Design eines Mappings und Implementierung eines Converters von LimeSurvey Umfragen und Antworten in CDISC ODM

Bachelorarbeit

vorgelegt von:

Antonius Johannes Mende

Matrikelnummer: 461 328

Studiengang: Informatik 1.FB

Thema gestellt von:

Dr. Ludger Becker

Arbeit betreut durch:

Dr. Ludger Becker und Dr. Tobias Brix

Münster, 6. Juli 2021

Inhaltsverzeichnis

1. Einleitung

Das Thema dieser Arbeit ist die Konvertierung von LimeSurvey Archiven (LSA) in das Operational Data Model (ODM). LimeSurvey ist ein Werkzeug, mit dem man einfach Umfragen erstellen kann, welche dann wiederum von beliebig vielen Teilnehmern beantwortet werden können. Umfragewerkzeuge lassen sich sehr vielseitig einsetzen, um die Meinungen von Kunden, Patienten oder jedem anderen Menschen einzuholen. Gerade in den heutigen Zeiten, wo immer mehr Daten gesammelt und verarbeitet werden können, ist es wichtig, Kompatibilität zwischen verschiedenen Werkzeugen und Formaten herstellen zu können. Das Institut für Medizinische Informatik an der WWU ist eine der vielen Institutionen, welche LimeSurvey nutzt, um die Meinungen und Erfahrungen von Patienten, das „electronic partient-reported outcome“(ePRO), einzuholen.

Auch das Bundesministerium für Bildung und Forschung (BMBF) hat schon vor Jahren erkannt, dass die digitale Verwaltung medizinischer Datensätze in der Medizininformatik stetig an Relevanz gewinnt und 150 Millionen Euro für eine Initiative bereitgestellt, welche verschiedene Institutionen in der Aufbau- und Vernetzungsphase unterstützen soll [1].

1.1. Problem

Beide Dateiformate sind in der „eXtensible Markup Language“(XML) geschrieben. Der XML-Standard lässt dem Programmierer sehr viel Freiheit, was das Design eines XML-Formates angeht. Dadurch sind verschiedene XML-Formate im Regelfall nicht kompatibel beziehungsweise untereinander austauschbar. Das macht es notwendig, XML-Dokumente zu konvertieren, um ein Dokument in einem Format mit einem Werkzeug nutzen zu können, welches nur ein anderes XML-Format unterstützt. Als Zielformat wird ODM-XML des *Clinical Data Interchange Standards Consortium* (CDISC) verwendet. ODM wurde mit dem Ziel entwickelt, den Austausch und die Archivierung von Forschungsdaten und anderen damit verbundenen Daten zu ermöglichen. Durch die Unabhängigkeit des Formates von spezifischen Plattformen oder Firmen wird es durch weit mehr Werkzeuge unterstützt als ein Format wie LSA, welches von LimeSurvey selbst und zu deren eigenen Zwecken entwickelt wurde. Dies sieht man auch an dem Institut für Medizinische Informatik, welche dieses Projekt betreut. Dort wird ODM zur Verwaltung klinischer Daten genutzt, viele davon sind über das „Medical Data Models“-Portal (MDM-Portal)[2] abrufbar.

Gleichzeitig ist aber auch LimeSurvey das Umfragewerkzeug der Wahl, um Daten von Patienten zu sammeln. Diese gesammelten Daten müssen dann exportiert und konvertiert werden, um sie in bestehenden Systemen einpflegen zu können.

1.2. Ziel

Ziel der Arbeit ist es, genau darzustellen, wie das LimeSurvey Archiv und die darin enthaltenen Dateien aufgebaut sind. Dann soll gezeigt werden, wie man ein LimeSurvey Archiv (LSA) in ein ODM Dokument umwandeln kann, indem ein Mapping erstellt wird. Anschließend soll anhand des Mappings ein Konverter implementiert werden, welcher die Umwandlung durchführt. Es sollen so viele der Forschungsdaten wie möglich übertragen werden, zusätzliche Daten, welche z.B. die Darstellung der Daten betreffen, sollen nicht mit übertragen werden. Auch ein Konverter in die andere Richtung, also von ODM Dateien zu LimeSurvey, soll diskutiert werden und eine möglichst funktionsreiche Version soll anschließend implementiert werden.

