in der Datenbank gespeichert sind. Auftretende Änderungen in den Wertelisten der Datenbank müssen dann auch in die Wertelisten der XML Datei eingearbeitet werden, sodass die Wertelisten an beiden Speicherorten konsistent sind. Unterscheiden sich die Wertelisten voneinander können Probleme auftreten, wenn die Daten der Datenbank ins XML Dokument exportiert werden. So wird entweder auf falsche Einträge referenziert oder Referenzen zu gewünschten Einträgen können nicht hergestellt werden. Aus diesem Grund werden innerhalb dieser Arbeit die XML Wertelistendokumente durch Auslesen der Wertelisten aus der Datenbank generiert.

Das Wertelistendokument weist zwei Wertelisten auf. Eine Werteliste, die alle möglichen Wertelistentypen beinhaltet und eine zweite Werteliste, die die einzelnen Wertelisteneinträge aufführt. Als Wurzelelement dieser XML Datei ist das `Wertelisten` Element zu verwenden.

```
<OpenInfRA:Wertelisten xmlns:OpenInfRA="OpenInfRA"
xmlns:gmd="http://www.isotc211.org/2005/gmd"
xmlns:xlink="http://www.w3.org/1999/xlink">
```

Auch in diesem Dokument ist das Namensraumpräfix *OpenInfRA* anzugeben und die nötigen Namensräume zur Verwendung von XLinks und Freitexten sind aufzuführen. Zunächst werden alle Wertelistentypen innerhalb des `WL_Typ` Element definiert. Jeder einzelne Wertelistentyp wird über das `Typ` Element repräsentiert. Um Referenzen zu anderen Typen, Wertelisteneinträgen oder Verweise aus dem Hauptdokument ermöglichen zu können, wird jedem Typ ein `id` Attribut hinzugefügt. Der Attributwert setzt sich dabei aus dem Wort „TypID“ und dem Identifikator des Wertelistentyps zusammen. Weiterhin weist jeder Typ einen Namen und eine Beschreibung vom Datentyp `PT_FreeText` auf. Dieser wird wie im vorherigen Kapitel beschrieben, umgesetzt.

```
<Typ id="TypID_1">
   <ID>1</ID>
   <Name>
      <gmd:PT_FreeText>
         <gmd:textGroup xlink:href="locale_1.xml#ID_1"/>
      </gmd:PT_FreeText>
   </Name>
   …
</Typ>
```

Jeder Wertelistentyp weist kein, ein oder mehrere `hat_Wert` Elemente auf. Dieses Element repräsentiert die Beziehung zu den Wertelisteneinträgen. Der Verweis auf den entsprechenden Eintrag wird über eine ID Referenz gewährleistet.

```
<hat_Wert idref="EintragID_1"/>
```

Weiterhin kann jeder Wertelistentyp eine Beziehung (SKOS Beziehungsart) zu einem weiteren Wertelistentyp aufweisen. Das Zielelement und der Wertelisteneintrag, der die entsprechende SKOS Beziehungsart beinhaltet, werden dabei wieder über eine ID Referenz aufgerufen.
Analog zu den Wertelistentypen wird auch die Werteliste mit allen Einträgen geführt. Jeder Eintrag weist eine `EintragID` auf, sowie die Unterelemente `ID`, `Name` und `Beschreibung`. Zusätzlich führt jeder Eintrag noch ein Element zur Beschreibung der *Sichtbarkeit*. Analog zur `hat_Wert` Beziehung weisen die Einträge eine `gehört-Zu_Typ` Beziehung auf. Auch Wertelisteneinträge können Beziehungen untereinander aufweisen, die einer SKOS Beziehungsart entsprechen.

3.5.3 Übersetzungscontainer

Für jede definierte Sprachumgebung wird ein XML Dokument erstellt. Dabei wird jedem Dokument zunächst die entsprechende Zeichenkodierung zugewiesen. Als mögliches XSchema kann entweder das gmd.xsd Schema oder das OpenInfRA XSchema, welches das gmd.xsd Schema importiert, zugewiesen werden. Weiterhin müssen, wie bereits in Abbildung 23 dargestellt, die Attribute `description`, `locale`, `date` und `responsibleParty` sowie die Beziehung `localisedString` im XML Dokument implementiert sein. Die nachfolgende Tabelle (Tabelle 4) erläutert dabei kurz diese Elemente.

Tabelle 4: Sprachumgebungscontainer[69]

Name/ Rollenname	Definition	Verbindlichkeit/ Bedingung	Maximales Vorkommen	Datentyp	Wertebereich
PT_Locale Container	Container lokalisierter Zeichenketten. Stellt ein Mittel bereit zur Isolierung der mit einer gegebenen Sprachumgebung zusammenhängenden lokalisierten Ketten bereit	O[70]	N	Class	nachfolgende Zeilen
description	Bezeichnung der Sprache der Sprachumgebung	M	1	Character String	Free text
locale	Sprachumgebung, in der die lokalisierten Ketten des Containers ausgedrückt werden	M	1	PT_Locale	
date	Datum der Erzeugung oder Überprüfung des Sprachumgebungscontainers	M	N	CI_Date	
ResponsibleParty	Für den Sprachumgebungscontainer verantwortliche Parteien	M	N	CI_ResponsibleParty	

[69] Vgl. Norm ISO 19139 (2010) s.91
[70] O: optional; M: verbildlich (*engl*. mandatory)

Role name: localised String	Stellt die Liste lokalisierter Zeichenketten bereit, die die linguistische Übersetzung einer Menge von Textinformationen in einer gegebenen Sprachumgebung ausdrücken	M	1	Association	Localised String

Die Parameter der Sprachumgebung, der Typ des Datums und die Rolle der verantwortlichen Stelle werden aus einer Codeliste bzw. einem Codelistenkatalog entnommen. Diese Liste ist im nachfolgenden Abschnitt erläutert. Das Referenzieren auf Einträge der Liste wird wiederum über XPointer ermöglicht, wie nachfolgend anhand des `locale` Elementes dargestellt wird.[71]

```
<gmd:locale>
      <gmd:PT_Locale id="locale"
         <gmd:languageCode>
            <gmd:LanguageCode codeList="codelist.xml#LanguageCode"
             codeListValue="deu">deu</gmd:LanguageCode>
         </gmd:languageCode>
         <gmd:country>
            <gmd:Country codeList="codelist.xml#CountryCode"
             codeListValue="DE">DE</gmd:Country>
         </gmd:country>
         <gmd:characterEncoding>
            <gmd:MD_CharacterSetCode
             codeList="codelist.xml#MD_CharacterSetCode"
             codeListValue="UTF-8">UTF-8</gmd:MD_CharacterSetCode>
         </gmd:characterEncoding>
      </gmd:PT_Locale>
   </gmd:locale>
```

Das Attribut `codeList` referenziert dabei auf die entsprechende Codeliste des Kataloges. Das Attribut `codeListValue` verweist auf einen Eintrag dieser Liste. Auf die definierte Sprachumgebung kann anschließend jede lokalisierte Zeichenkette referenzieren:

[71] Vgl. Norm ISO 19139 (2010) s.120, 121

```
<gmd:localisedString>
   <gmd:LocalisedCharacterString id="ID_1" locale="#locale">
      WL_Einheit
   </gmd:LocalisedCharacterString>
</gmd:localisedString>
```

3.5.4 Codeliste

Alle Codelisten, die für das zuvor erläuterte Übersetzungsdokument benötigt werden, sind in einem Katalog definiert. Jede Codeliste weist einen Namen, sowie eine Beschreibung und Definition der entsprechenden Werte auf. Das konzeptuelle Schema des Codelistenkatalogs ist in nachfolgender Abbildung (Abbildung 26) dargestellt.[72]

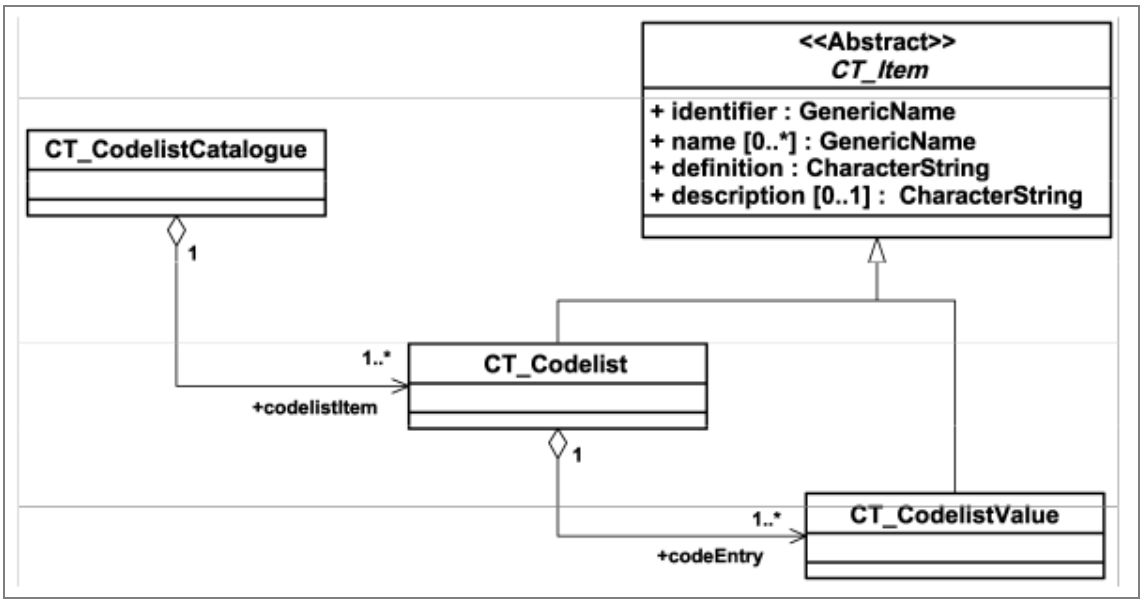


Abbildung 26: Anwendungsschema des Codelisten-Katalogs[73]

Die zugehörigen Elementdeklarationen und Datentypdefinitionen werden aus dem *gmx* Namensraum geladen. Jeder Katalog weist einen Namen, einen Gültigkeitsbereich, eine Version und ein entsprechendes Datum dieser Version auf. Diese Eigenschaften werden von der Abstrakten Klasse *CT_Catalogue* vererbt (Abbildung 27).

[72] Vgl. Norm ISO 19139 (2010) s.20
[73] Vgl. Norm ISO 19139 (2010) s. 23

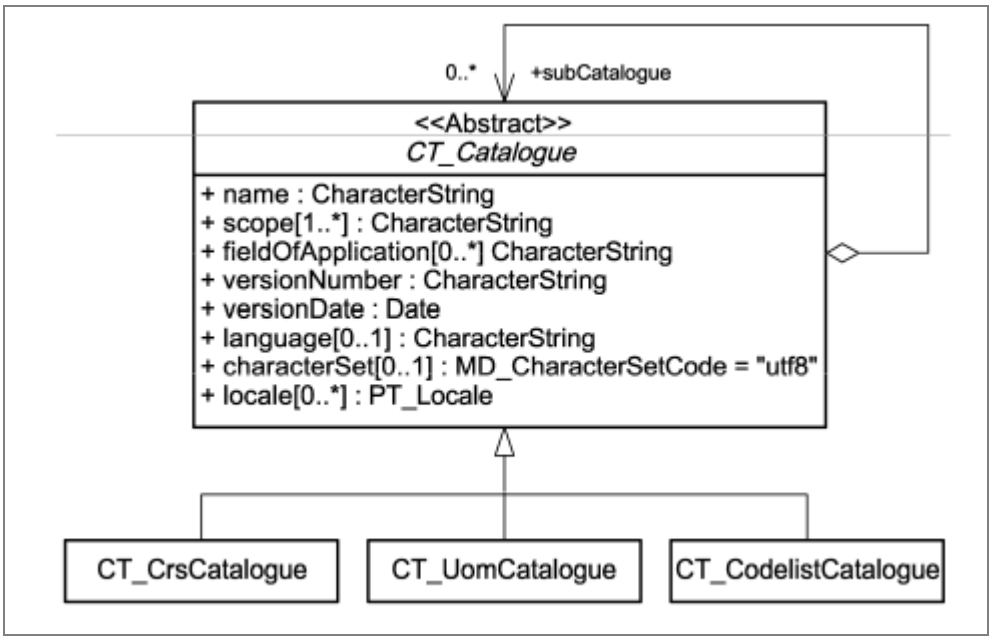


Abbildung 27: Anwendungsschema des CT_Catalogue[74]

Der Codelistenkatalog, der für die OpenInfRA XML benötigt wird, implementiert fünf Codelisten. Die Codeliste weist alle Zeichenkodierungen nach ISO 19115 auf. Weiterhin ist eine Codeliste implementiert, die einige Sprachcodes nach ISO 639-2 beinhaltet. Die dritte Codeliste weist einige Ländercodes nach ISO 3166-2 auf. Weiterhin ist eine Liste mit allen möglichen Typen implementiert, die ein Datum beschreiben können, sowie eine Liste, die alle Rollen beinhaltet, die eine Verantwortliche Stelle aufweisen kann. Jede Codeliste ist durch das Element `codelistItem` deklariert. Die Einträge einer Codeliste werden im Element `codeEntry` geführt. Jede Codeliste, sowie jeder Eintrag ist durch einen Identifikator beschrieben.

```
<gmx:codelistItem>
   <gmx:CodeListDictionary gml:id="CountryCode">
      <gml:identifier codeSpace="ISO 3166-2">CountryCode
      </gml:identifier>
      <gmx:codeEntry>
         <gmx:CodeDefinition gml:id="CountryCode_UK">
            <gml:identifier codeSpace="ISO 3166-2">UK</gml:identifier>
         </gmx:CodeDefinition>
      </gmx:codeEntry>
      …
   </gmx:CodeListDictionary>
</gmx:codelistItem>
```

[74] Vgl. Norm ISO 19139 (2010) s.20

Dieses XML Dokument wird einmalig erstellt und nicht aus den Daten der Datenbank generiert, da sich der Inhalt der Listen Zeichenkodierung, Länder- und Sprachcodes nach ISO Normen richtet und so fest vorgeschrieben ist. So ist sichergestellt, dass sich die Daten der Listen im Vergleich zu den Daten der Datenbank, in der Art der Darstellung, nicht unterscheiden, vorausgesetzt die Listen enthalten alle benötigten Einträge.

Alle Beispiele der Instanzdokumente sind unter Anlage A.7 aufgeführt.

4 Datenbankexport

Entsprechend des, im Kapitel 3.4 generierten, XML Schemas sollen anschließend die Daten der vorliegenden Datenbank überführt werden. Dazu werden die Möglichkeiten des XML Exports der Daten aus der Datenbank diskutiert. Da die exportierten Daten nicht entsprechend des generierten XSchemas bereit gestellt werden, ist es nötig die Daten in dieses zu transformieren. Nachfolgend werden alle nötigen Schritte erläutert, um schlussendlich ein XML Dokument zu erhalten, welches gültig zu dem XML Schema ist.

4.1 XML Export

Um die Daten der PostgreSQL/PostGIS Datenbank in ein XML Dokument zu exportieren, stehen einige Möglichkeiten bereit. Mit Hilfe von entsprechnder Software ist es möglich die Daten im XML Format zu exportieren. Jedoch müssen diese Programme kostenpflichtig erworben werden. Da alternative Optionen zur Verfügung stehen, wird diese Möglichkeit nicht näher erläutert. Eine Alternative wird dabei durch PostgreSQL/PostGIS selbst bereitgestellt, indem über das Absetzen von SQL Befehlen ein XML Export bewirkt wird. Dabei existieren unterschiedliche Befehle, die angewendet werden können. Diese werden nachfolgend erläutert. Nach dem Export der Daten in das XML Format ist es mit unter nötig diese mit Hilfe einer Stylesheet Transformation in das zuvor erstellt XML Schema zu bringen.

1. Möglicher SQL Befehl

Zum einen kann durch Absetzen des nachfolgenden Befehls[75] bewirkt werden, dass das ausgewählte Datenbankschema in das Format XML überführt und als Ergebnis angezeigt wird.[76]

```
schema_to_xml(schema name, nulls boolean, tableforest boolean,
targetns text)
```

Der Parameter `schema` gibt dabei das Datenbankschema an, das in eine XML Struktur überführt werden soll. Über den Parameter `nulls` wird festgelegt, ob das Ergebnis Nullwerte beinhalten soll. [77]

```
<columnname xsi:nil="true"/>
```

[75] Abgesetzt in pgAdmin Version 1.16.1
[76] Vgl. The PostgreSQL Global Development Group (2013)
[77] Vgl. The PostgreSQL Global Development Group (2013)

Mit Hilfe des Parameters `tableforest` wird die Struktur der XML Datei beeinflusst. Ist dieser Paramater auf den Wert „false“ gesetzt, weist die generierte XML folgende Struktur auf:[78]

```
<schemaname>
   <tablename>
     <row>
       <columnname1>data</columnname1>
       <columnname2>data</columnname2>
     </row>
     <row>
       ...
     </row>
     ...
   </tablename>
   ...
</schemaname>
```

Wird der Parameter `tableforest` auf den Wert „true“ gesetzt, wird die XML auf nachfolgende Art erzeugt:[79]

```
<schemaname>
   <tablename>
     <columnname1>data</columnname1>
     <columnname2>data</columnname2>
   </tablename>
   <tablename>
     ...
   </tablename>
   ...
</schemaname>
```

Diese beiden Möglichkeiten unterscheiden sich in der Darstellung der Tabellenzeilen. Diese werden entweder über das `row` Element beschrieben oder über Angabe des Tabellennamens. So ist im ersten Beispiel jeder Tabellenname nur einmal aufzufinden und im zweiten Beispiel ist jeder Tabellenname so oft aufgeführt, wie die Tabelle Zeilen aufweist. Das `row` Element entfällt in diesem zweiten Beispiel. Der vierte Parameter des `schema_to_xml` Befehls bietet die Möglichkeit für die Zieldatei einen Zielnamensraum festzulegen. Soll kein Zielnamensraum definiert werden, so kann dieser Parame-

[78] Vgl. The PostgreSQL Global Development Group (2013)
[79] Vgl. The PostgreSQL Global Development Group (2013)

ter leer gelassen werden. Im konkreten Beispiel sehen der SQL Befehl und ein Auszug aus dem zugehörigen Ergebnis wie folgt aus:

SQL Befehl

```
select schema_to_xml('projektdatenbank_v9', false, true,'')
```

Ergebnis

```
<projektdatenbank_v9 xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<!—Leerzeile →
<Landkodierung>
   <!—Leerzeile →
  <Id>1</Id>
  <Landkodierung>DE</Landkodierung>
</Landkodierung>
<!—Leerzeile →
<Landkodierung>
  <!—Leerzeile →
  <Id>6</Id>
  <Landkodierung>TR</Landkodierung>
</Landkodierung>
…
</projektdatenbank_v9>
```

Nullwerte werden in diesem Fall nicht zugelassen, sodass Zellen, die keinen Inhalt aufweisen nicht als Element auftreten. Das Ergebnis ist dabei in einer Zelle der Ergebnistabelle aufgeführt (Abbildung 28).

	schema_to_xml xml
1	<projektdatenbank_v9 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <Attributtyp> <Id>1</Id> <Name>41</Name> <Beschreibung>12</Beschreibung> <Datentyp>12</Datentyp>

Abbildung 28: Ergebnistabelle der *schema_to_xml* Abfrage[80]

[80] Auszug aus pgAdmin Version 1.16.1

Anschließend muss der Inhalt dieser Tabelle in eine XML Datei gespeichert werden. Dabei gibt es zum einen die Option das Ergebnis der *select* Anfrage z.B. über das *pgAdmin* Menü in eine Datei auszuführen. Zum anderen besteht die Möglichkeit das Ergebnis über einen SQL Befehl in eine Datei zu speichern. Dieses Verfahren bietet den Vorteil, dass sowohl das Erzeugen der XML Struktur, als auch das Ausführen des Ergebnisses in ein XML Dokument mit einem SQL Befehl abgearbeitet werden kann und ein erneutes Eingreifen durch den Nutzer nicht nötig ist. Die entsprechende Abfrage ist nachfolgend aufgeführt:

```
COPY (select schema_to_xml('projektdatenbank_v9', false, true,'')) TO
'absolute Pfadangabe\test.xml' WITH CSV QUOTE ' '
```

Durch den `copy` Befehl wird das Ergebnis der `select` Abfrage eine XML Datei kopiert. Die einzelnen Zeilen und Spalten der Ergebnistabelle sind dabei durch Kommas getrennt. Da das Ergebnis jeweils nur eine Zeile und Spalte aufweist, werden Kommas nicht zur Zieldatei hinzugefügt. Weiterhin ist zu beachten, dass der Ordnerpfad des Zielverzeichnisses immer als absolute Pfadangabe aufgeführt ist, da der `copy` Befehl eine relative Pfadangabe nicht zulässt. Zudem werden bei dieser Exportmöglichkeit standartmäßig Anführungszeichen um die komplette XML Struktur gesetzt. Dadurch werden später Probleme beim Anwenden der Stylesheet Transformation verursacht, da diese Datei nicht wohlgeformt ist. Aus diesem Grund ist es nötig mit Hilfe des `QUOTE` Parameters sicherzustellen, dass diese Anführungszeichen nicht generiert werden. Auffällig ist, dass im Ergebnisdokument viele Leerzeilen enthalten sind, wie bereits im vorherigen Beispiel dargestellt. Diese Leerzeilen haben jedoch keine Auswirkungen auf die Stylesheet Transformation, da dort direkt die XML Elemente angesprochen werden.