2. Methodik

2.1. LimeSurvey

LimeSurvey ist ein von der gleichnamigen deutschen Firma entwickeltes Werkzeug für Umfragen. Laut ihrer Website[3] ist LimeSurvey ein für Einsteiger und Profis, sowie für Privatpersonen als auch Institutionen gut geeignetes Werkzeug, um die Meinungen, Interessen und Entscheidungsgrundlagen einer Zielgruppe herauszufinden. Es wird seit 2006 an der Software entwickelt und sie ist sowohl als Cloud-Edition über die Firma erhältlich, sowie auch als OpenSource-Projekt in der Community-Edition.

Falls man die Cloud-Edition nutzen will, hat man fünf Optionen: *Free*, *Basic*, *Expert*, *Enterprise* und *Corporate*. Diese unterscheiden sich vor allem durch die Zahl an Antworten pro Monat, die Menge an Upload-Speicher, E-Mail-Support, White-Label-Umfragen, Alias-Domains und die Entfernung des LimeSurvey-Brandings. Innerhalb der *Corporate*-Lösung kann man noch dedizierte Server und die Nutzung der Security Assertion Markup Language (SAML) beantragen.

2.1.1. Community-Edition

Zum Erstellen von Umfragen, welche in dieser Arbeit gebraucht werden, wird eine Version der Community-Edition genutzt. Diese nutzt Docker, um die Website möglichst separat vom restlichen System zu halten. So muss man Abhängigkeiten nicht auf dem eigenen System installieren und verwalten. Zur Verwaltung der Docker-Version des „Australian Consortium for Social & Political Research Inc.“ (ACSPRI) wird eine Docker-Compose-Datei genutzt, welche online zur Verfügung steht [4].

2.1.2. Features

Alle her aufgelisteten Informationen wurden der Dokumentation, dem LimeSurvey Manual[5] entnommen.

ExpressionScript

Mit Expression Script lässt sich komplexe Logik in eine Umfrage einbringen. Die LimeSurvey-interne Skriptsprache lässt Bedingungen zu, unter denen be-

stimmte Fragen oder Antworten angezeigt werden sollen. Man kann hier sowohl Antworten auf vorherige Fragen einbinden, als auch Informationen über den Teilnehmer, welche er vorher angegeben hat beziehungsweise welche über ihn gespeichert wurden. Man kann mehrere Szenarios designen und es sind Vergleiche mit den Standard-Operatoren sowie RegEx möglich.

Timings

Es ist möglich, genau festzulegen, wie viel Zeit ein Nutzer zum Beantworten einer Frage hat. Es können Warnmeldungen zu bestimmten Abschnitten innerhalb der Zeitperiode angezeigt werden, die CSS-Klasse, welche Aspekte der Darstellung einer Frage bestimmt, kann angepasst werden und das Beantworten anderer Fragen kann unterbunden werden.

2.1.3. Fragegruppen

Fragen in LimeSurvey sind in Fragegruppen unterteilt. Jede Frage ist genau einer Gruppe zugeordnet. Eine Gruppe kann beliebig viele Fragen enthalten, weiterhin hat sie einen Titel, eine Beschreibung, eine Randomisierungsgruppe und eine Relevanz-Gleichung, wo mittels ExpressionScript (siehe Abschnitt 2.1.2) angegeben werden kann, wann die Fragen dieser Gruppe angezeigt werden sollen. Es darf beliebig viele Fragegruppen geben.

2.1.4. Fragetypen

Es gibt 36 Fragetypen in LimeSurvey, welche in fünf Kategorien unterteilt sind. Davon sind allerdings nicht alle tatsächlich Fragen, auf die der Teilnehmer antworten kann, insgesamt gibt es vier Fragetypen, welche nicht explizit Fragen sind. Die möglichen Antworten auf Freitext- und Zahlenfragen lassen sich mittels RegEx limitieren. Es gibt pro Frage einen optionalen Hilfstext. Fragen mit vordefinierten Antworten erhalten eine „Keine Antwort“-Option, wenn sie nicht verpflichtend sind. Im folgenden werden alle Typen einmal aufgelistet und kurz beschrieben.

Einfachauswahl

Auf folgende Fragen kann man maximal eine Antwort geben.

5 Punkte Wahl Hier kann auf einer Skala von 1 bis 5 ein Wert ausgewählt werden.

Liste Hier kann aus einer vordefinierten Liste eine Antwort gewählt werden. Es gibt drei unterschiedliche Darstellungsmöglichkeiten für diese Liste, entweder als Dropdown-Menü, als Bootstrap-Buttons oder mit Radio-Buttons neben den Antworten.

Liste mit Kommentar Dies ist eine Listen-Frage wie oben, allerdings kann für die Frage auch noch ein Kommentar im Freitext geschrieben werden.

Image-Select-List Hier wird zu der Liste an Antworten noch ein Bild angezeigt. Die Antworten werden mit Radio-Buttons daneben angezeigt.

Matrix

Pro Frage gibt es beliebig viele Subfragen, für jede Subfrage kann eine der vordefinierten Antworten ausgewählt werden. Subfragen werden typischerweise als Zeilen dargestellt, die Antworten als Spalten.

5 Punkte Hier kann auf einer Skala von 1 bis 5 ein Wert ausgewählt werden.

10 Punkte Hier kann auf einer Skala vom 1 bis 10 ein Wert ausgewählt werden.

Z/G/A Hier kann aus den drei Möglichkeiten „Zunahme/Gleich/Abnahme“ eine gewählt werden.

J/N/U Hier kann aus den drei Möglichkeiten „Ja/Nein/Unsicher“ eine gewählt werden.

Matrix (Custom) Hier kann eine eigene Liste an Antwortmöglichkeiten definiert werden.

Matrix nach Spalte Identisch zu Matrix (Custom), allerdings werden Zeilen und Spalten getauscht.

Dual Matrix Es gibt zwei selbst erstellbare Listen an Antwortmöglichkeiten, man kann aus beiden eine Antwort pro Subfrage wählen.

Matrix (Freitext) Hier kann man zwei Listen an Subfragen angeben. Daraus wird dann eine Matrix erstellt, wo in jeder Zelle Freitext als Antwort geschrieben werden kann.

Matrix (Zahlen) Identisch zu Matrix (Freitext), aber es können nur Zahlen als Antwort angegeben werden.

Multiple Choice

Hier können mehrere Antworten aus einer vordefinierten Liste ausgewählt werden. Optional kann ein „Anderes“-Feld zugänglich gemacht werden, in welches Teilnehmer eine eigene Antwort schreiben können.

Multiple Choice Darstellung als Bootstrap-Buttons oder Radio-List ist möglich.

Image Select Hier wird ein Bild mit angezeigt.

Kommentar Hier kann ein Kommentar pro Antwortfeld geschrieben werden.

Textfragen

Browser Detect Erkennt den Webbrowser des Teilnehmers. Dies ist keine Frage, welche der Teilnehmer beantworten kann.

Freitext Hier kann der Teilnehmer einen Text eintippen. Es gibt kurze, lange und riesige Freitexte.

Mehrere Texte Es gibt mehrere Subfragen. Für jede kann ein kurzer Freitext angegeben werden.

Input on Demand Die Frage ist funktional identisch zu „*Mehrere Texte*“, allerdings muss der Teilnehmer einen Knopf drücken, um die nächste Subfrage angezeigt zu bekommen.