2. Möglicher SQL Befehl

Weiterhin kann eine XML Struktur auch durch Erzeugen der einzelnen XML Elemente mittels SQL Abfragen generiert werden. Dabei ist es möglich gezielt auf die zu erzeugende Struktur einzuwirken, denn dieses Verfahren richtet sich nach keinem speziellen Muster der XML Generierung. Jedes XML Element und Attribut wird separat mit Hilfe der SQL Befehle `xmlelement` und `xmlattributes` erstellt. Durch eine Verschachtelung dieser Befehle können so Unterelemente erzeugt werden, wie in folgendem Beispiel anhand der Tabelle Projekt dargestellt wird.

```
SELECT XMLELEMENT (NAME Projekt,
            XMLELEMENT (NAME ID, p."Id"),
            XMLELEMENT (NAME Name,
               XMLELEMENT (NAME PT_FreeText,
                  XMLELEMENT (NAME textGroup,
                     xmlattributes(p."Name" as locale)
                  )
               )
            )
            …
       )
from projektdatenbank_v9."Projekt" p;
```

Jedem Element werden dabei ein Name und ein entsprechender Wert zugewiesen. Dieser Wert kann wieder ein neues Element oder ein Attribut sein. Das erzielte Ergebnis ist nachfolgend aufgezeigt.

```
<projekt>
   <id>1</id>
   <name>
      <pt_freetext>
         <textgroup locale="52"/>
      </pt_freetext>
   </name>
</projekt>
```

Groß- und Kleinschreibung wird dabei nicht beachtet. Weiterhin können Namensraumpräfixe nicht umgesetzt werden. Diese beiden Punkte müssten deshalb anschließend über eine Stylesheet Transformation berichtigt werden.

3. Möglicher SQL Befehl

Desweiteren können auch einzelne Tabellen in einer XML Struktur dargestellt und in eine Datei exportiert werden. Umgesetzt wird diese Option mit Hilfe des Befehlt `table_to_xml`. Dieser Befehl weist dabei dieselben Parameter die der bereits beschriebene `schema_to_xml` Befehl auf und stellt so die gleichen Ausgabemöglichkeiten wie dieser bereit. Die Befehle unterscheiden sich nur hinsichtlich der Tabellenanzahl, die Ausgegeben werden. Der `schema_to_xml` Befehl gibt alle Tabellen des angegebenen Schemas aus. Der Befehl `table_to_xml` erstellt nur die XML Struktur der angegeben Tabelle. Um die gesamten Daten der Datenbank zu exportieren ist dieser Befehl deshalb nicht geeignet.

4. Möglicher SQL Befehl

Neben dem Exportieren einzelner Tabellen, können auch Ergebnisse einer *select* Anfrage in einer XML Struktur abgebildet werden. Dazu wird beispielsweise folgender Befehl abgesetzt:

```
select query_to_xml('select * from projektdatenbank_v9."Projekt"',
false, true, '')
```

Das Ergebnis der s*elect* Abfrage wird mit den angegebenen Parametern in eine XML Datei überführt. Diese Parameter beschreiben, wie bei den vorherigen Befehlen, den tableforest, Nullwerte und den Zielnamensraum. Diese Möglichkeit ist ebenfalls nicht geeignet um alle Daten eines Datenbankschemas zu exportieren.

Um den kompletten Datensatz des Datenbankschemas in das XML Format zu exportieren wird die erste Möglichkeit angewendet, da so alle Tabellen des angegebenen Schemas gespeichert und anschließend über eine Stylesheet Transformation in das nötige XML Schema gebracht werden. So muss bei Änderungen des XML Schemas nur das Transformationsdokument angepasst werden. Die zweite Möglichkeit bietet bereits beim erzeugen der XML Struktur die Option, die XML Datei hinsichtlich des XSchemas anzupassen. Jedoch ist es nicht möglich die Struktur komplett entsprechend des XML Schemas zu generieren, da u.a. keine Namensraumpräfixe unterstützt werden. So ist es in jedem Fall nötig zusätzlich eine Transformation anzuwenden. Bei Änderungen des XML Schemas muss so zum einen der Exportbefehl und zum anderen die Transformationsdatei angepasst werden. Aus diesem Grund wird der Export durch Anwenden des ersten SQL Befehl durchgeführt. Zu beachten ist jedoch, dass bei allen Möglichkeiten die Zeichenfolge „_x“ (z.B. in Attributtyp_**x**_Attributtyp) im XML Dokument als Unicode „x005F“ umgesetzt wird (z.B. Attributtyp_**x005F**_x_Attributtyp). Eine Umbenennung der Zeichenfolge zu „_X“ würde diese Änderung umgegen. Andernfalls ist es auch möglich innerhalb der Stylesheet Transformation den Elementnamen, welcher diesen Unicode aufweist, zu adressieren. Aus diesem Grund stellt diese Änderung der Zeichen im Transformationsdokument kein Problem dar, muss aber bei der Adressierung berücksichtigt werden.

4.2 StyleSheet Transformation

Die exportierte XML Datei muss anschließend umstrukturiert werden, sodass sie dem OpenInfRA XSchema entspricht. Dazu ist eine Stylesheet Transformaton nötig. Zunächst werden dazu einige Grundlagen erläutert. Um die Übersichtlichkeit zu gewähr-

leisten, wird für jede Zieldatei ein separates Transformationsdokument erzeugt. So bewirkt ein Transformationsdokument das Generieren des Hauptdokumentes, ein weiteres das Erstellen der Wertelisten und ein drittes das Erzeugen des Übersetzungscontainer. Alle Transformationsdokumente sind unter Anlage A.7 als kommentierter Quellcode vorzufinden.

4.2.1 Grundlagen

Eine Stylesheet Transformation bzw. XSLT (XSL Transformation) ist Bestandteil der Sprache XSL (Extensible Stylesheet Language). Diese Sprache besteht aus zwei Komponenten, wobei eine Komponente zur Formatierung und die andere Komponente zur Transformation von XML Daten angewendet wird. Die Komponente zur Transformation ist unter der Abkürzung XSLT bekannt. Mit einer XSLT können XML Dateien in andere XML Dateien, durch Anwenden eines Transformationsdokumentes (*engl.* Stylesheet), umgewandelt werden. Diese Umwandlung erfolgt über einen XSLT Prozessor (Abbildung 29).[81]

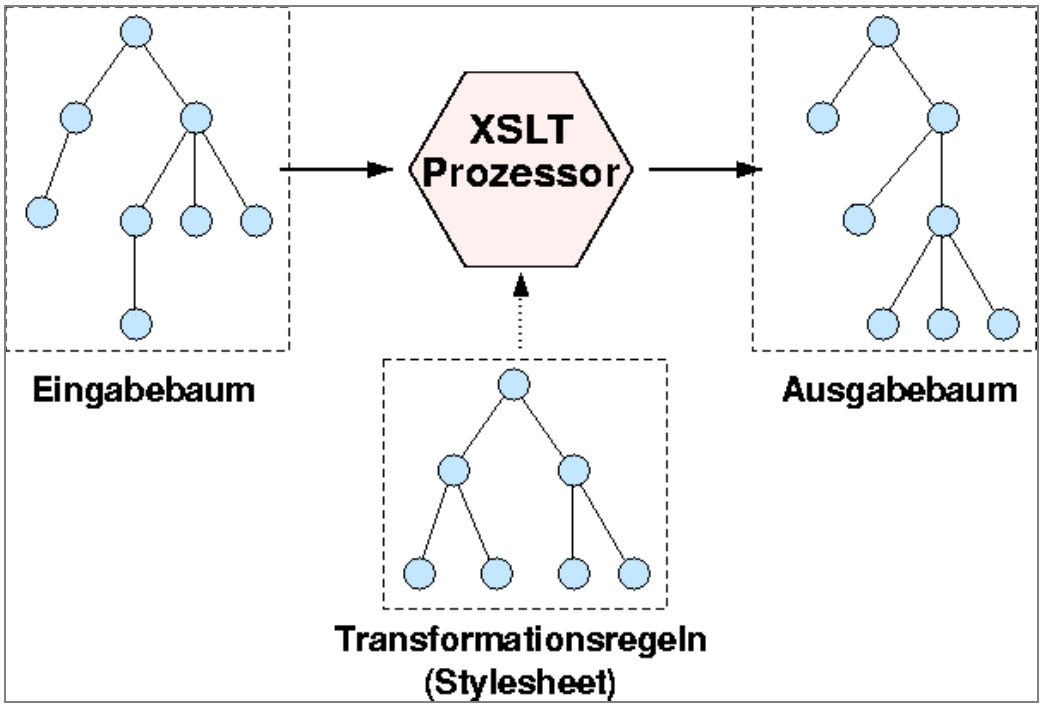


Abbildung 29: Stylesheet Transformation[82]

Sowohl die Eingabe und Ausgabedaten, sowie das Transformationsdokument weisen, wie in obiger Abbildung (Abbildung 29) ersichtlich, eine Baumstruktur auf. Das Transformationsdokument ist dabei eine wohlgeformte XML Datei, die Regeln bzw. Vorlagen

[81] Vgl. http://de.selfhtml.org/xml/darstellung/xslgrundlagen.htm
[82] Vgl. Becker (2002)

definiert, die beschreiben, was zu tun ist. Anhand dieser Regeln werden Elemente und Attribute umgewandelt und im Zieldokument dargestellt.[83] Dabei wird im Ausgangsbaum navigiert, benötigte Informationen werden selektiert, Elemente werden umstrukturiert und neue Inhalte werden hinzugefügt. Das Gerüst jeder Transformationsdatei stellt das Wurzelelement *stylesheet* dar.

```
<xsl:stylesheet  version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

   <!-- Inhalt des Stylesheet -->

</xsl:stylesheet>
```

Innerhalb des Tags des Wurzelelementes werden die genutzte XSLT Version sowie alle benötigten Namensräume angegeben. Als Stylesheet Inhalt werden ein oder mehrere Schablonen bzw. Templates aufgeführt. Jede Schablone passt dabei auf ein bestimmtes Element des Ausgangsdokuments. Das Element, auf das die Schablone passen soll wird über das *match* Attribut definiert. Anschließend sind innerhalb der Schablone Regeln angegeben, die ausgeführt werden, wenn das bestimmte Element auftritt. Diese Regeln werden durch den Aufruf bestimmter Elemente der Transformationsdatei erstellt. In der nachfolgenden Tabelle (Tabelle 5) sind alle Elemente aufgelistet, die in den drei benötigten Transformationsdokumenten genutzt werden.

Tabelle 5: Elemente der Transformationsdokumente[84]

Element	**Beschreibung**
`xsl:strip-space`	Entfernt Leerraumzeichen zwischen Elementen
`xsl:variable`	Definiert Variablen und ermöglicht den Aufruf des Variablenwertes zu einem späteren Zeitpunkt
`xsl:template`	Definiert Schablonen um den Ausgangsbaum in den Ergebnisbaum zu transformieren bzw. zu übersetzen
`xsl:result-document`	Erzeugt während der Transformation ein Ergebnisdokument nach angegeben Encoding
`xsl:element`	Erzeugt ein Element im Ergebnisdokument mit angegebenen Namen und Wert

[83] Vgl. Becker (2002)
[84] Vgl. data2type GmbH (2013)

`xsl:namespace`	Fügt einem Element einen Namensraum hinzu
`xsl:attribute`	Fügt einem Element ein Attribut mit angegebenen Namen und Wert hinzu
`xsl:apply-templates`	Ruft die Schablone auf, auf die das angegebene Element passt
`xsl:for-each`	Stellt eine For - Schleife dar, die alle, innerhalb dieser Schleife definierten, Anweisungen für jedes ausgewählte Element ausführt
`xsl:value-of`	Fügt dem Ergebnisdokument den Wert des angegebenen Elementes bzw. Attributes des Ausgangsdokumentes hinzu
`xsl:if`	Testet den angegebenen Ausdruck. Ist der Ausdruck wahr, werden folgend definierte Anweisungen ausgeführt

Innerhalb dieser Elemente können zusätzlich noch Funktionen aufgerufen werden, die den Elementinhalt bearbeiten. Die einzige hier genutzte Funktion ist dabei die `concat` Funktion. Diese ermöglicht das Verbinden mehrerer Zeichenketten.

Die nachfolgenden Transformationsdateien weisen zunächst ein Template auf, welche das Grundgerüst der Zieldatei aufbaut. Das Element, welches in diese Schablone passt ist dabei das Wurzelelement. Aus diesem Template heraus werden anschließend weitere Schablonen für andere Elemente aufgerufen. So werden beispielsweise mehrfach auftretende Unterelemente in das Zieldokument hinzugefügt. Die nachfolgenden Stylesheet Transformationen sind umfangreich, da die aus der Datenbank exportierte XML Struktur wenig der definierten XML Struktur des XSchemas entspricht. So muss die geforderte Struktur (Elemente, Attribute) innerhalb dieser Transformationsdokumente komplett aufgebaut werden.

4.2.2 Transformation Hauptdokument

Mit Hilfe dieses Transformationsdokumentes wird das Hauptdokument generiert. Dabei wird die komplette XML Struktur aufgebaut und mit den Inhalten der XML Export Datei der Datenbank gefüllt. Zunächst wird, wie zuvor beschrieben, das Grundgerüst der XSLT Datei aufgeführt. Innerhalb dieses Gerüstes wird zu Beginn die erste Schablone ausgeführt. Dabei wird als auslösendes Element das erste Element des XML Dokumentes, d.h. das Wurzelelement der XML Datei, angegeben. So werden die Regeln

dieses Templates einmalig ausgeführt, da jede XML Datei nur ein Wurzelelement beinhaltet.

```
<xsl:template match="/*">
```

Das Wurzelelement könnte auch direkt über den nachfolgenden Befehl adressiert werden. Jedoch richtet sich das Wurzelelement nach den Namen des Datenbankschemas. Dieser ist nicht fest vorgegeben und kann so von Datenbank zu Datenbank variieren. Aus diesem Grund wird das oben aufgeführte Pattern (XPath Ausdruck) zum adressieren des Wurzelelementes genutzt, da es sich nicht auf einen bestimmten Namen bezieht.

```
<xsl:template match="projektdatenbank_v9">
```

Innerhalb dieser Schablone wird zum Beginn eine XML Datei mit bestimmten Dateinamen erzeugt, in die anschließend geschrieben wird. Nachfolgend wird zum einen das Wurzelelement der Zieldatei (`OpenInfRA:Datensatz` Element) mit allen benötigten Namensräumen und dem Speicherort des XSchemas definiert. Zum anderen werden innerhalb dieses Templates weitere Schablonen aufgerufen, die das Erzeugen der Klassenelemente wie `Attributwert`, `Attributtyp` usw. bewirken. Der Aufruf der entsprechenden Schablonen erfolgt in der Reihenfolge, in der die Elemente anschließend im Zieldokument aufgeführt sein müssen. Diese Schablonen werden nicht über ein Pattern aufgerufen, sondern direkt über den Elementnamen, da diese bereits im UML Anwendungsschema definiert und so von Datenbank zu Datenbank einheitlich sind. An dieser Stelle sollte die im vorherigen Kapitel beschriebene Abänderung der Zeichen „_x“ zum Unicode „_x005F“ berücksichtigt werden. Andernfalls wird das entsprechende Element nicht erkannt. Eine Ausnahme zum Aufruf der Schablonen bilden dabei die Anweisungen zur Erstellung der AttributtypGruppen, sowie der Themen, da diese Klassen nicht in Form einer Tabelle in der Datenbank enthalten sind. Es sind lediglich Verweise auf den entsprechenden Wertelisteneintrag innerhalb der Themenausprägung bzw. Attributtyp enthalten. Deshalb müssen Schablonen für die Elemente definiert werden, die diese Verweise beinhalten. Dabei ist jedoch zu beachten, dass mehrere Themenausprägungen auf ein und dasselbe Thema verweisen können und diese Schablone so für ein Thema mehrfach durchgeführt wird und dieses Thema im Zieldokument folglich auch mehrfach aufgeführt ist. Aus diesem Grund werden innerhalb des `match` Attributes mehrfach auftretende Elemente aussortiert. Probleme ergeben sich dabei bei der Erstellung der Identifikatoren. Diese sind notwendig, damit anschließend Instanzen der Themenausprägung bzw. des Attributtyps über diese ID auf das jeweilige Element referenzieren können. Eine Möglichkeit wäre den Themen bzw.

AttributtypGruppen eine fortlaufende Nummer als ID zuzuweisen. Diese fortlaufende Nummer kann jedoch beim Erstellen des Referenzelementes in der Schablone der Themenausprägung bzw. Attributtypen nicht angesprochen werden, da diese an keiner Stelle gespeichert wird. So würden neue Nummer generiert werden, die nicht zu den alten Nummern passen. Folglich werden verfälschte Referenzen erstellt. Um das zu vermeiden werden die IDs der Wertelisteneinträge zur Erzeugung der ID der Themen und AttributtypGruppen verwendet. Dabei wird nach folgendem Schema der Identifikator generiert:

- Identifikator Thema ='ThemenID_'+ID des Wertelisteneintrags
- Identifikator AttributtypGruppe ='AttributtypGruppeID_'+ID des Wertelisteneintrags

Beim Aufruf der Schablonen zur Generierung der Klassenelemente wird zunächst dieses Klassenelement, sowie alle zugehörigen Unterelemente erzeugt. Wichtig ist dabei, dass alle Attribute des UML Anwendungsschemas, die die Kardinalität 0..1 aufweisen nicht im Exportdokument enthalten sein müssen. So muss zunächst auf die Existenz dieses Elementes geprüft werden um zu vermeiden, dass Elemente oder Attribute ohne Inhalt bzw. mit unvollständigem Wert generiert werden, wie in nachfolgendem Beispiel zu erkennen:

```
<Standardwert WertelistenEintrag="Wertelisten.xml#EintragID_"/>
```

Die Referenz auf den Wertelisteneintrag weist eine Unvollständige ID auf. Um das zu vermeiden, kann über den folgenden Befehl kann die Existenzüberprüfung erfolgen:

```
<xsl:if test="Beschreibung">
```

Ist das aufgeführte Element (Beschreibung) vorhanden, dann werden die Elemente im Zieldokument erzeugt. Weiterhin ist zu beachten, dass 1:n Beziehungen in der Datenbank abgebildet werden, indem die 1-Seite als Attribut mit Fremdschlüsselbeziehung auf der n-Seite implementiert wird. Im XML Dokument ist es jedoch möglich die Beziehung auf beiden Seiten zu implementieren. Diese bidirektionalen Beziehungen sind deshalb im XSchema vorgesehen und werden mit Hilfe der Stylesheet Transformation auch auf beiden Seiten umgesetzt. Die Implementierung der Beziehung der n-Seite zur 1-Seite kann mit wenigen XSLT Anweisungen implementiert werden, da die ID der Klasseninstanz der 1-Seite in dieser Tabelle aufgeführt ist und so lediglich selektiert werden muss. Beispielsweise weist das Element `Attributwert` die ID des zugehörigen `Attributtyps` auf. Das Element `Attributtyp` weist hingegen keine ID des Elementes `Attributwert` auf (Abbildung 30).

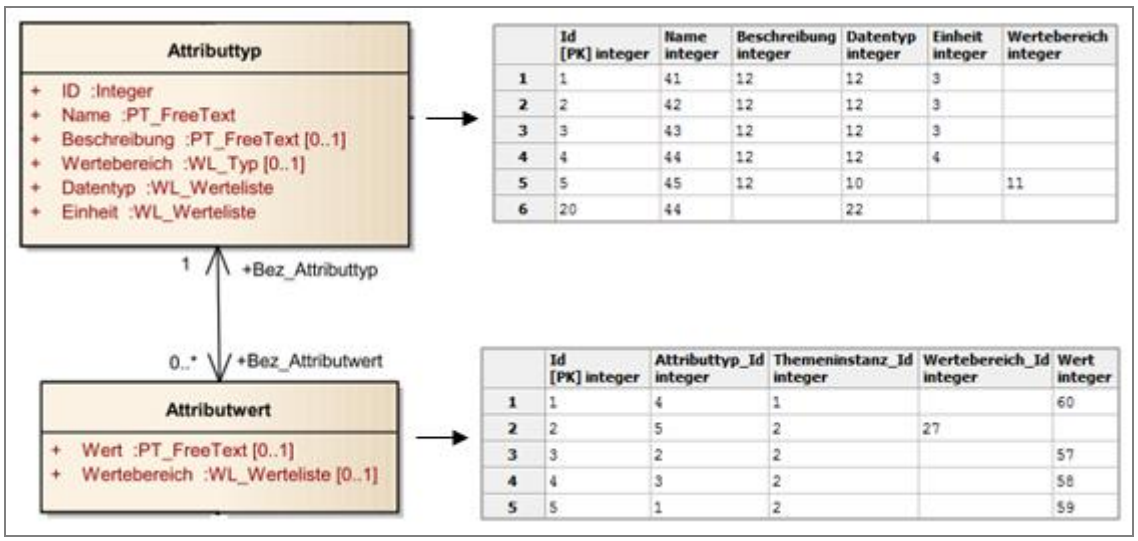


Abbildung 30: Beispiel 1:n Beziehung

Damit die Beziehung der 1-Seite zur n-Seite zuordnen werden kann, müssen z.B. beim Generieren der Attributtypen alle Attributwerte dahingehend überprüft werden, ob sie die ID dieses Attributtyps aufweisen. Ist diese ID vorhanden, wird der Identifikator des entsprechenden Attributwert Elements ausgelesen:

```
<xsl:for-each select="Attributwert">
   <xsl:if test="$id=Attributtyp_Id">
      <xsl:element name="Bez_Attributwert">
         <xsl:attribute name="idref">
            <xsl:value-of select="concat('AttributwertID_',Id)"/>
         </xsl:attribute>
      </xsl:element>
   </xsl:if>
</xsl:for-each>
```

Die n:n Beziehungen werden auf ähnlich Art und Weise implementiert. Diese Beziehungen werden in der Datenbank über Join Tabellen umgesetzt. So können diese Tabellen im Transformationsdokument ebenfalls auf die Existenz einer bestimmten ID überprüft und die nötigen Elemente generiert werden. Zu beachten ist jedoch, dass Beziehungen, die eine SKOS Beziehungsart oder einen bestimmten Beziehungstyp aufweisen nur in der Richtung implementiert werden können, in der sie in der Datenbank angegeben sind, da sich die Beziehungsart bzw. -typ nur auf diese bestimmte Richtung bezieht. Beispielsweise wird die Beziehung „Instanz 1 beinhaltet Instanz 2“ nur auf der Seite der Instanz 1 implementiert. Wird diese Beziehung mit gleichem Beziehungstyp auch auf der Seite von Instanz 2 umgesetzt, bedeutet das „Instanz 2 beinhaltet Instanz 1“. Diese Aussage ist jedoch falsch und darf so nicht implementiert wer-

den. Diese Ausnahme betrifft im konkreten Fall die SKOS Beziehungen zwischen *Attributtypen*, *Wertelisten* und *Wertelistentypen*, sowie der Beziehung von *Themeninstanzen* zu anderen *Themeninstanzen*.