Maskenfragen

Datum/Zeit Hier kann ein Datum und eine Uhrzeit angegeben werden. Das Format ist einstellbar, es kann auch ein minimales Datum angegeben werden.

Ja/Nein Hier kann mit Ja oder Nein geantwortet werden.

Gleichung Bei diesem Typ werden Antworten auf vorher gestellte Fragen verwendet, um sie in einer Gleichung zu verarbeiten und das Ergebnis anzuzeigen.

Dateiupload Hier kann eine Datei als Antwort hochgeladen werden.

Geschlecht Hier kann zwischen „Männlich“, „Weiblich“ und „Keine Antwort“ ausgewählt werden.

Sprachumschaltung Hier kann der Teilnehmer die Sprache ändern. Auch das ist keine richtige Frage.

Zahleneingabe Hier kann nach einer Zahl als Antwort gefragt werden.

Mehrfache Zahlen Hier gibt es mehrere Subfragen, auf die jeweils mit einer Zahl geantwortet werden muss. Die Maximal/Minimal-Werte sind limitierbar, genau so wie auch die Maximal- und Minimal-Summe. Es können genaue Gesamtergebnisse verlangt werden und nur Ganzzahlen als Antwort verlangt werden.

Ranking (Advanced) Hier kann der Teilnehmer Elemente aus einer vordifinierten Liste sortieren. Es müssen nicht alle Elemente einsortiert werden.

Textanzeige Hier wird ein Text angezeigt. Auch hier kann nicht geantwortet werden.

2.1.5. Export

LimeSurvey Archiv

Das LimeSurvey Archiv ist eine von sieben Möglichkeiten, Daten von einer LimeSurvey Umfrage zu exportieren. Ein solches Archiv ist eine komprimierte Datei im .lsa-Format, welche mehrere extrahierbare Dateien enthält. Die Zahl und Art der Dateien ist dabei abhängig von den Einstellungen. Zwei Dateien sind immer enthalten:

Die erste Datei enthält die Umfrage-Struktur sowie Informationen über die Art und Weise, wie die Fragen dargestellt werden sollen (die .lss-Datei). Die zweite Datei enthält die Antworten der Teilnehmer (.lsr-Datei). Ausdrücklich erwähnt wird dabei, dass Dateien, die als Antwort auf eine Frage hochgeladen wurden, nicht Teil des Archivs sind. Eine weitere, optionale, Datei ist eine Token-Datei (.lst), welche nur enthalten ist, wenn mögliche Teilnehmer der Umfrage im Vorhinein festgelegt wurden. Hier sind Informationen über den Teilnehmer gespeichert, in den Antworten wird pro Teilnehmer ein Token referenziert, sodass man die Antworten einem Teilnehmer zuordnen kann.

Weitere Exportmöglichkeiten

LSS Es ist möglich, nur die im LSA enthaltene LSS-Datei zu exportieren, das wird mittels dieser Option gemacht.

Excel/.csv Hier sind weitere Einstellungen möglich, wie das Exportieren eines Teils der Antworten oder die Wahl eines bestimmten Formates (Word, Excel, CSV, HTML, PDF).

SPSS SPSS ist ein Software-Paket, welches zur statistischen Analyse von Daten genutzt wird. Auch hier kann ausgewählt werden, welche Antworten exportiert werden sollen. Die Nutzung der Open-Source Version PSPP ist auch möglich.

R R ist eine Alternative zu SPSS, hier werden allerdings alle Daten exportiert.

STATA-xml Auch STATA ist eine komerzielle Lösung für Datenanalyse wie SPSS. Hierfür werden die Daten von LimeSurvey direkt in das proprietäre STATA-Format umgewandelt.

VV Durch „vertical verfication“ ist es möglich, die Antworten zu modifizieren und die modifizierte Datei dann wieder zu importieren.

2.2. Operational Data Model

Das Operational Data Model (ODM) ist einer der Standards, welcher von CDISC entwickelt wurde, um den gesamten Zyklus einer Studie in ihren ver-

schiedensten Formen zu standardisieren. Es dient dazu, sowohl Metadaten als auch klinische Daten einer Studie zu erfassen. Auch administrative Daten sind im Standard enthalten. ODM ist unabhängig von spezifischen Firmen und Plattformen und daher gut zum Austausch zwischen verschiedensten Werkzeugen und Gruppierungen geeignet. Bereits 2006 hat ODM im internationalen Raum Anklang gefunden, in Deutschland allerdings noch nicht[6]. Das ändert sich mittlerweile, das IMI der WWU zum Beispiel nutzt das Format bereits in mehreren Werkzeugen.

2.2.1. Aufbau

Zuerst sollte angemerkt werden, dass hier nicht alle Elemente, die im ODM-Standard definiert sind, vorgestellt werden. Das hat sowohl Zeit- als auch Platzgründe, weiterhin werden viele Elemente für die Abbildung von LimeSurvey nicht gebraucht. Eine vollständige Liste kann in der Dokumentation von ODM gefunden werden. Für diese Arbeit sind besonders zwei Teile des ODM Standards relevant: Die „*Study*“ und die „*ClinicalData*“. Weiterhin gibt es noch „*AdminData*“, „*ReferenceData*“ und „*Association*“. Grundlegend werden andere Elemente mittels einer OID referenziert, ein Attribut, welches jedes Element besitzt, das man referenzieren kann.

Study

Eine Studie hat globale Variablen und grundlegende Definitionen. Weiterhin gibt es die „*MetaDataVersion*“, wo die Struktur der Umfrage festgelegt wird. In der einer Version der Metadaten sind alle Elemente in Referenzen und Definitionen aufgeteilt, wobei die Referenzen immer zuerst im nächst „höhergelegenen“ Element vorkommen.

Das Basiselement einer Studie ist das „*Protocol*“. Dies enthält Elemente des Typs „*StudyEventRef*“. Dann folgen Elemente des Typs „*StudyEventDef*“, welche wiederum Referenzen auf Elemente des Typs „*FormDef*“ beinhalten. Dieses Schema setzt sich noch mit den Elementen „*ItemGroupDef*“ und „*ItemDef*“ fort.

Als weitere Elemente gibt es noch die „*CodeList*“ und die „*Condition*“. In einer „*CodeList*“ werden Antwortmöglichkeiten festgehalten, welche auf eine Frage (Element des Typs „*ItemDef*“) gegeben werden können. Dabei gibt es zwei Arten von CodeLists, die simple und die komplexe. Die simple Liste besteht aus „*EnumeratedItems*“, wo die Antwortmöglichkeit in dem Attribut „*CodedValue*“ angegeben ist. In den klinischen Daten wird als Antwort dann dieser codierte Wert stehen. Die komplexe Liste hat dieses Attribut ebenfalls, in Elementen des Typs „*CodeListItem*“. Es ist auch als Antwort angegeben, aber die tatsächliche Antwortmöglichkeit ist der Text von „*CodeListItem/Decode/TranslatedText*“.

„*ItemDef*“ enthält ein Element „*Question*“, in welchem eine Frage festgehalten wird. Mit „*ConditionDef*“ kann eine Bedingung festgelegt werden. Eine Bedingung hat ein Kind-Element „*FormalExpression*“, dessen Text die Bedingung selber enthält. Das Attribut „*Context*“ gibt an, in welcher Sprache die Bedingung verfasst wurde. Referenziert eine Frage eine Bedingung und evaluiert der Ausdruck zu „true“, wird die Frage nicht angezeigt.

ClinicalData

In der „*ClinicalData*“ gibt es für jeden Teilnehmer der Studie ein Element des Typs „*SubjectData*“. Ab hier ist dann die Stuktur der Studie abgebildet, wobei die „*ItemData*“ das unterste Element ist und die Antwort auf eine Frage beinhaltet.