Nachdem diese Transformation auf eine Datei angewendet wurde, die eine erheblich größere Anzahl von Daten[85] beinhaltet als die zuvor genutzte Testdatei, stellte sich heraus, dass die Dauer dieser Transformation nicht tragbar ist. Dieses Problem ist auf die zuvor beschriebe Umsetzung der 1:n und n:n Beziehungen zurückzuführen. Das Testen aller Elemente auf das Vorhandensein eines bestimmten Unterelements mittels `for-each` und `if` Schleifen stellt einen erheblichen Aufwand dar. Diese Transformationszeit kann um ein vielfaches verringert werden, indem das Durchlaufen der `if` Schleife umgangen wird. Das kann durch Anwenden von Schlüsseln (*engl*. key) bewirkt werden.

```
<xsl:key name="Attributwert_by_AttributtypId" match="Attributwert"
use="Attributtyp_Id" />
```

Diese Zugriffsschlüssel erleichtern dem XSLT Prozesser die Arbeit, was folglich zu einer Erhöhung der Verarbeitungsgeschwindigkeit führt. Wie in obigen Beispiel zu erkennen, wird über das Attribut *name* der Schlüssel benannt. Unter diesem Namen kann dieser Zugriffsschlüssel anschließend abgerufen werden. Das *match* Attribut gibt den Knoten bzw. das Element an, auf das der Schlüssel angewendet werden soll. Über das *use* Attribut wird anschließend definiert, auf welches Unterelement oder Attribut der Schlüssel zugreifen soll. Anschließen kann, wie in folgendem Beispiel dargestellt, der Schlüssel über die `key()` Funktion abgerufen werden.

```
<xsl:for-each
select="key('Attributwert_by_AttributtypId',$id_Attributtyp)">
```

Dabei wird innerhalb dieser Funktion der entsprechnde Schlüsselname angegeben, sowie ein Attributwert. Folglich werden nur die Einträge ausgewählt, die diesen Attributwert aufweisen.[86]

Wurden alle nötigen Schablonen ausgeführt, wird das XML Dokument geschlossen und ist im angegebenen Speicherort vorzufinden.

---

[85] XML Datei von ca. 30MB

[86] Vgl. http://de.selfhtml.org/xml/darstellung/xsltelemente.htm#key

4.2.3 Transformation Wertelisten

Beim auslesen der Werteliste aus der Datenban, ist es auch hier nötig eine Transformation vorzunehmen. Die entsprechenden Informationen befinden sich im selben Exportdokument wie die Daten des Hauptdokumentes. Deshalb werden diese Informationen in einem Transformationsdokument selektiet. Das Stylesheet weist dabei die gleiche Struktur auf, wie das zuvor beschriebe Dokument zur Transformation des Hauptdokumentes. So wird zunächst das *xsl:stylesheet* Wurzelelement angegeben und anschließend die erste Schablone definiert, die auch hier wieder das Wurzelelement des Exportdokumentes adressiert. Innerhalb dieses Templates wird zunächst die Zieldatei mit dem Namen *wertelisten.xml* angelegt. Nachfolgend wird das Wurzelelement der Zieldatei mit allen benötigten Namensräumen generiert (`OpenInfRA:Wertelisten` Element). Anschließend wird zum einen eine Schablone aufgerufen, die alle Wertelistentypen dem Zieldokument hinzufügt und zum anderen wird ein Template ausgeführt, welches alle Wertelisteneinträge in die Zieldatei schreibt. Bei der Umsetzung von Beziehungen und Attributen mit Kardinalität 0..1 wird auf gleiche Weise vorgegangen, wie bereits beim Erzeugen des Hauptdokumentes. Das Ergebnisdokument weist dabei die in Kapitel 3.5.2 beschriebene Struktur auf.

4.2.4 Transformation Übersetzungscontainer

Neben den zuvor genannten Dateien müssen auch die Übersetzungscontainer generiert werden. Dabei ist zu beachten, dass mehrere Container und demzufolge auch mehrere Dateien erzeugt werden müssen. Die Informationen die nötig sind um die Übersetzungsdateien zu generieren, befinden sich ebenfalls im Exportdokument der Datenbank. Die entsprechende Transformationsdatei weist zunächst das Grundgerüst, bestehend aus dem `xsl:stylesheet` Wurzelelement, auf. Nachfolgend wird das erste Template definiert, das wiederum das Wurzelelement der Exportdatei adressiert. Innerhalb dieser Schablone wird für jede definierte Sprachumgebung, die auch von einer Zeichenkette verwendet wird, eine Zieldatei angelegt. Die Benennung der Zieldateien richtet sich nach folgendem Muster:

```
Dateiname = 'locale_' + Identifikator der Sprachumgebung + .xml
```

Der entsprechende Sprachcode, sowie Ländercode und Zeichenkodierung sind der Sprachumgebung über einen Identifikator zugewiesen. Deshalb müssen zunächst die benötigten Codes selektiert werden. Anschließend erfolgt das Schreiben in die jeweilige Zieldatei. Zu Beginn wird die Zeichenkodierung (*encoding*) der Zieldatei angegeben. Diese Kodierung richtet sich nach der Zeichenkodierung der Sprachumgebung, die innerhalb dieses Übersetzungscontainers definiert ist. Weiterhin wird das Wurzelele-

ment (`gmd:PT_LocaleContainer`) mit allen nötigen Namensräumen generiert. Anschließend wird dem Dokument eine Beschreibung hinzugefügt, wobei diese angibt, aus welchen Codes sich die definierte Sprachumgebung zusammensetzt. Im Anschluss wird die Sprachumgebung definiert. Nachfolgend wird das Erstellungsdatum der Datei, sowie die verantwortliche Stelle hinzugefügt. Schlussendlich werden die lokalisierten Zeichenketten, die der definierten Sprachumgebung entsprechen, hinzugefügt. Wichtig ist dabei die Generierung der ID der Zeichenkette, da diese die Adressierung der jeweiligen Zeichenkette vom Hauptdokument oder den Wertelisten ermöglichen. So kann schlussendlich das in Kapitel 3.5.3 beschriebene Ergebnis erzielt werden.

Die kompletten Transformationsdateien sind in der digitalen Anlage (Anlage A.7) vorzufinden.

4.3 Automation des Exports

Der zuvor beschriebene XML Datenbankexport erfolgt in mehreren Schritten:

1. Absetzten des Befehlt zum Export der Daten aus der Datenbank
2. Transformation des Hauptdokumentes
3. Transformation der Wertelisten
4. Transformation der Übersetzungscontainer

Sinnvoller ist es jedoch, den Exportprozess einmalig zu starten und direkt das gewünschte Ergebnis zu erhalten, ohne dass die oben genannten Zwischenschritte manuell durchgeführt werden müssen. Das kann mit Hilfe eines Batch Prozesses ermöglicht werden. Dabei wird eine Reihe von MS-DOS-Befehlen automatisiert durchgeführt. Die Batch Datei nutzt relative Pfadangaben und ist deshalb an die unter Anlage A.7 aufgeführte Ordnerstruktur gebunden. Änderungen der Odnerstruktur müssen auch im Batch Dokument umgesetzt werden. Der Batch Prozess bewirkt zunächst, dass die Ordner, welche die Ergebnis- und Zwischendokumente beinhalten, geleert werden. Dashalb ist zu beachten, dass alle benötigten Dokumente vor dem Start des Prozesses in einem anderen Verzeichnes gesichert werden, andernfalls gehen diese Dateien verloren. Der Export der Daten aus der Datenbank erfolgt über folgenden Befehl:

```
D:\PostgreSQL\9.2\bin\psql -U Benutzername -d Datenbankname -a -c "CO-
PY (select schema_to_xml('Schemaname ', false, true,'')) TO
'absolute Pfadangabe\XML_Export\Datenbank_Exportdatei
\test_v9_.xml' WITH CSV QUOTE ' ' ENCODING 'UTF-8'"
```

Die fettgedruckten Angaben des Befehls müssen dabei entsprechend angepasst werden. Aus diesem Grund werden diese Angaben zuvor durch eine Benutzereingabe abgefangen und an Variablen übergeben. Mit Hilfe des, von PostgreSQL bereitgestellten, interaktiven Datenbankterminal `psql` ist es möglich den oben aufgeführten SQL Befehl über die MS-DOS-Eingabeaufforderung abzusetzten. Dabei erfolgt einer Verbindung der Datenbank über die angegebenen Parameter. Zusätzlich wird durch den Datenbankterminal das Passwort des angegeben Benutzers abgefragt. Nach korrekter Passworteingabe wird direkt der SQL Befehl ausgeführt. Nachfolgend bewirkt die Batch Datei, dass überprüft wird, ob der Exportprozess erfolgreich durchgeführt wird, indem auf das Vorhandensein der Exportdatei im Zielverzeichnis getestet und eine entsprechende Benutzerausgabe getätigt wird. Dieser Test ist wichtig, da die weiteren Befehle auf die Existenz dieser Datei angewiesen sind. Wurde die Exportdatei generiert, erfolgt die Transformation dieser Datei in die benötigten Ergebnisdokumente. Dabei werden nacheinander alle drei Transformationsdokumente ausgeführt mit Hilde des Saxon XSLT 2.0 Prozessors[87]. Saxon ist ein Open Source Produkt und steht entweder als Implementierung für Java oder .NET zur Verfügung. Zur Durchführung der Transformationen wird folgender Befehl abgesetzt, welcher die Java Implementierung des Saxon XSLT 2.0 Prozessors nutzt.

```
java -cp Saxon\saxon9he.jar net.sf.saxon.Transform
-s:Datenbank_Exportdatei\DB_Export.xml
-xsl:XSLT\trafo_Instanzdokument.xslt
```

Mit Hilfe des `cp` Parameters wird der Speicherort der benötigten Saxon Java Dateien angegeben. Die Parameter `s` und `xsl` definieren die zu verwendende Ausgangsdatei, sowie das Transformationsdokument. Eine Angabe der Zieldatei ist nicht nötig, da diese innerhalb des Transformationsdokumentes definiert ist. Eine erfolgreiche Durchführung der Transformationen wird ebenfalls über die Existenzprüfung der entsprechenden Ergebnisdateien im Zielordner getestet und anschließend dem Benutzer mitgeteilt.

[87] Vgl. Kay (2013)

5 Diskussion einer Transformation nach CIDOC CRM XML

Das zuvor entwickelte XML Format ist zum Austausch der Daten zwischen Datenbanken, die dem OpenInfRA Anwendungsschema entsprechen. Um den Datenaustauch von strukturierten Informationen auch im Bereich des Kulurellen Erbes zu ermöglich, ist es nötig eine Transformation in das entsprechende Datenformat durchzuführen. Dazu wird zunächst die Begrifflichkeit CIDOC CRM geklärt um anschließend das benötigte Dazenformat zu erläutern.

5.1 Definition CIDOC-CRM

Das CIDOC Conceptual Reference Model

> *„[…] ist eine formale Ontologie (formalisiertes Begriffsmodell), um die Integration, Zugriffsvermittlung und den Austausch verschiedenartig strukturierter Informationen aus dem Bereich des Kulturellen Erbes zu unterstützen.“* [88]

Das CRM wurde vom Internationalen Ausschuss für Dokumentation (CIDOC - International Committee for Documentation) des Internationalen Museumsrates (ICOM - International Council of Museums) entwickelt. Im Jahr 1996 oblag diese Entwichlung zu Beginn dem CIDOC Documentation Standards Working Group (DSWG) und wurde 2000 an die CIDOC CRM Special Interest Group (SIG) übergeben.[89] Seit Ende 2006 ist die CIDOC CRM als interationale Norm (ISO 21127:2006) von der Internationalen Organisation für Normung (ISO) annerkannt.[90]

> *„Im Wesentlichen definiert das CRM in Form einer formalen Ontologie die den Datenbankschemata und Strukturen von Dokumenten zu Grunde liegende Semantik, die in der Dokumentation des Kulturellen Erbes und in der Dokumentation der Museen benutzt werden. Es ist zugleich auch beschränkt auf diese Semantik. Es definiert keine Terminologien, die typischer Weise als Daten in entsprechenden Datenstrukturen erscheinen; es erklärt jedoch charakteristische Beziehungen für ihre Verwendung. Das CRM schlägt auch nicht vor, was Kulturinstitutionen dokumentieren sollten. Vielmehr wird durch das CRM die Logik dessen erklärt, was Kulturinsti-*

[88] Lampe u. Krause u. Doerr (2010) s.9
[89] Vgl. Lampe u. Krause u. Doerr (2010) s.9
[90] Vgl. FORTH (2013)

tutionen tatsächlich derzeitig dokumentieren und wie dadurch semantische Interoperabilität ermöglicht wird." [91]

Das vorrangige Ziel des CIDOC CRM ist dabei der Austausch bzw. die Integration heterogener Daten des kulturellen Erbes.

5.2 Aufbau des CIDOC CRM

Das CRM besteht aus Klassen von Objekten, wobei alle Objekte dieser Klasse gemeinsame Eigenschaften aufweisen. Weiterhin baut sich das CRM aus Unterklassen auf, die eine Spezialisierung der Oberklasse darstellen und so zusätzliche Eigenschaften definieren. Die Eigenschaften der Oberklasse werden an die Unterklasse vererbt. Eine Unterklasse kann von einer Oberklasse, aber auch von mehreren Oberklassen erben. Eine Unterklasse weist dabei zu ihrer Oberklassen eine *„…ist ein…"* Beziehung auf. So ist eine Gitarre zum Beispiel ein Musikinstrument und die Klasse Gitarre ist eine Unterklasse der Klasse Musikinstrument. Das CRM weist so eine strikte Hierarchie auf, wobei die oberste Klasse dieser Hierarchie die Klasse *CRMEntity* ist, die jedoch nur eine formale Bedeutung hat. Diese Klasse definiert ein freies Textfeld, welches auch an alle Unterklassen vererbt wird, sodass alle Klassen des CRM ein freies Textfeld aufweisen. Jede Instanz einer Klasse kann so ein konkreter Wert zugewiesen werden. Innerhalb des CRM verfügen die Klassen neben Beziehungen zu Unterklassen auch über Beziehungen zu weiteren Klassen. Die Klasse Musikinstrument kann beispielsweise eine *„…wurde hergestellt von…"* Beziehung zur Klasse Person aufweisen. Beziehungen tragen sozugen einen Großteil der Informationen. Die Klasse, auf die über eine Beziehung verwiesen wird, wird als „Range" bezeichnet. Die Ausgangsklasse, von der aus verwiesen wird, wird hingegen „Domain" genannt. Klassen werden innerhalb des CRM als Entities bezeichnet, weshalb allen Klassenbezeichnungen der Buchstabe *„E"* vorangestellt wird, z.B. *E1 CRMEntity.* Eigenschaften werden hingegen als Properties benannt und werden mit dem vorangestellten Buchstaben *„P"* benannt, z.B *P1 is identified by (identifies).* In nachfolgender Abbildung (Abbildung 31) werden einige wichtige Klassen der insgesamt 90 Klassen[92] des CRM dargestellt.[93]

[91] Lampe u. Krause u. Doerr (2010) s.9
[92] CIDOC CRM Version 5.0.4
[93] Vgl. Deutscher Museumsbund (2004) s.2

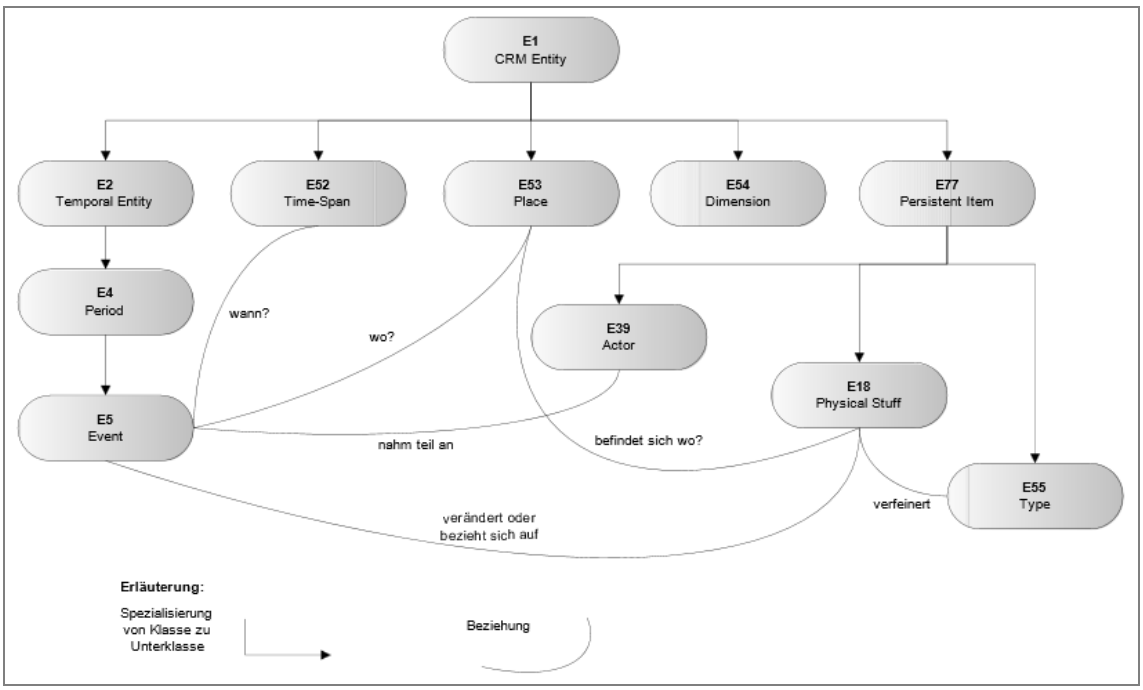


Abbildung 31: Wichtige Klassen des CIDOC CRM[94]

In obiger Abbildung werden die Klassen der obersten Ebene der Klassenhierarchie des CRM dargestellt. Folgende Grafik (Abbildung 32) bildet einige Beziehungen der insgesamt 149 Beziehungen[95] des CRM am Beispiel der Klasse *E21 Person* ab.

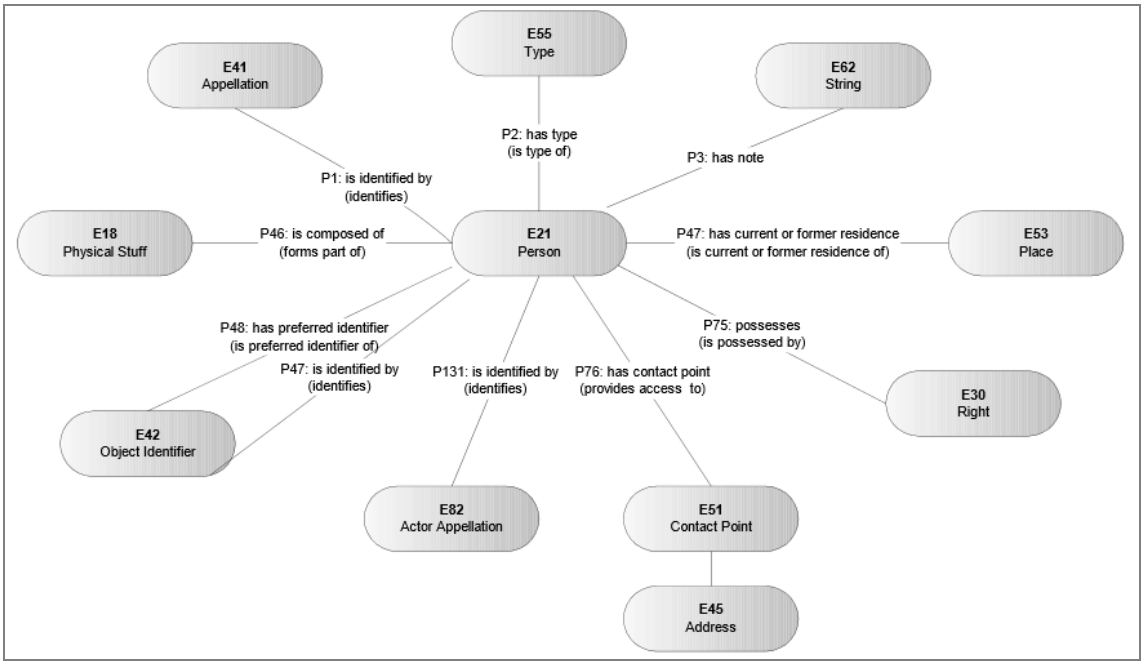


Abbildung 32: einige Beziehungen des CRM[96]

[94] Vgl. Deutscher Museumsbund (2004) s.3
[95] CIDOC CRM Version 5.0.4
[96] Vgl. Deutscher Museumsbund (2004) s.5

5.3 CIDOC CRM in XML

Das XML Format wird im CIDOC CRM als Transferformat zum Datenaustausch verwendet. Die Struktur der jeweiligen XML Instanz ist dabei nicht über ein XSchema Dokument gegeben, sondern innerhalb einer Document Type Definition (DTD) festgelegt.