2.3. Syntaxerweiterung des Instituts für Medizinische Infomatik

ODM besitzt eine sehr lose Definition dessen, was als Bedingung gilt. Daher kann man eine extra Syntax/Sprache angeben, in welcher die Bedingung verfasst ist. Das Institut für Medizinische Informatik der WWU (IMI) hat in einem internen Dokument eine eigene Syntax für Bedingungen entwickelt. Die Antwort auf eine Frage wird mit dem folgenden Ausdruck referenziert:

SE-{SeOID}/F-{FOID}/IG-{IGOID}/I-{IOID}

wobei entsprechende OIDs eingesetzt werden, sodass die Frage eindeutig identifiziert werden kann. Diese Selektoren werden wie Variablen in der sh-Syntax mit einem Dollarsymbol und runden Klammern umschlossen. Durch Angabe des ganzen Pfades wird sichergestellt, dass die Referenz eindeutig ist. Auch eine bestimmte Wiederholung kann angegeben werden. Sonst gibt es logische Operatoren wie „==, !=, ¡=, ¿=, AND, OR, NOT, IN,...“ und mathematische Operatoren wie „+, -, *, /, MIN, MAX, SUM, MEDIAN, MEAN,...“.

2.4. dom4j

dom4j ist eine API, welche die bestehenden Funktionen der *javax*-Bibliothek abstrahiert und so wesentlich simplere Wege bietet, diese zu nutzen[7]. Unter anderem ist es möglich, Elemente in einem XML-Dokument mittels XPath-Ausdrücken zu finden und DOM/SAX-Parser zu nutzen, um ein Dokument zu verarbeiten. Auch ein *XMLWriter* existiert, welcher unter anderem Standardelemente von XML automatisch an ein Dokument anhängt und das Format des Dokuments mittels eines einzeiligen Befehls so anpassen kann, dass es leicht für einen Menschen lesbar ist.

2.5. Reguläre Ausdrücke

Reguläre Ausdrücke (RegEx) sind ein beliebtes Mittel, um in verschiedensten Anwendungsgebieten der Informatik Zeichenketten zu analysieren. Die Hauptfunktion ist es, zu überprüfen, ob eine Zeichenkette einem bestimmten Format entspricht (bzw. ob es ein Match gibt). Weitere Funktionen bestehen zum Beispiel darin, Teile der Zeichenkette zu extrahieren.

Es gibt eine eigene Syntax, in welcher ein Format spezifiziert werden kann[8]. Nennenswert hier ist, dass es verschiedene Syntaxen gibt, je nachdem in welcher Programmiersprache oder Umgebung man arbeitet. Am bekanntesten und am weitesten verbreitet ist die *Perl-Syntax*, Java hat jedoch eine leicht abweichende Syntax. Der Anfang einer Zeichenkette wird mit „^“ markiert, das Ende selbiger mit „$“. Weiterhin steht zum Beispiel „\d“ für eine beliebige Zahl (\\d in Java) oder „.“ für jedes beliebige Zeichen. Dann gibt es Quantoren, mit denen die Häufigkeit eines Ausdrucks bestimmt werden kann. Sie stehen hinter den Ausdrücken, deren Häufigkeit sie bestimmen. Dabei steht „?“ für Null oder ein Vorkommnis, „*“ für beliebig viele und „+“ für mindestens ein Vorkommen. In eckigen Klammern können Zeichen angegeben werden, welche gematcht werden sollen. So verlangt „[abc]“ zum Beispiel, dass ein Zeichen an einer entsprechenden Stelle a, b oder c sein muss.

Es gibt auch *Capture Groups*. Hiermit kann man, nachdem ein Match gefunden wurde, einen Teil der Zeichenkette erhalten, welcher dem Muster in der Capture Group entspricht. Diese wird mit runden Klammern markiert.