DTDs definieren welche Elemente wie im Instanzdokument dargestellt werden. Dabei definiert eine DTD folgende Bausteine:[97]

- Elemente
- Attribute der Elemente
- Anordnung der Elemente im Instanzdokument
- Entities (Textbausteine)

Elemente können so auf folgende Art und Weise deklariert werden:

```
<!ELEMENT elementname inhalt >
```

Einem Element können anschließend Attribute in Form einer Liste hinzugefügt werden. Innerhalb dieser Attributliste können alle, für das Element zutreffenden, Attribute deklariert werden.

```
<!ATTLIST elementname
          Attributname typ default
          …
>
```

Im Vergleich zum XSchema stellt die DTD weniger vordefinierte Datentypen bereit. Weiterhin können Attributwerte nicht eingeschränkt werden z.B. hinsichtlich Länge und Zeichen. Auch die Verwendung von Namensräumen ist innerhalb der DTD nicht möglich.
Die CIDOC CRM DTD[98] implementiert die CRM Properties als XML Elmente. Dabei werden die Beziehungen sowohl von Domain in Richtung Range, gekennzeichnet durch den Buchstaben „F“ (*engl.* forward), als auch umgekehrt von Range in Richtung Domain, gekennzeichnet durch den Buchstaben „B“ (*engl.* backwards), umgesetzt. Die Konsistenz der Daten wird dabei durch die DTD nicht überprüft und muss durch den Nutzer sichergestellt werden, da die DTD es ermöglicht, dass jede Klasse jede Beziehung zu jeder anderen Klasse aufweisen kann. Das vollständige DTD Dokument ist

[97] Vgl. Heymann (2003)
[98] http://www.cidoc-crm.org/docs/crm_entity_plhres_shorted.dtd

unter Anlage A.6 aufgeführt. Ein XML Dokument, welches gegen die Document Type Definition valide ist, muss zunächst das Wurzelelement `CRMset` aufweisen. Dieses Wurzelelement kann anschließend beliebig viele `E1.CRM_Entity` Elemente beinhalten, welches wiederum alle Entitäten und Properties aufführt. Eine Klasse bzw. Entität wird über ein `in_class` Element definiert. Der zugehörige Wert wird im Element `Identifier` geführt. Nachfolgend ist beispielsweise ein Auszug aus einem Instanzdokument dargestellt:[99]

```
<CRMset>
   <E1.CRM_Entity>
      <Identifier>Epitaphios GE34604</Identifier>
      <in_class>E22.Man-Made_Object</in_class>
      <P1F.is_identified_by>
         <Identifier>GE 34604</Identifier>
         <in_class>E42.Identifier</in_class>
      </P1F.is_identified_by>
      <P2F.has_type>
         <Identifier>liturgical cloth</Identifier>
         <in_class>E55.Type</in_class>
      </P2F.has_type>
      <P52F.has_current_owner>
         <Identifier>Museum Benaki </Identifier>
         <in_class>E40.Legal_Body</in_class>
         <P2F.has_type>
            <Identifier>private museum</Identifier>
            <in_class>E55.Type</in_class>
         </P2F.has_type>
         <P76F.has_contact_point>
            <Identifier>Koumbari Street 1, Athens</Identifier>
            <in_class>E45.Address</in_class>
         </P76F.has_contact_point>
      </P52F.has_current_owner>
   </E1.CRM_Entity>
</CRMset>
```

Die Aufgabe besteht nun darin, die OpenInfRA Daten in dieser CIDOC CRM XML darzustellen. Zuvor ist es jedoch nötig die Fachsicht des OpenInfRA Schemas der Fachsicht des CIDOC CRM zuzuordnen. Dieses semantische Mapping wurde durch Herrn Philipp Gerth durchgeführt und steht unter Anlage A.7 bereit. Auf Grundlage dieses Mappings kann nun eine Stylesheet Transformation durchgeführt werden. Dabei dienen die auf der CIDOC Homepage bereitgestellten Beispiele[100] als Grundlage. Das nachfolgend beschriebene Transformationsdokument ist auf das Thema „Pergamon

[99] http://www.cidoc-crm.org/docs/epitafios5.0.1.xml
[100] Vgl. http://www.cidoc-crm.org/crm_mappings.html

Münzen“ angepasst und kann deshalb nur zur Transformation der Daten dieses Themas genutzt und ist nicht universell verwendet werden.
Die Transformation wird, wie die vorherigen Transformationen, auf die XML Export Datei der Datenbank angewendet. Zu Beginn des Transformationsdokumentes wird zunächst die Sprachumgebung definiert, die die Texte des `Identifier` Elementes aufweisen.

```
<xsl:variable name="locale" select='2'/>
```

Die Sprachumgebung weist zunächst den Wert „2“ auf, der in der Datenbank die Sprache Deutsch mit der Zeichenkodierung UTF-8 definiert. Zur Ausgabe des Ergebnisses in einer anderen Sprachumgebung ist es nötig diesem Parameter anzupassen. Wichtig ist dabei, dass die Id der Spachumgebung der in der Datenbank befindlichen Id entspricht. Folgend werden alle benötigten Zugriffsschlüssel definiert. Anschließend wird das Wurzelelement der Ausgangdatei über ein Template adressiert. Innerhalb dieser Schablone wird das Wurzelelement der Zieldatei generiert. Nachfolgend wird für jede Themeninstanz bzw. Münze ein `E1.CRM_Entity` Element erzeugt. Jedem `E1.CRM_Entity` Element werden die Eigenschaften und Klassen bzw. Entitäten, die unter Anlage A.7 aufgeführt sind, hinzugefügt. Klassen, für die kein entsprechender Attributwert in der Datenbank vorhanden ist, werden nicht erzeugt. Die Auswahl der Werte der Klassen bzw. Entitäten (Identifier) richtet sich nach den speziellen Ids der Attributtypen aus der Datenbank. Die Grafik, die das semantische Mapping beinhaltet (Anlage A.7), weist Entitäten auf, für die keine nähere Beschreibung vorgesehen ist. Die Klassen weisen in der CIDOC CRM XML ein leeres *Identifier* Element auf.

```
<P4F.has_time-span>
   <Identifier/>
   <in_class>E52.Time-Span</in_class>
…
```

Die Entität „E47 Spatial Coordinates“ wurde aufgrund der fehlenden Geometrie in der Datenbank nicht umgesetzt.

6 Ausblick

Die nachfolgend beschriebenen Überlegungen wurden, aufgrund der zeitlichen Beschränkung der Arbeit bzw. aufgrund noch nicht vorhandener Daten in der Datenbank, nicht umgesetzt und werden deshalb in diesem Ausblick aufgeführt.

Die vorliegende Arbeit beschreibt die Grundlagen zur Generierung eines OpenInfRA XSchemas entsprechend des UML Anwedungsschemas, sowie eine Transformation der exportieren Daten des PostgeSQL/PostGIS Datenbank in dieses XSchema. Dabei erfolgt die Umsetzung in vielen Fällen händisch oder auf der Grundlage von gegeben Skripten, die anschließen nachbearbeitet werden. Das Integrieren etweiliger Änderungen des UML Anwendungsschemas bringt deshalb einigen Aufwand mit sich. Damit mögliche Änderungen automatisiert umgesetzt werden können sind in diesem Kapitel einige Überlegungen beschrieben, welche in Anlehnung an diese Arbeit umgesetzt werden können. Weiterhin ist eine Möglichkeit dargestellt, die die Implementierung von Geometrie im XSchema vorzunehmem. Desweiteren sind mögliche Schritte erläutert, die neben dem zuvor beschrieben Export auch einen Import von Daten in eine entsprechende PostgreSQL/PostGIS Datenbank ermöglichen.

6.1 Umsetzung von Geometrie

Die PostgreSQL/PostGIS Datenbank weist eine Tabelle *Geometrie_3d_SRID_0* auf. Diese Tabelle beinhaltet Geometriedaten für Attributwerte. Der Stand der Datenbank, welcher in dieser Arbeit implementiert wird, weist jedoch zu diesem Zeitpunkt keine Geometriedaten auf, weshalb diese im XSchema keine Umsetzung finden. Die Implementierung der Geometrie im XML Schema kann beispielsweise über den Import von Elementen und komplexen Datentypen aus dem GML Namensraum erfolgen. Der *GML* (Geography Markup Language) Namensraum stellt dabei eine vielzahl von Datentypen zur Beschreibung von geografischen Objekten, z.B. geometrische Grundformen wie Punkt, Linie, Flächen und Polygon aber auch Splines, Ellipsen oder volumenhafte Geometrien, bereit. So kann eine Punktgeometrie im XSchema auf folgende Weise implementiert werden:

```
<xs:element name="Geometrie" type="gml:PointPropertyType"/>
```

Die Umsetzung in dem entsprechenden Instanzdokument ist nachfolgend dargestellt:

```
<Geometrie>
   <gml:Point gml:id="ID_5">
      <gml:pos>10.0 20.0</gml:pos>
```

```
    </gml:Point>
</Geometrie>
```

Innerhalb des `pos` Elementes werden die Koordinaten des Punktes aufgeführt.

6.2 Automatisiertes Erzeugen des Implementierungsschemas

Die Generierung des Implementierungsschemas ist vorraussetzung für die Umsetzung des XSchemas. Das unter Kapitel 3.2 beschriebene Schema wurde händisch erzeugt. Änderungen des UML Anwendungsschemas müssen dabei ebenfalls händisch in das Implementierungsschema integriert werden. Um das Generieren des Implementierungsschemas zu erleichtern, ist es hilfreich ein Skript einzusetzten, das dieses Schema automatisiert erzeugt. Das Anwenden von Skripten ist unter *Enterprise Architect* möglich. Dabei stehen folgende Skriptsprachen zur Verfügung:

- JavaScript
- JScript
- VBScript

Innerhalb des Skripts müssen dabei mehrere Schritte durchlaufen werden, damit das benötigte Implementierungsschema entsteht. Zu Beginn ist es zu empfehlen das Implementierungsschema in ein neues Package des bestehen Modells zu speichern. Aus diesem Grund muss als erster Schritt ein neues Package im Modell erstellt werden.

```
model.Packages.AddNew( "Implementierungsschema", "Class" );
```

Im zweiten Schritt ist es nötig ein neues Diagramm zu generieren, welchem anschließend Klassen und Beziehungen hinzugefügt werden.

```
package.Diagrams.AddNew( "OpenInfRA Diagram", "Class" );
```

Der dritte Schritt soll anschießend alle benötigten Elemente aus dem Ausgangsschema dem neuen Package, sowie dem Diagramm hinzufügen. Dabei sollen sowohl Klassen und Attribute, als auch Beziehungen übernommen werden. Während dieses Vorganges ist zu prüfen, ob es sich um Assoziationsklassen handelt.

```
if (element.IsAssociationClass() == true){}
```

Ist diese Bedingung erfüllt, so muss die Assoziationsklasse aufgelöst und die Beziehungen zu anderen Klassen neu generiert werden. Die Klassen können dabei über folgenden Befehl erzeugt werden:

```
package.Elements.AddNew( "Attributtyp" , "Class");
```

Die Zuordnung von Attributen zu den einzelnen Klassen erfolgt über:

```
testElement.Attributes.AddNew( "ID" , "Integer");
```

Dabei ist es wichtig Attributtypen, die nicht im XSchema umgesetzt werden können zu filtern und alternative Attributtypen zu vergeben. Anschließend müssen die Beziehungen zwischen den Klassen aus dem Anwendungsschema in das Implementierungsschema übertragen werden.

```
newElement.Connectors.AddNew("Test" , "Association");
```

Zuvor ist es nötig den Typ der vorliegenden Beziehung zu prüfen, da Generalisierungen herausgefiltert und anschließend aufgelöst werden müssen.

```
if ( connector.Type() == "Generalization" ){}
```

Während des Hinzufügens der Beziehungen ist es ebenfalls nötig Rollennamen zu vergeben. Die genannten Schritte sind Vorüberlegungen und schließen alternative Lösungswege nicht aus. Ein Ansatz zum durchführen dieses Skripts befindet sich in Anlage A.7.

6.3 Transformation von XMI nach XSD

Wie in Kapitel 3.3 bereits beschrieben, erzielt die Implementierung mittels *ShapeChange* nicht das benötigte Ergebnis und bedarf einer Nachbearbeitung des generierten XSchemas. Um direkt ein verwendbares Ergebnis erhalten zu können, ist es möglich ein Skript zu schreiben, welches anstelle vom Tool *ShapeChange* die Transformation einer XMI Datei in ein XSchema Dokument nach den geforderten Paramtetern durchführt. Zu bedenken ist dabei jedoch, dass einige Implementierungsregeln individuell für das entsprechende Element sind und nicht allgemein angewendet werden können. So zum Beispiel das Implementieren des Datentyps *Projekt*. Dieser Datentyp wird im Vergleich zu anderen Datentypen individuell verarbeitet.

6.4 Datenimport

Neben den bereits beschriebenen Datenexport, ist es ebenfalls nötig Daten aus einem XML Dokument in die PostgreSQL/PostGIS Datenbank zu importieren. Um diesen Import durchführen zu können, muss zunächst geklärt werden, auf welche Art und Weise XML Daten in eine PostgreSQL Datenbank geschrieben werden können. Dabei ist zu

erkennen, dass ein direkter Import von XML Daten in die Datenbank nicht möglich ist. Eine Möglichkeit ist der Import der kompletten XML Struktur in eine Zelle einer Tabelle und anschließendes Selektieren der Daten mittels XPath Ausdrücken. Diese Option ist jedoch umständlich. Besser ist es, aus dem XML Dokument, über eine Stylesheet Transformation, die SQL `Insert` Befehle zu generieren und diese anschließend in PostgreSQL/PostGIS abzusetzen. Voraussetzung dafür ist, dass das zu importierende XML Dokument gültig zum OpenInfRA XSchema Dokument ist. Die Transformationsdatei generiert ein `Import.sql` Dokument mittels nachfolgenden Befehls:

```
<xsl:result-document method="text" href="Import.sql">
```

Das Zieldokument wird dabei zunächst erzeugt und anschließend über die Anweisung `value-of` befüllt. Über diesen Befehl werden alle Attributwerte einer Klasse der `Insert` Anweisung hinzugefügt. Nachfolgend ist beispielsweise das Generieren eines `Insert` Befehls für die Klasse `Attributtyp` dargestellt.

```
<xsl:value-of select="concat('INSERT INTO &quot;Attributtyp&quot; VALUES (' ,$Id, ', ' ,$Name, ', ' , $Beschreibung, ', ' ,$Datentyp,', ',$Einheit,', ',$Wertebereich,');')"/>
```

Die grün hervorgehobenen Parameter sind dabei variabel und übertragen die individuellen Attributwerte der Instanzen der Klasse *Attributtyp* in die Zieldatei. Ein mögliches Ergebnis ist nachfolgend abgebildet:

```
INSERT INTO "Attributtyp" VALUES (2, 2636, NULL, 22, NULL, NULL);
```

Das komplette Transformationsdokument zum Erzeugen der `insert` Befehle der Attributtypen befindet sich in Anlage A.7. Diese Datei ist als Ansatz zum Umsetzten des Importprozesses zu sehen und deshalb exemplarisch nur für eine Klasse generiert.

7 Zusammenfassung

Das Ziel dieser Masterarbeit war das Entwickeln eines XML Schemas für das webbasierte Imformationssystem OpenInfRA, sowie das Bereitstellen einer Exportmöglichkeit der Datenbankdaten in ein Format, welches der XSchema Datei entspricht. Als Grundlage standen dabei zum einen das konzeptuelle Anwendungsschema in Form eines UML Diagramms, sowie ein Create Skript zum Generieren der Datenbank in PostgreSQL/PostGIS, bereit.

Die Generierung des XSchemas wurde zunächst mittels der Werkzeuge *Enterprise Architect* und *ShapeChange* durchgeführt. Dabei stellte sich heraus, dass diese Tools, aufgrund der Abstraktheit des UML Anwendungsschemas, das XSchema nicht auf die benötige Art und Weise generierten. Dennoch war das erzielte Ergebnis eine gute Grundlage zum Erstellen des XML Schemas. Deshalb wurden, in Anlehnung an die Norm ISO 19118, Implementierungsregeln aufgestellt, die ein Übersetzen des UML Anwendungsschema in ein XSchema ermöglichen. Desweiteren wurden Beispielinstanzen dargstellt, die verdeutlichen, welche Struktur eine mögliche XML Datei, die gültig zu diesem Schema ist, aufweist. Geometrieeigenschaften wurden in diesem XSchema nicht implementiert, da diese in der Datenbank noch nicht enthalten sind. Deshalb besteht die Notwenigkeit des Implementierens dieser Eigenschaften zu einem späteren Zeitpunkt. Abgesehen von der Geometrie wurden alle Elemente des UML Anwendungsschema im XSchema umgesetzt.

Der Export der Daten aus der PostgreSQL/PostGIS Datenbank wurde in mehreren Schritten durchgeführt, da ein direkter Export der Daten in das geforderte XSchema nicht möglich war. Dabei wurden zunächst Befehle zum speichern der Daten in eine XML Date beschrieben. Anschließend wurde diese XML Datei über eine Stylesheet Transformation in die, durch das XSchema Dokument beschriebene, XML Struktur überführt. Die Transformationsdokumente sind dabei sehr umfangreich.

Weiterhin wurde mittels einer Stylesheet Transformation eine CIDOC CRM XML generiert, die Daten eines bestimmten OpenInfRA Themas in der CIDOC CRM Struktur abbildet. Als Ausgangsdatei für diese Transformation wurde dabei das XML Exportdokument der Datenbank verwendet. Hierbei ist es denkbar, dass ein weiteres Transformationsdokument bereit gestellt werden könnte, welches es ermöglicht, aus einem Instanzdokument des OpenInfRA XSchemas ebenfalls eine CIDOC CRM XML zu generieren.

Schlussendlich wurde in dieser Arbeit ein Ausblick gegeben, der weiterführende Schritte, wie z.B. den Datenimport, darstellt.

Zusammenfassend ist zu sagen, dass sowohl das Generieren des XSchemas, sowie der Datenexport und das Erzeugen der CIDOC CRM XML mit Ausnahme der Geometrieeigenschaften, umgesetzt werden konnte.

Literaturverzeichnis

Internetquellen

Amrhein, Beatrice (2011): XSD *XML Schema Definition.*
URL: http://www.sws.bfh.ch/~amrhein/Skripten/XML/XSDSkript.pdf
(letzter Zugriff: September 2013)

Becker, Oliver (2002): *Transformationen mit XSLT.*
URL: http://www2.informatik.hu-berlin.de/~obecker/Lehre/SS2002/XML/07a-xslt.html
(letzter Zugriff: Oktober 2013)

data2type GmbH (2013): *XML-Elemente*
URL: http://www.data2type.de/xml-xslt-xslfo/xslt/xslt-referenz
(letzter Zugriff: November 2013)

Datypic, Inc.: *Schema Central.*
URL: http://www.schemacentral.com/index.html
(letzter Zugriff: Oktober 2013)

Deutscher Museumsbund (2004): *Das CIDOC Conceptual Reference Model: Eine Hilfe für den Datenaustausch?*
URL: http://www.cidoc-crm.org/docs/cidoc_paper_german.pdf
(letzter Zugriff: November 2013)

FORTH (2013): *The CIDOC Conceptual Reference Model.*
URL: http://www.cidoc-crm.org/index.html
(letzter Zugriff: Oktober 2013)

Heymann, Stefan (2003): *DTD – Document Type Definition.*
URL: http://www.stefanheymann.de/xml/dtdxml.htm
(letzter Zugriff: Oktober 2013)

interactive instruments GmbH u. The MITRE Corporation (2013): *ShapeChange Processing application schemas for geographic information.*
URL: http://shapechange.net/
(letzter Zugriff: Oktober 2013)

ISO/TC 211 (2013): *Geographic MetaData extensible markup language (GMD).*
URL: http://www.isotc211.org/schemas/2005/gmd/
(letzter Zugriff: Oktober 2013)

Jakobs, Holger (2011): *XML Schemas.*
URL: https://www.bg.bib.de/portale/xml/pdf/XML-Schema.pdf
(letzter Zugriff: September 2013)

Jeckle, Mario (2004): *OMG's XML Metadata Interchange Format XMI*.
URL: http://www.jeckle.de/files/xml4bpm_pres.pdf
(letzter Zugriff: Oktober 2013)

Kay, Michael H. (2013): *SAXON The XSLT and XQuery Processor.*
URL: http://saxon.sourceforge.net/
(letzter Zugriff: Oktober 2013)

Klippstein, Björn (2007): *XPointer.*
URL: http://4webmaster.de/wiki/XPointer
(letzter Zugriff: Oktober 2013)

Lampe, Karl-Heinz; Krause, Siegfried; Doerr, Martin (2010): *Definition des CIDOC Conceptual Reference Model.*
URL: http://www.icom-deutschland.de/client/media/380/cidoccrm_end.pdf
(letzter Zugriff: Oktober 2013)

Microsoft Corporation (2013): *UInteger-Datentyp.*
URL: http://msdn.microsoft.com/de-de/library/x79h55x9%28v=vs.90%29.aspx
(letzter Zugriff: September 2013)

O'Reilly & Associates, Inc. (2002): *XML IN A NUTSHELL Second Edition.*
URL: http://docstore.mik.ua/orelly/xml/xmlnut/index.htm
(letzter Zugriff: Oktober 2013)

Refsnes Data (2013): *W3CSchools - XLink and XPointer.*
URL: http://www.w3schools.com/xlink/xlink_intro.asp
(letzter Zugriff: Oktober 2013)

SELFHTML e.V. (2007): *Grundlagen von XSL/XSLT.*
URL: http://de.selfhtml.org/xml/darstellung/xslgrundlagen.htm
(letzter Zugriff: Oktober 2013)

SparxSystems Software GmbH (2013): *Enterprise Architect in kurzen Worten*.
URL: http://www.sparxsystems.de/uml/ea-function?L=%25Ftx_indexedsearch%25Bsword%25D%DAktivit%EF%BF%BDtendiagram
(letzter Zugriff: Oktober 2013)

Stein, Benno (2013). *XML-Schema.*
URL: http://www.uni-weimar.de/medien/webis/teaching/lecturenotes/web-technology/unit-de-doclang-xml-schema.pdf
(letzter Zugriff: September 2013)

The PostgreSQL Global Development Group (2013): *PostgreSQL 9.1.10 Documentation - XML Functions.*
URL: http://www.postgresql.org/docs/9.1/static/functions-xml.html
(letzter Zugriff: Oktober 2013)

W3C Recommendation (2004): *XML Schema Part 2: Datatypes Second Edition.*
URL: http://www.w3.org/TR/xmlschema-2/
(letzter Zugriff: Oktober 2013)

Wruck, Robert (2000): *XML Pointer Language (XPointer).*
URL: http://www.fh-wedel.de/~si/seminare/ws00/Ausarbeitung/3.xlink/xlink6.htm
(letzter Zugriff: Oktober 2013)

sonstige Quellen

Balzert, Helmut: *Java: objektorientiert programmieren: vom objektorientierten Analysemodell bis zum objektorientierten Programm.*
W3l GmbH, 2010

Norm ISO 19118 (2006). *Geographic information – Encoding*. International Organization for Standardization.

[DVD]

Norm ISO 19139 (2010). *Geoinformation - Metadaten – XML Schema Implementierung*. International Organization for Standardization.

[DVD]

OpenInfRA (23.07.2013): Grobkonzept für ein webbasiertes Informationssystem zur Dokumentation archäologischer Forschungsprojekte v.2.2

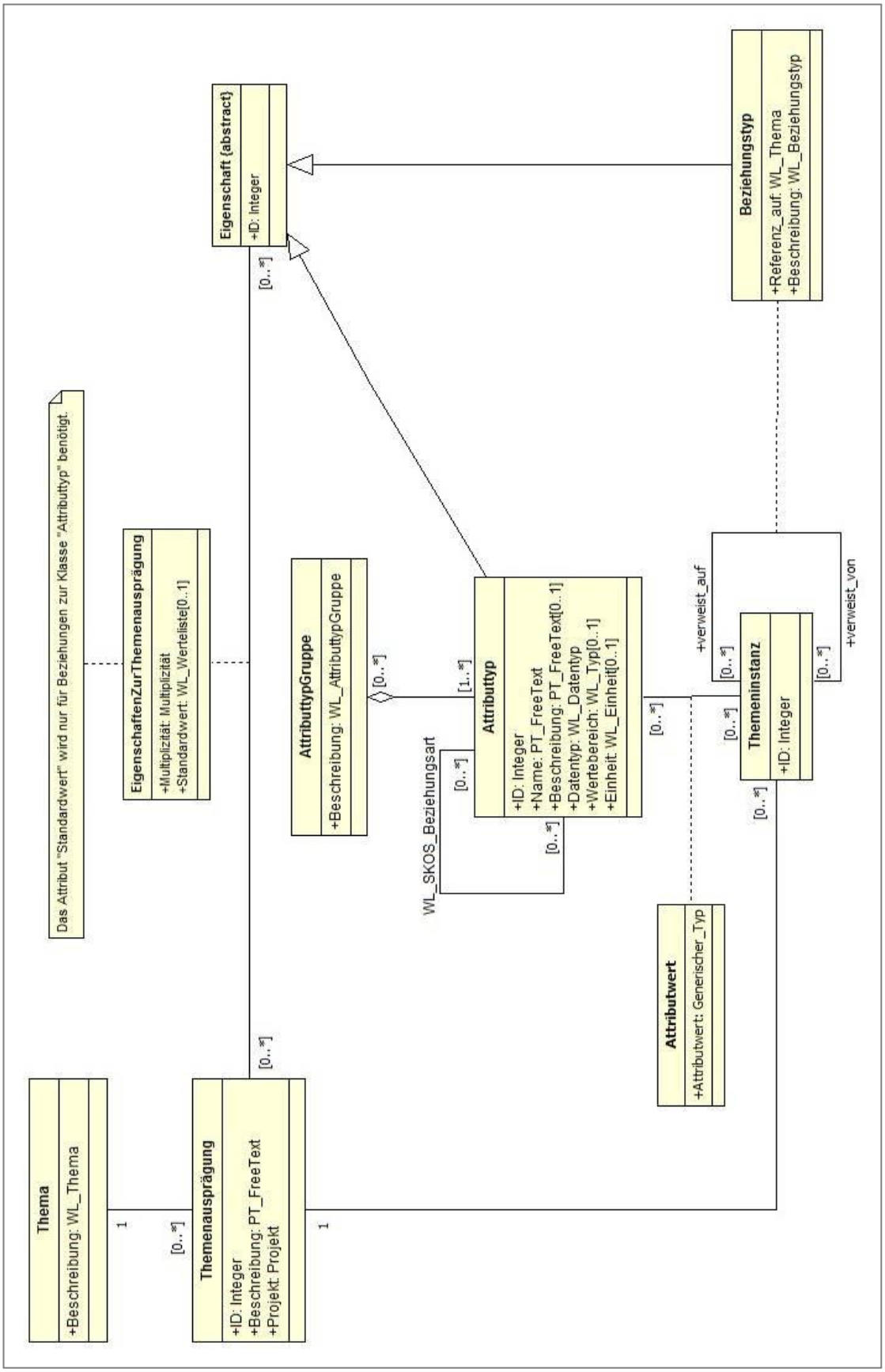
[DVD]

Abbildungsverzeichnis

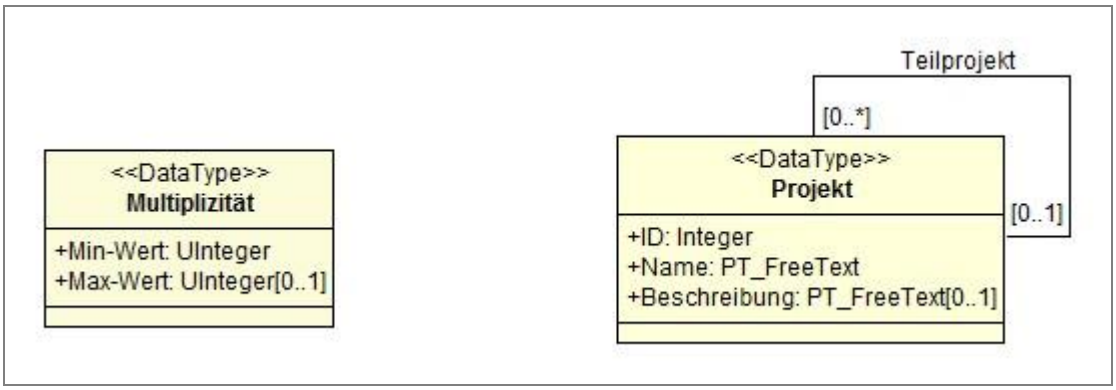
Tabellenverzeichnis

Anlagenverzeichnis

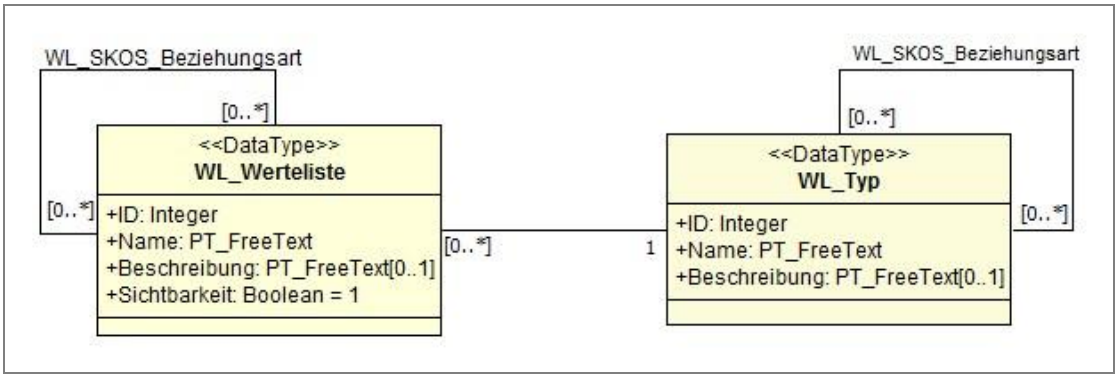
A.1 UML Anwendungsschema



Anwendungsschema Datensicht (Stand zum Bearbeitungszeitpunkt der Masterarbeit)



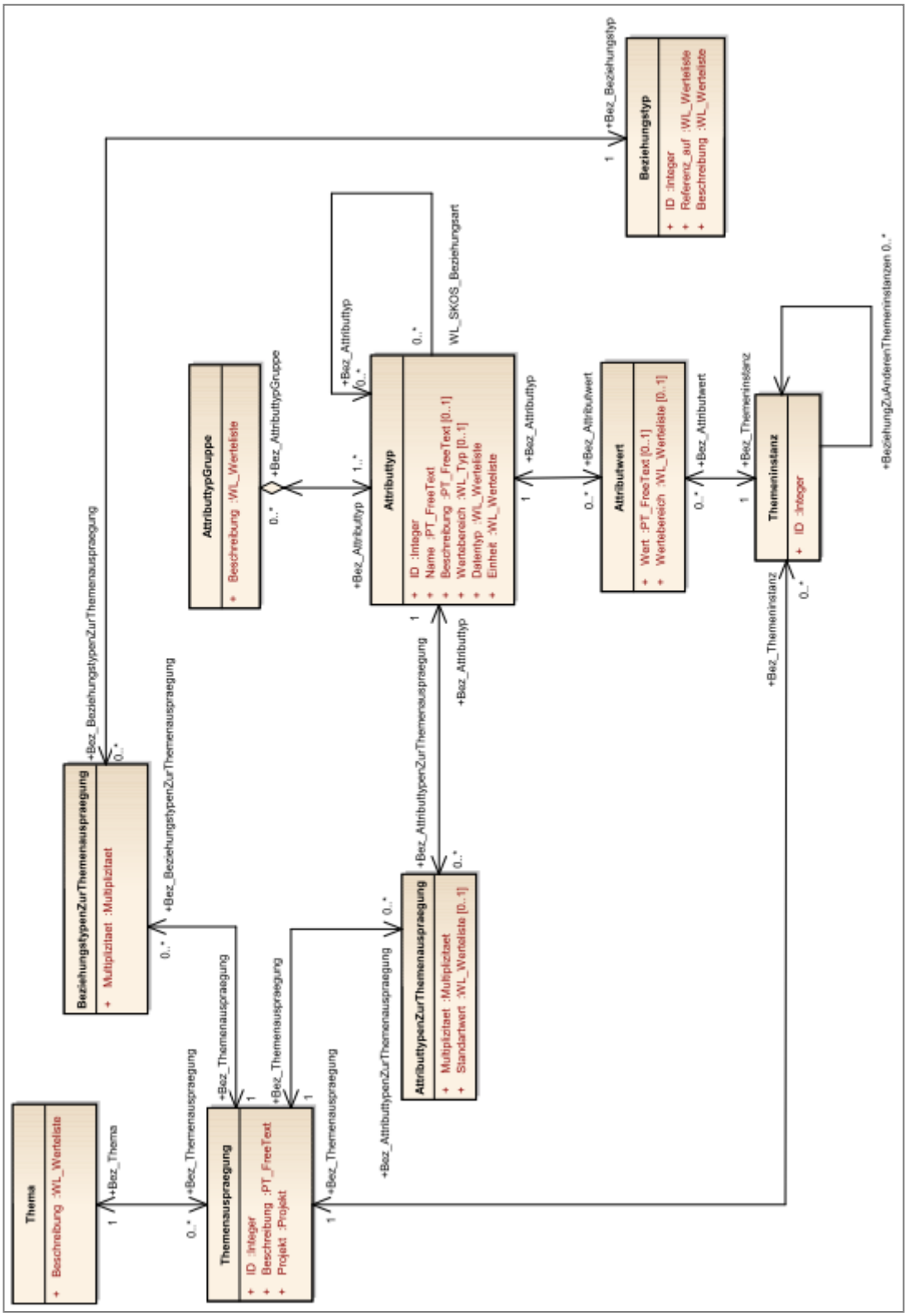
Anwendungsschema Datentypen (Multiplizität, Projekt)



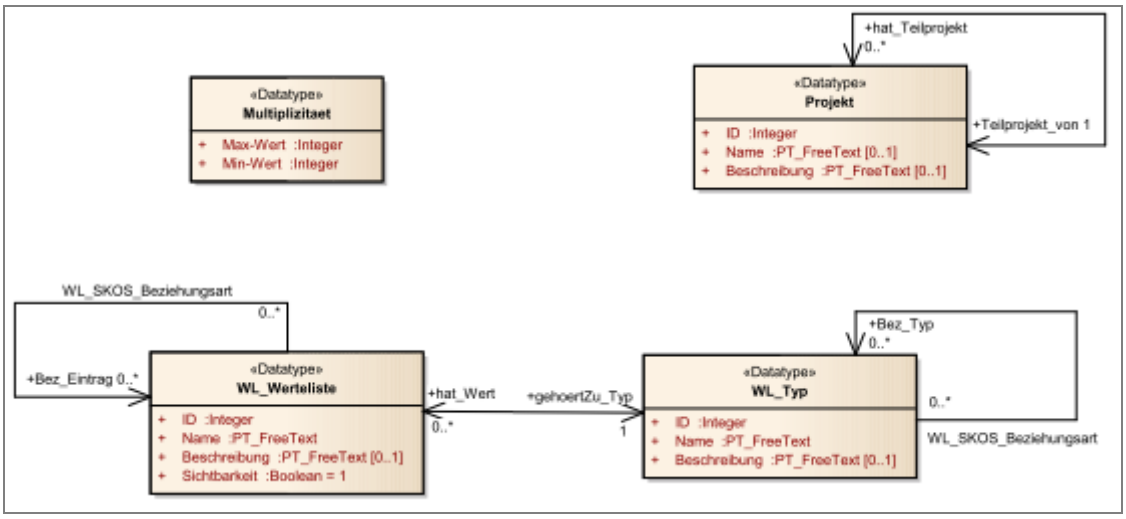
Anwendungsschema Datentypen (Werteliste, Wertelistentyp)

A.2 Implementierungsschema

Das nachfolgende Implementierungsschema wurde unter *Enterprise Architect* erstellt.



Implementierungsschema Datensicht



Implementierungsschema Datentypen

A.3 XSchema – Erstellt mittels *ShapeChange*

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:OpenInfRA="OpenInfRA"
xmlns:gco="http://www.isotc211.org/2005/gco"
xmlns:gmd="http://www.isotc211.org/2005/gmd" targetNamespace="OpenInfRA" ele-
mentFormDefault="qualified">
    <import namespace="http://www.isotc211.org/2005/gco"
    schemaLocation=http://schemas.opengis.net/iso/19139/20070417/gco/gco.xsd/>
    <import namespace="ttp://www.isotc211.org/2005/gmd"
     schemaLocation=http://schemas.opengis.net/iso/19139/20070417/gmd/gmd.xsd/>
    <!--XML Schema document created by ShapeChange - http://shapechange.net/-->
    <element name="Attributtyp" type="OpenInfRA:Attributtyp_Type"
     substitutionGroup="gco:AbstractObject"/>
    <complexType name="Attributtyp_Type">
        <complexContent>
            <extension base="gco:AbstractObject_Type">
                <sequence>
                    <element name="Bez_Attributwert"
                     type="OpenInfRA:Attributwert_PropertyType" minOccurs="0"
                     maxOccurs="unbounded"/>
                    <element name="Bez_Attributtyp"
                     type="OpenInfRA:Attributtyp_PropertyType" minOccurs="0"
                     maxOccurs="unbounded"/>
                    <element name="Bez_AttributtypGruppe"
                     type="OpenInfRA:AttributtypGruppe_PropertyType" minOccurs="0"
                     maxOccurs="unbounded"/>
                    <element name="Bez_AttributtypenZurThemenauspraegung"
                   type="OpenInfRA:AttributtypenZurThemenauspraegung_PropertyType"
                     minOccurs="0" maxOccurs="unbounded"/>
                    <element name="ID" type="gco:Integer_PropertyType"/>
                    <element name="Name" type="gmd:PT_FreeText_PropertyType"/>
                    <element name="Beschreibung"
                     type="gmd:PT_FreeText_PropertyType" minOccurs="0"/>
                    <element name="Wertebereich"
                     type="OpenInfRA:WL_Typ_PropertyType" minOccurs="0"/>
                    <element name="Datentyp"
                     type="OpenInfRA:WL_Werteliste_PropertyType"/>
                    <element name="Einheit"
                    type="OpenInfRA:WL_Werteliste_PropertyType"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <complexType name="Attributtyp_PropertyType">
        <sequence minOccurs="0">
            <element ref="OpenInfRA:Attributtyp"/>
        </sequence>
        <attributeGroup ref="gco:ObjectReference"/>
        <attribute ref="gco:nilReason"/>
    </complexType>
    <element name="AttributtypGruppe" type="OpenInfRA:AttributtypGruppe_Type"
     substitutionGroup="gco:AbstractObject"/>
    <complexType name="AttributtypGruppe_Type">
        <complexContent>
            <extension base="gco:AbstractObject_Type">
                <sequence>
                    <element name="Bez_Attributtyp"
                     type="OpenInfRA:Attributtyp_PropertyType"
                     maxOccurs="unbounded"/>
                    <element name="Beschreibung"
                     type="OpenInfRA:WL_Werteliste_PropertyType"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <complexType name="AttributtypGruppe_PropertyType">
        <sequence minOccurs="0">
```