Dazu ein Beispiel: Man hat den Ausdruck „X(\d+)X“, es wird also verlangt, dass die Zeichenkette mit einem *X* beginnen muss, dann mindestens eine Ziffer hat und auf einem *X* endet. Zusätzlich hat man noch die Zeichenkette „X0135X“, diese stimmt mit dem Format des Ausdrucks überein. Durch Zugriff auf die Gruppe kann man nun die Zeichenkette „0135“ erhalten.

In Java sieht der Einsatz von regulären Ausdrücken so aus: Als erstes wird ein Ausdruck erstellt. Dazu dient in Java die Klasse *Pattern*:

```
Pattern p = Pattern.compile("{PATTERN}");
```

Als nächstes wird die Zeichenkette angegeben, die überprüft werden soll:

```
Matcher m = p.matcher("{STRING}");
```

Nun wird die Methode *find* der Klasse *Matcher* aufgerufen, welche prüft, ob die Zeichenkette dem Ausdruck entspricht.

```
m.find();
```

Zuletzt kann man auf alle *Capture Groups* zugreifen, dazu dient die Methode *group*:

```
m.group({GRUPPENNUMMER});
```

3. Ergebnisse

3.1. Analyse des LSS- und LSR-Formates

In der Online-Dokumetation von LimeSurvey gibt es zwar eine Sektion zum Thema Export und Export-Formate, der Abschnitt war aber bis vor kurzem leer. Auch jetzt enthält das Kapitel nur eine oberflächliche Beschreibung der Formate, die genaue Struktur wird nicht erklärt. Da diese allerdings benötigt wird, um eine Konvertierung vornehmen zu können, muss zunächst ermittelt werden, wie die .lss- und .lsr-Dateien genau aufgebaut sind.

Dies wurde bewerkstelligt, indem zunächst eine simple Umfrage erstellt wurde, welches eine Fragegruppe und drei Freitextfragen enthält. Dafür wurde eine Instanz der LimeSurvey Community Edition benötigt, diese aufzusetzen war dank existierenden Docker-Compose-Dateien relativ simpel.

Nachdem die Struktur der Umfrage selbst, sowie der Fragegruppen so ermittelt werden konnte, wurden komplexere Dateien erstellt. Diese enthielten zunächst Fragen mit festen Antwortmöglichkeiten, also vor allem Maskenfragen und Multiple Choice Fragen. Nachdem so auch die Struktur der Antworten deutlich geworden war, wurden zuletzt die Matrixfragen eingebaut und die Struktur der Subfragen ermittelt. Zuletzt wurden Anzeigebedingungen in die Umfragen eingebracht und deren Struktur ermittelt.

Das Ergebis dieser Analyse wird im Folgenden dargestellt (Es werden nicht alle existierenden Elemente angesprochen, sondern nur die für diese Arbeit relevanten, eine vollständige Auflistung kann im Anhang A gefunden werden):

3.1.1. Grundlegende Struktur

Das Wurzel-Element in LSS ist „*document*“. Hier sind zuerst grundlegende Informationen wie die Datenbank-Version und den Typ des Dokuments enthalten. Jedes der größeren direkten Kind-Elemente in „*document*“ hat die gleiche Struktur. Es gibt dort zwei Kind-Elemente, „*fields*“ und „*rows*“. In „*rows*“ gibt es „*row*“ Elemente, welche die Informationen selbst in weiteren Elementen enthalten. In „*fields*“ gibt es „*fieldname*“ Elemente, wobei es für jedes mögliche Kind-Element von „*row*“ ein „*fieldname*“ Element mit dem Namen des Kindelements als Text gibt. In den Feldern werden also einmal alle Elemente genannt, die sich innerhalb einer Reihe befinden können.

3.1.2. Fragegruppen

Im Element „*groups*“ werden Metadaten über Fragegruppen gesammelt, wie zum Beispiel die Gruppen-ID. Diese sind aber auch in „*group_l10ns*“ enthalten, darüber hinaus finden sich auch noch weitere Inhalte wie Gruppenname und Sprache in diesem Element. Daher werden später keine Informationen aus „*groups*“ verwendet. Ein „*row*“ Element steht hier für eine Fragegruppe.