```
            <element ref="OpenInfRA:AttributtypGruppe"/>
        </sequence>
        <attributeGroup ref="gco:ObjectReference"/>
        <attribute ref="gco:nilReason"/>
    </complexType>
    <element name="AttributtypenZurThemenauspraegung"
     type="OpenInfRA:AttributtypenZurThemenauspraegung_Type"
     substitutionGroup="gco:AbstractObject"/>
    <complexType name="AttributtypenZurThemenauspraegung_Type">
        <complexContent>
            <extension base="gco:AbstractObject_Type">
                <sequence>
                    <element name="Bez_Themenauspraegung"
                     type="OpenInfRA:Themenauspraegung_PropertyType"/>
                    <element name="Bez_Attributtyp"
                     type="OpenInfRA:Attributtyp_PropertyType"/>
                    <element name="Multiplizitaet"
                     type="OpenInfRA:Multiplizitaet_PropertyType"/>
                    <element name="Standartwert"
                     type="OpenInfRA:WL_Werteliste_PropertyType" minOccurs="0"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <complexType name="AttributtypenZurThemenauspraegung_PropertyType">
        <sequence minOccurs="0">
            <element ref="OpenInfRA:AttributtypenZurThemenauspraegung"/>
        </sequence>
        <attributeGroup ref="gco:ObjectReference"/>
        <attribute ref="gco:nilReason"/>
    </complexType>
    <element name="Attributwert" type="OpenInfRA:Attributwert_Type"
     substitutionGroup="gco:AbstractObject"/>
    <complexType name="Attributwert_Type">
        <complexContent>
            <extension base="gco:AbstractObject_Type">
                <sequence>
                    <element name="Bez_Themeninstanz"
                     type="OpenInfRA:Themeninstanz_PropertyType"/>
                    <element name="Bez_Attributtyp"
                     type="OpenInfRA:Attributtyp_PropertyType"/>
                    <element name="Wert" type="gmd:PT_FreeText_PropertyType"
                     minOccurs="0"/>
                    <element name="Wertebereich"
                     type="OpenInfRA:WL_Werteliste_PropertyType" minOccurs="0"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <complexType name="Attributwert_PropertyType">
        <sequence minOccurs="0">
            <element ref="OpenInfRA:Attributwert"/>
        </sequence>
        <attributeGroup ref="gco:ObjectReference"/>
        <attribute ref="gco:nilReason"/>
    </complexType>
    <element name="Beziehungstyp" type="OpenInfRA:Beziehungstyp_Type"
     substitutionGroup="gco:AbstractObject"/>
    <complexType name="Beziehungstyp_Type">
        <complexContent>
            <extension base="gco:AbstractObject_Type">
                <sequence>
                    <element name="Bez_BeziehungstypenZurThemenauspraegung"
                    type="OpenInfRA:BeziehungstypenZurThemenauspraegung_PropertyType"
                     minOccurs="0" maxOccurs="unbounded"/>
                    <element name="ID" type="gco:Integer_PropertyType"/>
                    <element name="Referenz_auf"
                     type="OpenInfRA:WL_Werteliste_PropertyType"/>
                    <element name="Beschreibung"
                     type="OpenInfRA:WL_Werteliste_PropertyType"/>
```

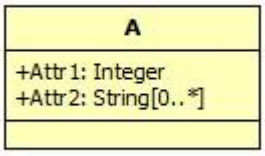
```
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="Beziehungstyp_PropertyType">
      <sequence minOccurs="0">
         <element ref="OpenInfRA:Beziehungstyp"/>
      </sequence>
      <attributeGroup ref="gco:ObjectReference"/>
      <attribute ref="gco:nilReason"/>
   </complexType>
   <element name="BeziehungstypenZurThemenauspraegung"
    type="OpenInfRA:BeziehungstypenZurThemenauspraegung_Type"
    substitutionGroup="gco:AbstractObject"/>
   <complexType name="BeziehungstypenZurThemenauspraegung_Type">
      <complexContent>
         <extension base="gco:AbstractObject_Type">
            <sequence>
               <element name="Bez_Themenauspraegung"
                type="OpenInfRA:Themenauspraegung_PropertyType"/>
               <element name="Bez_Beziehungstyp"
                type="OpenInfRA:Beziehungstyp_PropertyType"/>
               <element name="Multiplizitaet"
                type="OpenInfRA:Multiplizitaet_PropertyType"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="BeziehungstypenZurThemenauspraegung_PropertyType">
      <sequence minOccurs="0">
         <element ref="OpenInfRA:BeziehungstypenZurThemenauspraegung"/>
      </sequence>
      <attributeGroup ref="gco:ObjectReference"/>
      <attribute ref="gco:nilReason"/>
   </complexType>
   <element name="Multiplizitaet" type="OpenInfRA:Multiplizitaet_Type"
substitutionGroup="gco:AbstractObject"/>
   <complexType name="Multiplizitaet_Type">
      <complexContent>
         <extension base="gco:AbstractObject_Type">
            <sequence>
               <element name="Max-Wert" type="gco:Integer_PropertyType"/>
               <element name="Min-Wert" type="gco:Integer_PropertyType"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="Multiplizitaet_PropertyType">
      <sequence minOccurs="0">
         <element ref="OpenInfRA:Multiplizitaet"/>
      </sequence>
      <attributeGroup ref="gco:ObjectReference"/>
      <attribute ref="gco:nilReason"/>
   </complexType>
   <element name="Projekt" type="OpenInfRA:Projekt_Type"
substitutionGroup="gco:AbstractObject"/>
   <complexType name="Projekt_Type">
      <complexContent>
         <extension base="gco:AbstractObject_Type">
            <sequence>
               <element name="Teilprojekt_von"
                type="OpenInfRA:Projekt_PropertyType"/>
               <element name="hat_Teilprojekt"
                type="OpenInfRA:Projekt_PropertyType" minOccurs="0"
                maxOccurs="unbounded"/>
               <element name="ID" type="gco:Integer_PropertyType"/>
               <element name="Name" type="gmd:PT_FreeText_PropertyType"
                minOccurs="0"/>
               <element name="Beschreibung"
                type="gmd:PT_FreeText_PropertyType" minOccurs="0"/>
```

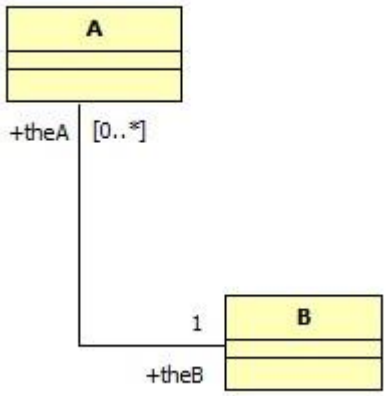
```
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="Projekt_PropertyType">
      <sequence minOccurs="0">
         <element ref="OpenInfRA:Projekt"/>
      </sequence>
      <attributeGroup ref="gco:ObjectReference"/>
      <attribute ref="gco:nilReason"/>
   </complexType>
   <element name="Thema" type="OpenInfRA:Thema_Type"
    substitutionGroup="gco:AbstractObject"/>
   <complexType name="Thema_Type">
      <complexContent>
         <extension base="gco:AbstractObject_Type">
            <sequence>
               <element name="Bez_Themenauspraegung"
                type="OpenInfRA:Themenauspraegung_PropertyType" minOccurs="0"
                maxOccurs="unbounded"/>
               <element name="Beschreibung"
                type="OpenInfRA:WL_Werteliste_PropertyType"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="Thema_PropertyType">
      <sequence minOccurs="0">
         <element ref="OpenInfRA:Thema"/>
      </sequence>
      <attributeGroup ref="gco:ObjectReference"/>
      <attribute ref="gco:nilReason"/>
   </complexType>
   <element name="Themenauspraegung" type="OpenInfRA:Themenauspraegung_Type"
substitutionGroup="gco:AbstractObject"/>
   <complexType name="Themenauspraegung_Type">
      <complexContent>
         <extension base="gco:AbstractObject_Type">
            <sequence>
               <element name="Bez_BeziehungstypenZurThemenauspraegung"
               type="OpenInfRA:BeziehungstypenZurThemenauspraegung_PropertyType"
                minOccurs="0" maxOccurs="unbounded"/>
               <element name="Bez_AttributtypenZurThemenauspraegung"
              type="OpenInfRA:AttributtypenZurThemenauspraegung_PropertyType"
                minOccurs="0" maxOccurs="unbounded"/>
               <element name="Bez_Themeninstanz"
                type="OpenInfRA:Themeninstanz_PropertyType" minOccurs="0"
                maxOccurs="unbounded"/>
               <element name="Bez_Thema"
                type="OpenInfRA:Thema_PropertyType"/>
               <element name="ID" type="gco:Integer_PropertyType"/>
               <element name="Beschreibung"
                type="gmd:PT_FreeText_PropertyType"/>
               <element name="Projekt"
                type="OpenInfRA:Projekt_PropertyType"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <complexType name="Themenauspraegung_PropertyType">
      <sequence minOccurs="0">
         <element ref="OpenInfRA:Themenauspraegung"/>
      </sequence>
      <attributeGroup ref="gco:ObjectReference"/>
      <attribute ref="gco:nilReason"/>
   </complexType>
   <element name="Themeninstanz" type="OpenInfRA:Themeninstanz_Type"
    substitutionGroup="gco:AbstractObject"/>
   <complexType name="Themeninstanz_Type">
      <complexContent>
```

```
        <extension base="gco:AbstractObject_Type">
            <sequence>
                <element name="Bez_Attributwert"
                 type="OpenInfRA:Attributwert_PropertyType" minOccurs="0"
                 maxOccurs="unbounded"/>
                <element name="Bez_Themenauspraegung"
                 type="OpenInfRA:Themenauspraegung_PropertyType"/>
                <element name="BeziehungZuAnderenThemeninstanzen"
                 type="OpenInfRA:Themeninstanz_PropertyType" minOccurs="0"
                 maxOccurs="unbounded"/>
                <element name="ID" type="gco:Integer_PropertyType"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<complexType name="Themeninstanz_PropertyType">
    <sequence minOccurs="0">
        <element ref="OpenInfRA:Themeninstanz"/>
    </sequence>
    <attributeGroup ref="gco:ObjectReference"/>
    <attribute ref="gco:nilReason"/>
</complexType>
<element name="WL_Typ" type="OpenInfRA:WL_Typ_Type"
 substitutionGroup="gco:AbstractObject"/>
<complexType name="WL_Typ_Type">
    <complexContent>
        <extension base="gco:AbstractObject_Type">
            <sequence>
                <element name="hat_Wert"
                 type="OpenInfRA:WL_Werteliste_PropertyType" minOccurs="0"
                 maxOccurs="unbounded"/>
                <element name="Bez_Typ" type="OpenInfRA:WL_Typ_PropertyType"
                 minOccurs="0" maxOccurs="unbounded"/>
                <element name="ID" type="gco:Integer_PropertyType"/>
                <element name="Name" type="gmd:PT_FreeText_PropertyType"/>
                <element name="Beschreibung"
                 type="gmd:PT_FreeText_PropertyType" minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<complexType name="WL_Typ_PropertyType">
    <sequence minOccurs="0">
        <element ref="OpenInfRA:WL_Typ"/>
    </sequence>
    <attributeGroup ref="gco:ObjectReference"/>
    <attribute ref="gco:nilReason"/>
</complexType>
<element name="WL_Werteliste" type="OpenInfRA:WL_Werteliste_Type"
 substitutionGroup="gco:AbstractObject"/>
<complexType name="WL_Werteliste_Type">
    <complexContent>
        <extension base="gco:AbstractObject_Type">
            <sequence>
                <element name="Bez_Eintrag"
                 type="OpenInfRA:WL_Werteliste_PropertyType" minOccurs="0"
                 maxOccurs="unbounded"/>
                <element name="gehoertZu_Typ"
                 type="OpenInfRA:WL_Typ_PropertyType"/>
                <element name="ID" type="gco:Integer_PropertyType"/>
                <element name="Name" type="gmd:PT_FreeText_PropertyType"/>
                <element name="Beschreibung"
                 type="gmd:PT_FreeText_PropertyType" minOccurs="0"/>
                <element name="Sichtbarkeit" type="gco:Boolean_PropertyType"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<complexType name="WL_Werteliste_PropertyType">
    <sequence minOccurs="0">
```

```
            <element ref="OpenInfRA:WL_Werteliste"/>
        </sequence>
        <attributeGroup ref="gco:ObjectReference"/>
        <attribute ref="gco:nilReason"/>
    </complexType>
</schema>
```

A.4 Implementierungsregeln

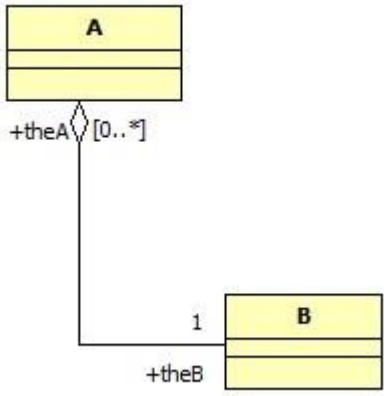
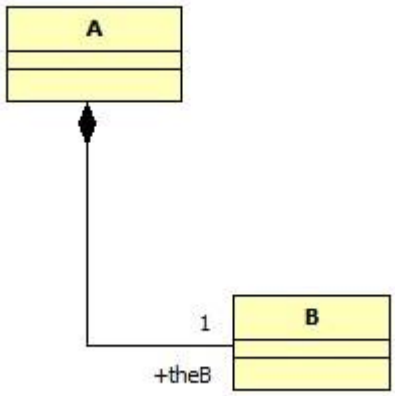
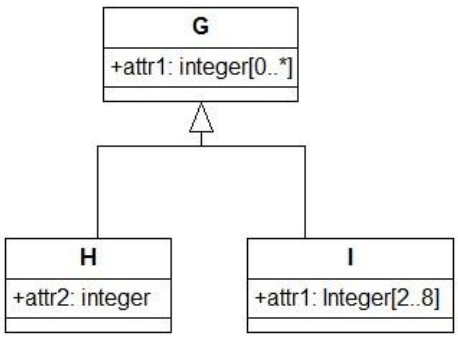
<table>
<tr><th></th><th>UML</th><th>Implementierung XSchema</th></tr>
<tr><td colspan="3">1. Klassen und Attribute</td></tr>
<tr><td>1</td><td> 
</td><td><pre>
<xs:element name="A"
 type="KlassennameType"/>

<xs:complexType name="A_Type">
  <xs:sequence>
    <xs:element name="Attr1"
     type="xs:Integer"/>
    <xs:element name="Attr2"
     type="xs:String" minOccurs="0"
     maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id"
   type="xs:ID" use="required"/>
</xs:complexType>
</pre></td></tr>
<tr><td colspan="3">2. Beziehungen</td></tr>
<tr><td colspan="3">2.1. Assoziation</td></tr>
<tr><td>2</td><td> 
</td><td><pre>
<xs:complexType name="A_Type">
  <xs:sequence>
    <xs:element name="theB"
     type="ref_B"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ref_B">
  <xs:attribute name="idref"
   type="xs:IDREF" use="required"/>
</xs:complexType>

<xs:complexType name="B_Type">
  <xs:sequence>
    <xs:element name="theA"
     type="ref_A" minOccurs="0"
    maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

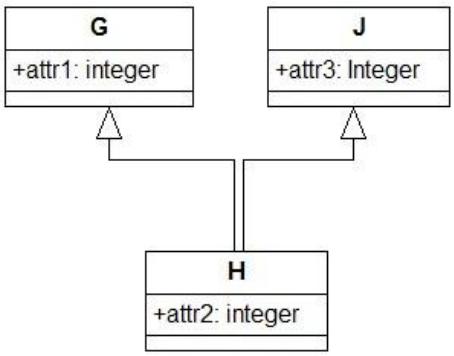
<xs:complexType name="ref_A">
  <xs:attribute name="idref"
   type="xs:IDREF" use="required"/>
</xs:complexType>
</pre></td></tr>
</table>

<table>
<tr><td colspan="3">2.2. Aggregation</td></tr>
<tr><td>3</td><td> 
</td><td>Implementierung wie unter Zeile 2</td></tr>
<tr><td colspan="3">2.3. Komposition</td></tr>
<tr><td>4</td><td> 
</td><td><pre>
<xs:complexType name="E">
  <xs:sequence>
    <xs:element name="theE" type="F"
     minOccurs="2" maxOccurs="8"/>
  </xs:sequence>
</xs:complexType>
</pre></td></tr>
<tr><td colspan="3">2.4. Generalisierung (Implementierung der erbenden Klasse)</td></tr>
<tr><td>5</td><td>Einfache Vererbung:
 
</td><td><pre>
<xs:complexType name="H">
  <xs:complexContent>
    <xs:extension base="G">
      <xs:sequence>
        <xs:element name="attr2"
         type="integer"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="I">
  <xs:complexContent>
    <xs:restriction base="G">
      <xs:sequence>
        <xs:element name="attr1"
         type="Integer" minOccurs="2"
</pre></td></tr>
</table>

<table>
<tr><td></td><td></td><td>

```
        maxOccurs="8"/>
      </xs:sequence>
    </xs:restricton>
  </xs:complexContent>
</xs:complexType>
```

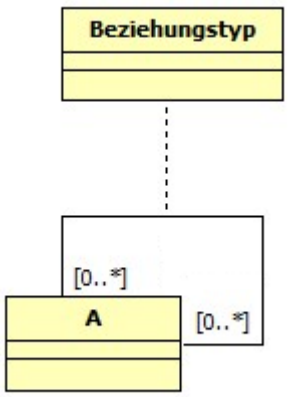
</td></tr>
<tr><td>6</td><td>Mehrfache Vererbung:
 
</td><td>

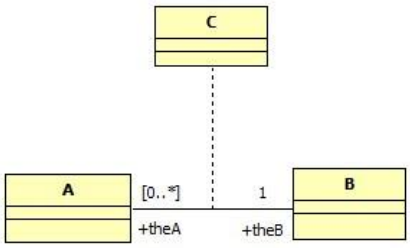
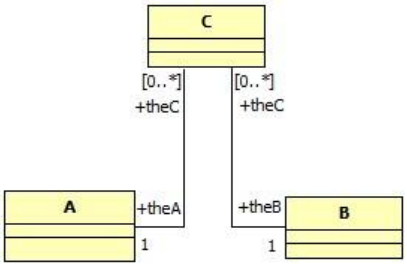
```
<xs:complexType name="H">
  <xs:complexContent>
      <xs:sequence>
        <xs:element name="attr1"
         type="Integer"/>
        <xs:element name="attr2"
         type="Integer"/>
        <xs:element name="attr3"
         type="Integer"/>
      </xs:sequence>
  </xs:complexContent>
</xs:complexType>
```

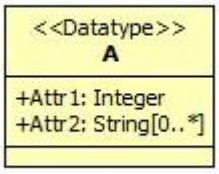
</td></tr>
<tr><td colspan="3">2.5. SKOS Beziehungsart</td></tr>
<tr><td>7</td><td> 
</td><td>

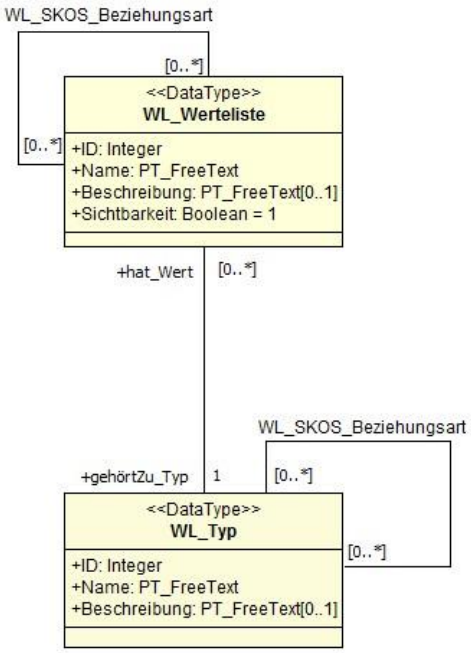
```
<xs:element name="Beziehung_A"
type="SKOSReferenzType" minOccurs="0"
maxOccurs="unbounded"/>

<xs:complexType
 name="SKOSReferenzType">
  <xs:attribute name="idref"
   type="xs:IDREF" use="required"/>
  <xs:attribute
   name="SKOS_Beziehungsart"
   type="xs:anyURI" use="required"/>
</xs:complexType>
```

</td></tr>
<tr><td colspan="3">2.6. Beziehungen zu anderen Instanzen</td></tr>
</table>

8		`<xs:element name="BeziehungZuAnderenInstanzen" type="BeziehungZuAnderenInstanzen Type" minOccurs="0"maxOccurs="unbounded"/>`
		
		`<xs:complexType name="BeziehungZu AnderenInstanzenType">`
		`<xs:attribute name="idref" type="xs:IDREF" use="required"/>`
		`<xs:attribute name="Beziehungstyp" type="xs:IDREF" use="required"/>`
		`</xs:complexType>`

3. Assoziationsklassen		
9		keine Implementierungsmöglichkeit
	Assoziationsklasse auflösen:	
10		Implementierung wie unter Zeile 2
4. Datentypen		
4.1. build-in Datentypen des XSchema Namensraums		
11	Siehe Zeile 1	`<xs:element name="Attr1" type="xs:integer"/>`

<table>
<tr><td colspan="3">4.1. Klassen mit Stereotyp <<Datatype>></td></tr>
<tr><td>12</td><td> 
</td><td>
<pre>
…
<xs:element name="Multiplizitaet"
type="MultiplizitaetType"/>
…

<xs:complexType
 name="MultiplizitaetType">
   <xs:sequence>
      <xs:element name="Min-Wert"
       type="xs:unsignedInt"/>
      <xs:element name="Max-Wert"
       type="xs:unsignedInt"
       minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
</pre>
</td></tr>
<tr><td colspan="3">4.3. Wertelisten</td></tr>
<tr><td>13</td><td> 
</td><td>
<pre>
<xs:element name="Wertelisten">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="WL_Typ">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Typ"
              type="WL_TypType"
              maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element
       name="WL_Werteliste">
        <xs:complexType>
          <xs:sequence>
            <xs:element
             name="Eintrag"
             type="WL_WertelisteType"
             maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</pre>

Umsetzung <code>WL_TypType</code> und <code>WL_Werteliste</code> wie unter Zeile 12.
Umsetzung der SKOS Beziehungsart wie unter Zeile 7.

</td></tr>
</table>

Aufruf des Wertelisteneintrags:

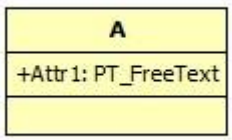
```
<xs:element name="Beschreibung"
 type="WL_ReferenzType"/>

<xs:complexType
 name="WL_ReferenzType">
   <xs:attribute
    name="WertelistenEintrag"
    type="xs:anyURI" use="required"/>
</xs:complexType>
```

Aufruf des Wertelistentyps:

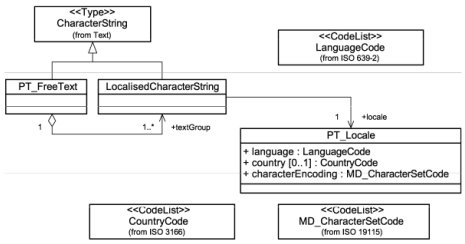
```
<xs:element name="Wertebereich"
 type="WL_TypReferenzType"
 minOccurs="0"/>

<xs:complexType
 name="WL_TypReferenzType">
  <xs:attribute name="WertelistenTyp"
   type="xs:anyURI" use="required"/>
</xs:complexType>
```

4.4 PT_Freetext

14



Implementierung im GMD Namensraum nach ISO 19139:




```
<xs:element name="Attr1"
type="gmd:PT_FreeText_PropertyType"/>
```

Implementierung im GMD Namensraum nach ISO 19139:

```
<xs:complexType
 name="PT_FreeText_PropertyType">
   <xs:complexContent>
       <xs:extension
        base="gco:CharacterString_
              PropertyType">
          <xs:sequence minOccurs="0">
            <xs:element
             ref="gmd:PT_FreeText"/>
          </xs:sequence>
       </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="PT_FreeText"
type="gmd:PT_FreeText_Type"/>

<xs:complexType
```

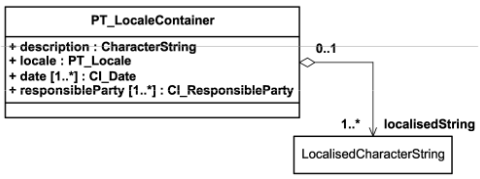
```
 name="PT_FreeText_Type">
  <xs:complexContent>
    <xs:extension
     base="gco:AbstractObject_Type">
       <xs:sequence>
         <xs:element name="textGroup"
          type="gmd:Localised
                CharacterString_
                PropertyType"
          maxOccurs="unbounded"/>
       </xs:sequence>
     </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType
 name="LocalisedCharacterString_
       Type">
  <xs:simpleContent>
    <xs:extension
       base="xs:string">
      <xs:attribute name="id"
        type="xs:ID"/>
      <xs:attribute name="locale"
        type="xs:anyURI"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:element
name="LocalisedCharacterString"
type="gmd:LocalisedCharacterString_
Type" substitutionGroup="gco:
CharacterString"/>

<xs:complexType
   name="LocalisedCharacterString_
         PropertyType">
   <xs:complexContent>
     <xs:extension
        base="gco:ObjectReference_
              PropertyType">
       <xs:sequence minOccurs="0">
          <xs:element
           ref="gmd:Localised
                CharacterString"/>
       </xs:sequence>
     </xs:extension>
   </xs:complexContent>
</xs:complexType>

<xs:complexType
name="PT_Locale_Type">
 <xs:complexContent>
   <xs:extension base="gco:Abstract
                       Object_Type">
     <xs:sequence>
      <xs:element name="languageCode"
        type="gmd:LanguageCode_
              PropertyType"/>
```

<table>
<tr>
<td></td>
<td></td>
<td>

```
      <xs:element name="country"
       type="gmd:Country_
               PropertyType"
       minOccurs="0"/>
      <xs:element
       name="characterEncoding"
       type="gmd:MD_CharacterSet
             Code_PropertyType"/>
    </xs:sequence>
   </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="PT_Locale"
 type="gmd:PT_Locale_Type"/>

<xs:complexType
 name="PT_Locale_PropertyType">
  <xs:sequence minOccurs="0">
    <xs:element ref="gmd:PT_Locale"/>
  </xs:sequence>
  <xs:attributeGroup
   ref="gco:ObjectReference"/>
  <xs:attribute ref="gco:nilReason"/>
</xs:complexType>
```

</td>
</tr>
<tr>
<td>15</td>
<td>

Implementierung von Übersetzungs-containern im GD Namensraum nach ISO 19139:



</td>
<td>

```
<xs:complexType
 name="PT_LocaleContainer_Type">
   <xs:sequence>
      <xs:element name="description"
       type="gco:CharacterString_
             PropertyType"/>
     <xs:element name="locale"
      type="gmd:PT_Locale_Property
            Type"/>
     <xs:element name="date"
       type="gmd:CI_Date_Property
             Type"
       maxOccurs="unbounded"/>
     <xs:element
       name="responsibleParty"
       type="gmd:CI_Responsible
             Party_PropertyType"
       maxOccurs="unbounded"/>
     <xs:element
       name="localisedString"
       type="gmd:LocalisedCharacter
             String_PropertyType"
       maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>

<xs:element name="PT_LocaleContainer"
 type="gmd:PT_LocaleContainer_Type"/>

<xs:complexType
name="PT_LocaleContainer_Property
       Type">
  <xs:sequence minOccurs="0">
```

</td>
</tr>
</table>

		<xs:element ref="gmd:PT_LocaleContainer"/> </xs:sequence> <xs:attributeGroup ref="gco:ObjectReference"/> <xs:attribute ref="gco:nilReason"/> </xs:complexType>
5. Wurzelelement		
16	nicht im UML Anwendungsschema dargestellt	<xs:element name="Datensatz"> <xs:complexType> <xs:sequence> <xs:element name="A" type="A_Type" maxOccurs="unbounded"/> … </xs:sequence> </xs:complexType> </xs:element>

A.5 XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="OpenInfRA" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:gmd="http://www.isotc211.org/2005/gmd"
xmlns:gco="http://www.isotc211.org/2005/gco" targetNamespace="OpenInfRA" ver-
sion="1.0">
    <xs:import namespace="http://www.isotc211.org/2005/gmd"
    schemaLocation=http://schemas.opengis.net/iso/19139/20070417/gmd/gmd.xsd/>
    <xs:import namespace="http://www.isotc211.org/2005/gco"
     schemaLocation=http://schemas.opengis.net/iso/19139/20070417/gco/gco.xsd/>

    <!--Wurzelelemente-->
    <xs:element name="Dataset">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Attributtyp" type="AttributtypType"
                 maxOccurs="unbounded"/>
                <xs:element name="AttributtypGruppe" type="AttributtypGruppeType"
                 maxOccurs="unbounded"/>
                <xs:element name="AttributtypenZurThemenauspräqung"
                 type="AttributtypenZurThemenauspräqungType"
                 maxOccurs="unbounded"/>
                <xs:element name="Attributwert" type="AttributwertType"
                 maxOccurs="unbounded"/>
                <xs:element name="Beziehungstyp" type="BeziehungstypType"
                 maxOccurs="unbounded"/>
                <xs:element name="BeziehungstypenZurThemenauspräqung"
                 type="BeziehungstypenZurThemenauspräqungType"
                 maxOccurs="unbounded"/>
                <xs:element name="Thema" type="ThemaType" maxOccurs="unbounded"/>
                <xs:element name="Themenauspräqung" type="ThemenauspräqungType"
                 maxOccurs="unbounded"/>
                <xs:element name="Themeninstanz" type="ThemeninstanzType"
                 maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="Wertelisten">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="WL_Typ">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Typ" type="WL_TypType"
                             maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="WL_Werteliste">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Eintrag" type="WL_WertelisteType"
                             maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

```
<!—Gruppierung Attributwert-->
<xs:group name="Attributwert">
    <xs:choice>
        <xs:element name="Wert" type="gmd:PT_FreeText_PropertyType"/>
        <xs:element name="Wertebereich" type="WL_ReferenzType"/>
    </xs:choice>
</xs:group>

<!--Datentypen-->
<xs:complexType name="AttributtypType">
    <xs:sequence>
        <xs:element name="ID" type="xs:integer"/>
        <xs:element name="Name" type="gmd:PT_FreeText_PropertyType"/>
        <xs:element name="Beschreibung" type="gmd:PT_FreeText_PropertyType"
         minOccurs="0"/>
        <xs:element name="Datentyp" type="WL_ReferenzType"/>
        <xs:element name="Wertebereich" type="WL_TypReferenzType"
         minOccurs="0"/>
        <xs:element name="Einheit" type="WL_ReferenzType" minOccurs="0"/>
        <xs:element name="Bez_AttributtypGruppe" type="ReferenzType"
         minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Bez_Attributwert" type="ReferenzType"
         minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Bez_AttributtypenZurThemenausprägung"
         type="ReferenzType" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Bez_Attributtyp" type="SKOSReferenzType"
         minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>
<xs:complexType name="AttributtypGruppeType">
    <xs:sequence>
        <xs:element name="Beschreibung" type="WL_ReferenzType"/>
        <xs:element name="Bez_Attributtyp" type="ReferenzType"
         maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>
<xs:complexType name="AttributtypenZurThemenausprägungType">
    <xs:sequence>
        <xs:element name="Multiplizität" type="MultiplizitätType"/>
        <xs:element name="Standardwert" type="WL_ReferenzType"
         minOccurs="0"/>
        <xs:element name="Bez_Attributtyp" type="ReferenzType"/>
        <xs:element name="Bez_Themenausprägung" type="ReferenzType"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>
<xs:complexType name="AttributwertType">
    <xs:sequence>
        <xs:group ref="Attributwert"/>
        <xs:element name="Bez_Attributtyp" type="ReferenzType"/>
        <xs:element name="Bez_Themeninstanz" type="ReferenzType"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>
<xs:complexType name="BeziehungstypType">
    <xs:sequence>
        <xs:element name="ID" type="xs:integer"/>
        <xs:element name="Referenz_auf" type="WL_ReferenzType"/>
        <xs:element name="Beschreibung" type="WL_ReferenzType"/>
        <xs:element name="Bez_BeziehungsttypenZurThemenausprägung"
```

```
            type="ReferenzType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required"/>
    </xs:complexType>
    <xs:complexType name="BeziehungstypenZurThemenausprägungType">
        <xs:sequence>
            <xs:element name="Multiplizität" type="MultiplizitätType"/>
            <xs:element name="Bez_Beziehungstyp" type="ReferenzType"/>
            <xs:element name="Bez_Themenausprägung" type="ReferenzType"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required"/>
    </xs:complexType>
    <xs:complexType name="MultiplizitätType">
        <xs:sequence>
            <xs:element name="Min-Wert" type="xs:unsignedInt"/>
            <xs:element name="Max-Wert" type="xs:unsignedInt" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ProjektType">
        <xs:sequence>
            <xs:element name="ID" type="xs:integer"/>
            <xs:element name="Name" type="gmd:PT_FreeText_PropertyType"/>
            <xs:element name="Beschreibung" type="gmd:PT_FreeText_PropertyType"
             minOccurs="0"/>
            <xs:element name="Teilprojekt_von" type="ProjektType"
             minOccurs="0"/>
            <xs:element name="hat_Teilprojekt" type="ProjektType" minOccurs="0"
             maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ThemaType">
        <xs:sequence>
            <xs:element name="Beschreibung" type="WL_ReferenzType"/>
            <xs:element name="Bez_Themenausprägung" type="ReferenzType"
             minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required"/>
    </xs:complexType>
    <xs:complexType name="ThemenausprägungType">
        <xs:sequence>
            <xs:element name="ID" type="xs:integer"/>
            <xs:element name="Beschreibung"
             type="gmd:PT_FreeText_PropertyType"/>
            <xs:element name="Projekt" type="ProjektType"/>
            <xs:element name="Bez_Themeninstanz" type="ReferenzType"
             minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Bez_AttributtypenZurThemenausprägung"
             type="ReferenzType" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Bez_BeziehungstypenZurThemenausprägung"
             type="ReferenzType" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Bez_Thema" type="ReferenzType"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required"/>
    </xs:complexType>
    <xs:complexType name="ThemeninstanzType">
        <xs:sequence>
            <xs:element name="ID" type="xs:integer"/>
            <xs:element name="BeziehungZuAnderenThemeninstanzen"
             type="BeziehungZuAnderenThemeninstanzenType" minOccurs="0"
             maxOccurs="unbounded"/>
            <xs:element name="Bez_Attributwert" type="ReferenzType"
             minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Bez_Themenausprägung" type="ReferenzType"/>
```

```
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required"/>
    </xs:complexType>
    <xs:complexType name="WL_TypType">
        <xs:sequence>
            <xs:element name="ID" type="xs:integer"/>
            <xs:element name="Name" type="gmd:PT_FreeText_PropertyType"/>
            <xs:element name="Beschreibung" type="gmd:PT_FreeText_PropertyType"
             minOccurs="0"/>
            <xs:element name="hat_Wert" type="ReferenzType" minOccurs="0"
             maxOccurs="unbounded"/>
            <xs:element name="Bez_Typ" type="WL_SKOSReferenzType" minOccurs="0"
             maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required"/>
    </xs:complexType>
    <xs:complexType name="WL_WertelisteType">
        <xs:sequence>
            <xs:element name="ID" type="xs:integer"/>
            <xs:element name="Name" type="gmd:PT_FreeText_PropertyType"/>
            <xs:element name="Beschreibung" type="gmd:PT_FreeText_PropertyType"
             minOccurs="0"/>
            <xs:element name="Sichtbarkeit" type="xs:boolean" default="1"/>
            <xs:element name="gehörtZu_Typ" type="ReferenzType"/>
            <xs:element name="Bez_Eintrag" type="WL_SKOSReferenzType"
             minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required"/>
    </xs:complexType>
    <!--Referenzen-->
    <xs:complexType name="ReferenzType">
        <xs:attribute name="idref" type="xs:IDREF" use="required"/>
    </xs:complexType>
    <xs:complexType name="WL_ReferenzType">
        <xs:attribute name="WertelistenEintrag" type="xs:anyURI"
         use="required"/>
    </xs:complexType>
    <xs:complexType name="WL_TypReferenzType">
        <xs:attribute name="WertelistenTyp" type="xs:anyURI" use="required"/>
    </xs:complexType>
    <xs:complexType name="WL_SKOSReferenzType">
        <xs:attribute name="idref" type="xs:IDREF" use="required"/>
        <xs:attribute name="SKOS_Beziehungsart" type="xs:IDREF" use="required"/>
    </xs:complexType>
    <xs:complexType name="SKOSReferenzType">
        <xs:attribute name="idref" type="xs:IDREF" use="required"/>
        <xs:attribute name="SKOS_Beziehungsart" type="xs:anyURI"
         use="required"/>
    </xs:complexType>
    <xs:complexType name="BeziehungZuAnderenThemeninstanzenType">
        <xs:attribute name="idref" type="xs:IDREF" use="required"/>
        <xs:attribute name="Beziehungstyp" type="xs:IDREF" use="required"/>
    </xs:complexType>
</xs:schema>
```

A.6 CIDOC CRM – Document Type Definition

```
<?xml version='1.0' encoding='ISO-8859-1' ?>

<!-- This format encodes all properties of the CIDOC CRM v.4.2.1 as XML ele-
ments. Classes are attached to instance identifiers as additional elements.
This format can be used to encode CIDOC CRM instances for data transport. Note
that it does NOT enforce the correct combination of classes and properties. In
order to validate the latter, please use a transformation utility to RDF and
validate the RDF instance.
Created by Lida Harami for ICS-FORTH (ISL-ICS), September 2007)
 -->

<!ENTITY % ATTRIB "(Identifier, in_class+, (P1F.is_identified_by |
P1B.identifies |
P2F.has_type |
P2B.is_type_of |
P3F.has_note |
P3.1F.has_type |
P4F.has_time-span |
P4B.is_time-span_of |
P5F.consists_of |
P5B.forms_part_of |
P7F.took_place_at |
P7B.witnessed |
P8F.took_place_on_or_within |
P8B.witnessed |
P9F.consists_of |
P9B.forms_part_of |
P10F.falls_within |
P10B.contains |
P11F.had_participant |
P11B.participated_in |
P12F.occurred_in_the_presence_of |
P12B.was_present_at |
P13F.destroyed |
P13B.was_destroyed_by |
P14F.carried_out_by |
P14B.performed |
P14.1F.in_the_role_of |
P15F.was_influenced_by |
P15B.influenced |
P16F.used_specific_object |
P16B.was_used_for |
P16.1F_mode_of_use |
P17F.was_motivated_by |
P17B.motivated |
P19F.was_intended_use_of |
P19B.was_made_for |
P19.1F_mode_of_use |
P20F.had_specific_purpose |
P20B.was_purpose_of |
P21F.had_general_purpose |
P21B.was_purpose_of |
P22F.transferred_title_to |
P22B.acquired_title_through |
P23F.transferred_title_from |
P23B.surrendered_title_through |
P24F.transferred_title_of |
P24B.changed_ownership_through |
P25F.moved |
P25B.moved_by |
P26F.moved_to |
P26B.was_destination_of |
P27F.moved_from |
P27B.was_origin_of |
P28F.custody_surrendered_by |
P28B.surrendered_custody_through |
```

```
P29F.custody_received_by |
P29B.received_custody_through |
P30F.transferred_custody_of |
P30B.custody_transferred_through |
P31F.has_modified |
P31B.was_modified_by |
P32F.used_general_technique |
P32B.was_technique_of |
P33F.used_specific_technique |
P33B.was_used_by |
P34F.concerned |
P34B.was_assessed_by |
P35F.has_identified |
P35B.identified_by |
P36F.registered |
P36B.was_registered_by |
P37F.assigned |
P37B.was_assigned_by |
P38F.deassigned |
P38B.was_deassigned_by |
P39F.measured |
P39B.was_measured_by |
P40F.observed_dimension |
P40B.was_observed_in |
P41F.classified |
P41B.was_classified_by |
P42F.assigned |
P42B.was_assigned_by |
P43F.has_dimension |
P43B.is_dimension_of |
P44F.has_condition |
P44B.condition_of |
P45F.consists_of |
P45B.is_incorporated_in |
P46F.is_composed_of |
P46B.forms_part_of |
P47F.is_identified_by |
P47B.identifies |
P48F.has_preferred_identifier |
P48B.is_preferred_identifier_of |
P49F.has_former_or_current_keeper |
P49B.is_former_or_current_keeper_of |
P50F.has_current_keeper |
P50B.is_current_keeper_of |
P51F.has_former_or_current_owner |
P51B.is_former_or_current_owner_of |
P52F.has_current_owner |
P52B.is_current_owner_of |
P53F.has_former_or_current_location |
P53B.is_former_or_current_location_of |
P54F.has_current_permanent_location |
P54B.is_current_permanent_location_of |
P55F.has_current_location |
P55B.currently_holds |
P56F.bears_feature |
P56B.is_found_on |
P57F.has_number_of_parts |
P58F.has_section_definition |
P58B.defines_section |
P59F.has_section |
P59B.is_located_on_or_within |
P62F.depicts |
P62B.is_depicted_by |
P62.1F_mode_of_depiction |
P65F.shows_visual_item |
P65B.is_shown_by |
P67F.refers_to |
P67B.is_referred_to_by |
P67.1F_has_type |
P68F.usually_employs |
```

```
P68B.is_usually_employed_by |
P69F.is_associated_with |
P69.1F_has_type |
P70F.documents |
P70B.is_documented_in |
P71F.lists |
P71B.is_listed_in |
P72F.has_language |
P72B.is_language_of |
P73F.has_translation |
P73B.is_translation_of |
P74F.has_current_or_former_residence |
P74B.is_current_or_former_residence_of |
P75F.possesses |
P75B.is_possessed_by |
P76F.has_contact_point |
P76B.provides_access_to |
P78F.is_identified_by |
P78B.identifies |
P79F.beginning_is_qualified_by |
P80F.end_is_qualified_by |
P81F.ongoing_throughout |
P82F.at_some_time_within |
P83F.had_at_least_duration |
P83B.was_minimum_duration_of |
P84F.had_at_most_duration |
P84B.was_maximum_duration_of |
P86F.falls_within |
P86B.contains |
P87F.is_identified_by |
P87B.identifies |
P88F.consists_of |
P88B.forms_part_of |
P89F.falls_within |
P89B.contains |
P90F.has_value |
P91F.has_unit |
P91B.is_unit_of |
P92F.brought_into_existence |
P92B.was_brought_into_existence_by |
P93F.took_out_of_existence |
P93B.was_taken_out_of_existence_by |
P94F.has_created |
P94B.was_created_by |
P95F.has_formed |
P95B.was_formed_by |
P96F.by_mother |
P96B.gave_birth |
P97F.from_father |
P97B.was_father_for |
P98F.brought_into_life |
P98B.was_born |
P99F.dissolved |
P99B.was_dissolved_by |
P100F.was_death_of |
P100B.died_in |
P101F.had_as_general_use |
P101B.was_use_of |
P102F.has_title |
P102B.is_title_of |
P102.1F_has_type |
P103F.was_intended_for |
P103B.was_intention_of |
P104F.is_subject_to |
P104B.applies_to |
P105F.right_held_by |
P105B.has_right_on |
P106F.is_composed_of |
P106B.forms_part_of |
P107F.has_current_or_former_member |
```

```
P107B.is_current_or_former_member_of |
P108F.has_produced |
P108B.was_produced_by |
P109F.has_current_or_former_curator |
P109B.is_current_or_former_curator_of |
P110F.augmented |
P110B.was_augmented_by |
P111F.added |
P111B.was_added_by |
P112F.diminished |
P112B.was_diminished_by |
P113F.removed |
P113B.was_removed_by |
P114F.is_equal_in_time_to |
P115F.finishes |
P115B.is_finished_by |
P116F.starts |
P116B.is_started_by |
P117F.occurs_during |
P117B.includes |
P118F.overlaps_in_time_with |
P118B.is_overlapped_in_time_by |
P119F.meets_in_time_with |
P119B.is_met_in_time_by |
P120F.occurs_before |
P120B.occurs_after |
P121F.overlaps_with |
P122F.borders_with |
P123F.resulted_in |
P123B.resulted_from |
P124F.transformed |
P124B.was_transformed_by |
P125F.used_object_of_type |
P125B.was_type_of_object_used_in |
P126F.employed |
P126B.was_employed_in |
P127F.has_broader_term |
P127B.has_narrower_term |
P128F.carries |
P128B.is_carried_by |
P129F.is_about |
P129B.is_subject_of |
P130F.shows_features_of |
P130B.features_are_also_found_on |
P130.1F_kind_of_similarity |
P131F.is_identified_by |
P131B.identifies |
P132F.overlaps_with |
P133F.is_separated_from |
P134F.continued |
P134B.was_continued_by |
P135F.created_type |
P135B.was_created_by |
P136F.was_based_on |
P136B.supported_type_creation |
P136.1F_in_the_taxonomic_role |
P137F.is_exemplified_by |
P137B.exemplifies |
P137.1F_in_the_taxonomic_role |
P138F.represents |
P138B.has_representation |
P138.1F_mode_of_representation |
P139F.has_alternative_form |
P139.1F_has_type |
P140F.assigned_attribute_to |
P140B.was_attributed_by |
P141F.assigned |
P141B.was_assigned_by)*)">
```

```
<!ELEMENT String (#PCDATA)>
<!ELEMENT Number (#PCDATA)>
<!ELEMENT CRMset (E1.CRM_Entity*)>
<!ELEMENT Identifier (#PCDATA)>
<!ELEMENT E1.CRM_Entity (%ATTRIB;)>
<!ELEMENT in_class (#PCDATA)>
<!ELEMENT P1F.is_identified_by (%ATTRIB;)>
<!ELEMENT P1B.identifies (%ATTRIB;)>
<!ELEMENT P2F.has_type (%ATTRIB;)>
<!ELEMENT P2B.is_type_of (%ATTRIB;)>
<!ELEMENT P3F.has_note  (P3.1F.has_type*, String)>
<!ELEMENT P3.1F.has_type  (#PCDATA)>
<!ELEMENT P4F.has_time-span (%ATTRIB;)>
<!ELEMENT P4B.is_time-span_of (%ATTRIB;)>
<!ELEMENT P5F.consists_of (%ATTRIB;)>
<!ELEMENT P5B.forms_part_of (%ATTRIB;)>
<!ELEMENT P7F.took_place_at (%ATTRIB;)>
<!ELEMENT P7B.witnessed (%ATTRIB;)>
<!ELEMENT P8F.took_place_on_or_within (%ATTRIB;)>
<!ELEMENT P8B.witnessed (%ATTRIB;)>
<!ELEMENT P9F.consists_of (%ATTRIB;)>
<!ELEMENT P9B.forms_part_of (%ATTRIB;)>
<!ELEMENT P10F.falls_within (%ATTRIB;)>
<!ELEMENT P10B.contains (%ATTRIB;)>
<!ELEMENT P11F.had_participant (%ATTRIB;)>
<!ELEMENT P11B.participated_in (%ATTRIB;)>
<!ELEMENT P12F.occurred_in_the_presence_of (%ATTRIB;)>
<!ELEMENT P12B.was_present_at (%ATTRIB;)>
<!ELEMENT P13F.destroyed (%ATTRIB;)>
<!ELEMENT P13B.was_destroyed_by (%ATTRIB;)>
<!ELEMENT P14F.carried_out_by (P14.1F.in_the_role_of*, %ATTRIB;)>
<!ELEMENT P14B.performed (P14.1F.in_the_role_of*, %ATTRIB;)>
<!ELEMENT P14.1F.in_the_role_of (#PCDATA)>
<!ELEMENT P15F.was_influenced_by (%ATTRIB;)>
<!ELEMENT P15B.influenced (%ATTRIB;)>
<!ELEMENT P16F.used_specific_object (P16.1F_mode_of_use*, %ATTRIB;)>
<!ELEMENT P16B.was_used_for (P16.1F_mode_of_use*, %ATTRIB;)>
<!ELEMENT P16.1F_mode_of_use (#PCDATA)>
<!ELEMENT P17F.was_motivated_by (%ATTRIB;)>
<!ELEMENT P17B.motivated (%ATTRIB;)>
<!ELEMENT P19F.was_intended_use_of (P19.1F_mode_of_use*, %ATTRIB;)>
<!ELEMENT P19B.was_made_for (P19.1F_mode_of_use*, %ATTRIB;)>
<!ELEMENT P19.1F_mode_of_use (#PCDATA)>
<!ELEMENT P20F.had_specific_purpose (%ATTRIB;)>
<!ELEMENT P20B.was_purpose_of (%ATTRIB;)>
<!ELEMENT P21F.had_general_purpose (%ATTRIB;)>
<!ELEMENT P21B.was_purpose_of (%ATTRIB;)>
<!ELEMENT P22F.transferred_title_to (%ATTRIB;)>
<!ELEMENT P22B.acquired_title_through (%ATTRIB;)>
<!ELEMENT P23F.transferred_title_from (%ATTRIB;)>
<!ELEMENT P23B.surrendered_title_through (%ATTRIB;)>
<!ELEMENT P24F.transferred_title_of (%ATTRIB;)>
<!ELEMENT P24B.changed_ownership_through (%ATTRIB;)>
<!ELEMENT P25F.moved (%ATTRIB;)>
<!ELEMENT P25B.moved_by (%ATTRIB;)>
<!ELEMENT P26F.moved_to (%ATTRIB;)>
<!ELEMENT P26B.was_destination_of (%ATTRIB;)>
<!ELEMENT P27F.moved_from (%ATTRIB;)>
<!ELEMENT P27B.was_origin_of (%ATTRIB;)>
<!ELEMENT P28F.custody_surrendered_by (%ATTRIB;)>
<!ELEMENT P28B.surrendered_custody_through (%ATTRIB;)>
<!ELEMENT P29F.custody_received_by (%ATTRIB;)>
<!ELEMENT P29B.received_custody_through (%ATTRIB;)>
<!ELEMENT P30F.transferred_custody_of (%ATTRIB;)>
<!ELEMENT P30B.custody_transferred_through (%ATTRIB;)>
<!ELEMENT P31F.has_modified (%ATTRIB;)>
<!ELEMENT P31B.was_modified_by (%ATTRIB;)>
<!ELEMENT P32F.used_general_technique (%ATTRIB;)>
<!ELEMENT P32B.was_technique_of (%ATTRIB;)>
```

```
<!ELEMENT P33F.used_specific_technique (%ATTRIB;)>
<!ELEMENT P33B.was_used_by (%ATTRIB;)>
<!ELEMENT P34F.concerned (%ATTRIB;)>
<!ELEMENT P34B.was_assessed_by (%ATTRIB;)>
<!ELEMENT P35F.has_identified (%ATTRIB;)>
<!ELEMENT P35B.identified_by (%ATTRIB;)>
<!ELEMENT P36F.registered (%ATTRIB;)>
<!ELEMENT P36B.was_registered_by (%ATTRIB;)>
<!ELEMENT P37F.assigned (%ATTRIB;)>
<!ELEMENT P37B.was_assigned_by (%ATTRIB;)>
<!ELEMENT P38F.deassigned (%ATTRIB;)>
<!ELEMENT P38B.was_deassigned_by (%ATTRIB;)>
<!ELEMENT P39F.measured (%ATTRIB;)>
<!ELEMENT P39B.was_measured_by (%ATTRIB;)>
<!ELEMENT P40F.observed_dimension (%ATTRIB;)>
<!ELEMENT P40B.was_observed_in (%ATTRIB;)>
<!ELEMENT P41F.classified (%ATTRIB;)>
<!ELEMENT P41B.was_classified_by (%ATTRIB;)>
<!ELEMENT P42F.assigned (%ATTRIB;)>
<!ELEMENT P42B.was_assigned_by (%ATTRIB;)>
<!ELEMENT P43F.has_dimension (%ATTRIB;)>
<!ELEMENT P43B.is_dimension_of (%ATTRIB;)>
<!ELEMENT P44F.has_condition (%ATTRIB;)>
<!ELEMENT P44B.condition_of (%ATTRIB;)>
<!ELEMENT P45F.consists_of (%ATTRIB;)>
<!ELEMENT P45B.is_incorporated_in (%ATTRIB;)>
<!ELEMENT P46F.is_composed_of (%ATTRIB;)>
<!ELEMENT P46B.forms_part_of (%ATTRIB;)>
<!ELEMENT P47F.is_identified_by (%ATTRIB;)>
<!ELEMENT P47B.identifies (%ATTRIB;)>
<!ELEMENT P48F.has_preferred_identifier (%ATTRIB;)>
<!ELEMENT P48B.is_preferred_identifier_of (%ATTRIB;)>
<!ELEMENT P49F.has_former_or_current_keeper (%ATTRIB;)>
<!ELEMENT P49B.is_former_or_current_keeper_of (%ATTRIB;)>
<!ELEMENT P50F.has_current_keeper (%ATTRIB;)>
<!ELEMENT P50B.is_current_keeper_of (%ATTRIB;)>
<!ELEMENT P51F.has_former_or_current_owner (%ATTRIB;)>
<!ELEMENT P51B.is_former_or_current_owner_of (%ATTRIB;)>
<!ELEMENT P52F.has_current_owner (%ATTRIB;)>
<!ELEMENT P52B.is_current_owner_of (%ATTRIB;)>
<!ELEMENT P53F.has_former_or_current_location (%ATTRIB;)>
<!ELEMENT P53B.is_former_or_current_location_of (%ATTRIB;)>
<!ELEMENT P54F.has_current_permanent_location (%ATTRIB;)>
<!ELEMENT P54B.is_current_permanent_location_of (%ATTRIB;)>
<!ELEMENT P55F.has_current_location (%ATTRIB;)>
<!ELEMENT P55B.currently_holds (%ATTRIB;)>
<!ELEMENT P56F.bears_feature (%ATTRIB;)>
<!ELEMENT P56B.is_found_on (%ATTRIB;)>
<!ELEMENT P57F.has_number_of_parts (#PCDATA)>
<!ELEMENT P58F.has_section_definition (%ATTRIB;)>
<!ELEMENT P58B.defines_section (%ATTRIB;)>
<!ELEMENT P59F.has_section (%ATTRIB;)>
<!ELEMENT P59B.is_located_on_or_within (%ATTRIB;)>
<!ELEMENT P62F.depicts (P62.1F_mode_of_depiction*, %ATTRIB;)>
<!ELEMENT P62B.is_depicted_by (P62.1F_mode_of_depiction*, %ATTRIB;)>
<!ELEMENT P62.1F_mode_of_depiction (#PCDATA)>
<!ELEMENT P65F.shows_visual_item (%ATTRIB;)>
<!ELEMENT P65B.is_shown_by (%ATTRIB;)>
<!ELEMENT P67F.refers_to (P67.1F_has_type*, %ATTRIB;)>
<!ELEMENT P67B.is_referred_to_by (P67.1F_has_type*, %ATTRIB;)>
<!ELEMENT P67.1F_has_type (#PCDATA)>
<!ELEMENT P68F.usually_employs (%ATTRIB;)>
<!ELEMENT P68B.is_usually_employed_by (%ATTRIB;)>
<!ELEMENT P69F.is_associated_with (P69.1F_has_type*, %ATTRIB;)>
<!ELEMENT P69.1F_has_type (#PCDATA)>
<!ELEMENT P70F.documents (%ATTRIB;)>
<!ELEMENT P70B.is_documented_in (%ATTRIB;)>
<!ELEMENT P71F.lists (%ATTRIB;)>
<!ELEMENT P71B.is_listed_in (%ATTRIB;)>
<!ELEMENT P72F.has_language (%ATTRIB;)>
```

```
<!ELEMENT P72B.is_language_of (%ATTRIB;)>
<!ELEMENT P73F.has_translation (%ATTRIB;)>
<!ELEMENT P73B.is_translation_of (%ATTRIB;)>
<!ELEMENT P74F.has_current_or_former_residence (%ATTRIB;)>
<!ELEMENT P74B.is_current_or_former_residence_of (%ATTRIB;)>
<!ELEMENT P75F.possesses (%ATTRIB;)>
<!ELEMENT P75B.is_possessed_by (%ATTRIB;)>
<!ELEMENT P76F.has_contact_point (%ATTRIB;)>
<!ELEMENT P76B.provides_access_to (%ATTRIB;)>
<!ELEMENT P78F.is_identified_by (%ATTRIB;)>
<!ELEMENT P78B.identifies (%ATTRIB;)>
<!ELEMENT P79F.beginning_is_qualified_by (#PCDATA)>
<!ELEMENT P80F.end_is_qualified_by (#PCDATA)>
<!ELEMENT P81F.ongoing_throughout (#PCDATA)>
<!ELEMENT P82F.at_some_time_within (#PCDATA)>
<!ELEMENT P83F.had_at_least_duration (%ATTRIB;)>
<!ELEMENT P83B.was_minimum_duration_of (%ATTRIB;)>
<!ELEMENT P84F.had_at_most_duration (%ATTRIB;)>
<!ELEMENT P84B.was_maximum_duration_of (%ATTRIB;)>
<!ELEMENT P86F.falls_within (%ATTRIB;)>
<!ELEMENT P86B.contains (%ATTRIB;)>
<!ELEMENT P87F.is_identified_by (%ATTRIB;)>
<!ELEMENT P87B.identifies (%ATTRIB;)>
<!ELEMENT P88F.consists_of (%ATTRIB;)>
<!ELEMENT P88B.forms_part_of (%ATTRIB;)>
<!ELEMENT P89F.falls_within (%ATTRIB;)>
<!ELEMENT P89B.contains (%ATTRIB;)>
<!ELEMENT P90F.has_value (#PCDATA)>
<!ELEMENT P91F.has_unit (%ATTRIB;)>
<!ELEMENT P91B.is_unit_of (%ATTRIB;)>
<!ELEMENT P92F.brought_into_existence (%ATTRIB;)>
<!ELEMENT P92B.was_brought_into_existence_by (%ATTRIB;)>
<!ELEMENT P93F.took_out_of_existence (%ATTRIB;)>
<!ELEMENT P93B.was_taken_out_of_existence_by (%ATTRIB;)>
<!ELEMENT P94F.has_created (%ATTRIB;)>
<!ELEMENT P94B.was_created_by (%ATTRIB;)>
<!ELEMENT P95F.has_formed (%ATTRIB;)>
<!ELEMENT P95B.was_formed_by (%ATTRIB;)>
<!ELEMENT P96F.by_mother (%ATTRIB;)>
<!ELEMENT P96B.gave_birth (%ATTRIB;)>
<!ELEMENT P97F.from_father (%ATTRIB;)>
<!ELEMENT P97B.was_father_for (%ATTRIB;)>
<!ELEMENT P98F.brought_into_life (%ATTRIB;)>
<!ELEMENT P98B.was_born (%ATTRIB;)>
<!ELEMENT P99F.dissolved (%ATTRIB;)>
<!ELEMENT P99B.was_dissolved_by (%ATTRIB;)>
<!ELEMENT P100F.was_death_of (%ATTRIB;)>
<!ELEMENT P100B.died_in (%ATTRIB;)>
<!ELEMENT P101F.had_as_general_use (%ATTRIB;)>
<!ELEMENT P101B.was_use_of (%ATTRIB;)>
<!ELEMENT P102F.has_title (P102.1F_has_type*, %ATTRIB;)>
<!ELEMENT P102B.is_title_of (P102.1F_has_type*, %ATTRIB;)>
<!ELEMENT P102.1F_has_type (#PCDATA)>
<!ELEMENT P103F.was_intended_for (%ATTRIB;)>
<!ELEMENT P103B.was_intention_of (%ATTRIB;)>
<!ELEMENT P104F.is_subject_to (%ATTRIB;)>
<!ELEMENT P104B.applies_to (%ATTRIB;)>
<!ELEMENT P105F.right_held_by (%ATTRIB;)>
<!ELEMENT P105B.has_right_on (%ATTRIB;)>
<!ELEMENT P106F.is_composed_of (%ATTRIB;)>
<!ELEMENT P106B.forms_part_of (%ATTRIB;)>
<!ELEMENT P107F.has_current_or_former_member (%ATTRIB;)>
<!ELEMENT P107B.is_current_or_former_member_of (%ATTRIB;)>
<!ELEMENT P108F.has_produced (%ATTRIB;)>
<!ELEMENT P108B.was_produced_by (%ATTRIB;)>
<!ELEMENT P109F.has_current_or_former_curator (%ATTRIB;)>
<!ELEMENT P109B.is_current_or_former_curator_of (%ATTRIB;)>
<!ELEMENT P110F.augmented (%ATTRIB;)>
<!ELEMENT P110B.was_augmented_by (%ATTRIB;)>
<!ELEMENT P111F.added (%ATTRIB;)>
```

```
<!ELEMENT P111B.was_added_by (%ATTRIB;)>
<!ELEMENT P112F.diminished (%ATTRIB;)>
<!ELEMENT P112B.was_diminished_by (%ATTRIB;)>
<!ELEMENT P113F.removed (%ATTRIB;)>
<!ELEMENT P113B.was_removed_by (%ATTRIB;)>
<!ELEMENT P114F.is_equal_in_time_to (%ATTRIB;)>
<!ELEMENT P115F.finishes (%ATTRIB;)>
<!ELEMENT P115B.is_finished_by (%ATTRIB;)>
<!ELEMENT P116F.starts (%ATTRIB;)>
<!ELEMENT P116B.is_started_by (%ATTRIB;)>
<!ELEMENT P117F.occurs_during (%ATTRIB;)>
<!ELEMENT P117B.includes (%ATTRIB;)>
<!ELEMENT P118F.overlaps_in_time_with (%ATTRIB;)>
<!ELEMENT P118B.is_overlapped_in_time_by (%ATTRIB;)>
<!ELEMENT P119F.meets_in_time_with (%ATTRIB;)>
<!ELEMENT P119B.is_met_in_time_by (%ATTRIB;)>
<!ELEMENT P120F.occurs_before (%ATTRIB;)>
<!ELEMENT P120B.occurs_after (%ATTRIB;)>
<!ELEMENT P121F.overlaps_with (%ATTRIB;)>
<!ELEMENT P122F.borders_with (%ATTRIB;)>
<!ELEMENT P123F.resulted_in (%ATTRIB;)>
<!ELEMENT P123B.resulted_from (%ATTRIB;)>
<!ELEMENT P124F.transformed (%ATTRIB;)>
<!ELEMENT P124B.was_transformed_by (%ATTRIB;)>
<!ELEMENT P125F.used_object_of_type (%ATTRIB;)>
<!ELEMENT P125B.was_type_of_object_used_in (%ATTRIB;)>
<!ELEMENT P126F.employed (%ATTRIB;)>
<!ELEMENT P126B.was_employed_in (%ATTRIB;)>
<!ELEMENT P127F.has_broader_term (%ATTRIB;)>
<!ELEMENT P127B.has_narrower_term (%ATTRIB;)>
<!ELEMENT P128F.carries (%ATTRIB;)>
<!ELEMENT P128B.is_carried_by (%ATTRIB;)>
<!ELEMENT P129F.is_about (%ATTRIB;)>
<!ELEMENT P129B.is_subject_of (%ATTRIB;)>
<!ELEMENT P130F.shows_features_of (P130.1F_kind_of_similarity*, %ATTRIB;)>
<!ELEMENT P130B.features_are_also_found_on (P130.1F_kind_of_similarity*,
%ATTRIB;)>
<!ELEMENT P130.1F_kind_of_similarity (#PCDATA)>
<!ELEMENT P131F.is_identified_by (%ATTRIB;)>
<!ELEMENT P131B.identifies (%ATTRIB;)>
<!ELEMENT P132F.overlaps_with (%ATTRIB;)>
<!ELEMENT P133F.is_separated_from (%ATTRIB;)>
<!ELEMENT P134F.continued (%ATTRIB;)>
<!ELEMENT P134B.was_continued_by (%ATTRIB;)>
<!ELEMENT P135F.created_type (%ATTRIB;)>
<!ELEMENT P135B.was_created_by (%ATTRIB;)>
<!ELEMENT P136F.was_based_on (P136.1F_in_the_taxonomic_role*, %ATTRIB;)>
<!ELEMENT P136B.supported_type_creation (P136.1F_in_the_taxonomic_role*,
%ATTRIB;)>
<!ELEMENT P136.1F_in_the_taxonomic_role (#PCDATA)>
<!ELEMENT P137F.is_exemplified_by (P137.1F_in_the_taxonomic_role*, %ATTRIB;)>
<!ELEMENT P137B.exemplifies (P137.1F_in_the_taxonomic_role*, %ATTRIB;)>
<!ELEMENT P137.1F_in_the_taxonomic_role (#PCDATA)>
<!ELEMENT P138F.represents (P138.1F_mode_of_representation*, %ATTRIB;)>
<!ELEMENT P138B.has_representation (P138.1F_mode_of_representation*,
%ATTRIB;)>
<!ELEMENT P138.1F_mode_of_representation (#PCDATA)>
<!ELEMENT P139F.has_alternative_form (P139.1F_has_type*, %ATTRIB;)>
<!ELEMENT P139.1F_has_type (#PCDATA)>
<!ELEMENT P140F.assigned_attribute_to (%ATTRIB;)>
<!ELEMENT P140B.was_attributed_by (%ATTRIB;)>
<!ELEMENT P141F.assigned (#PCDATA)>
<!ELEMENT P141B.was_assigned_by (%ATTRIB;)>
```

A.7 Digitale Anlagen

/Ausgangsdaten

/Beispiele

/CIDOC_CRM

/EnterpriseArchitect

/Masterarbeit

/ShapeChange

/XML_Export_automatisiert

/XML_Export_manuell

/XSD

/readme.txt