3.1.3. Fragen

Das Element „*questions*“ enthält Metadaten über Fragen, wie „*qid*“, „*gid*“, „*title*“ und „*type*“. Ein „*row*“ Element steht hier für eine Frage, pro „Haupt“-Frage in der Umfrage gibt es ein Element in „*questions*“. „*subquestions*“ enthält Subfragen von Arrays, Matrizen et cetera. Diese sind via „*parent_qid*“ an ein Element aus „*questions*“ gekoppelt. Auch in „*subquestions*“ sind nur Metadaten enthalten, jede Subfrage hat, wie die „Haupt“-Fragen auch, eine Fragen-ID.

In „*question_l10ns*“ gibt es nun die tatsächliche Frage in „*question*“, ein Element innerhalb einer Reihe. Weiterhin gibt es mit „*help*“ einen Hilfstext für die Frage, „*language*“ gibt die Sprache des Fragetextes an. Für jedes Element aus „*question*“ und „*subquestion*“ gibt es hier ein Element pro Sprache. Referenziert werden diese mit der „*qid*“.

„*question_attributes*“ enthält Informationen über eine Frage wie ein Prä-/Suffix zu der Antwort, RegEx-Validations-Ausdrücke für die Antwort, Timings und Informationen zur Darstellung einer Frage (Textfeldbreite, Default-Antworten).

3.1.4. Antwortmöglichkeiten

Es gibt eine Reihe an Fragen, für die es eine Menge an vordefinierten Antworten gibt. Diese sind entweder schon durch die Frage festgelegt, wie bei dem Fragetyp „*5 Punkte Wahl*“ oder dem Typ „*Geschlecht*“, oder der Umfrage-Ersteller kann sie selber angeben. Sind die Möglichkeiten schon durch den Typ festgelegt, werden die Antwortmöglichkeiten implizit ermittelt und nie konkret im LSS-Dokument niedergeschrieben.

Hat der Umfrage-Ersteller die Möglichkeiten selber festgelegt, werden diese in „*answers*“ und „*answer_l10ns*“ gespeichert. In „*answers*“ gibt es dabei wieder Metadaten wie „*qid*“, „*aid*“, und einen „*code*“. In „*answer_l10ns*“ hingegen gibt es den Antworttext in „*answer*“, eine „*aid*“ zur Verknüpfung mit den Metadaten und die Sprache in „*language*“. Für jedes Element aus „*answers*“ gibt es pro Sprache ein Element in „*answer_l10ns*“.

3.1.5. Umfrage-Metadaten

In „*surveys*“ findet man Metadaten über die Umfrage, allerdings sind keine davon für die Konvertierung relevant. Beispiele wären Daten darüber, ob Will-

kommenstexte angezeigt werden sollen oder ob IP-Adressen der Teilnehmer gespeichert werden sollen. „*surveyls_languagesettings*“ enthält relevante Informationen wie die „*surveyls_id*“, den Titel in „*surveyls_title*“ und eine Beschreibung der Umfrage in „*surveyls_description*“. „*themes*“ und „*themes_inherited*“ enthalten Informationen über die visuelle Darstellung der Umfrage in LimeSurvey.

3.1.6. LSR-Aufbau

Auch für die Antworten wurde die gleiche Strategie wie für die Umfragestruktur verwendet. Die Ergebnisse sind wie folgt:

Das Wurzel-Element ist wieder „*Document*“, zuerst gibt es wieder einige Metadaten wie der Typ des Dokuments, die Datenbank-Version und die Sprache. Dann gibt es ein „*responses*“ Element, welches dieselbe grundlegende Struktur wie die Elemente in der LSS-Datei besitzt. Für jeden Teilnehmer der Umfrage gibt es ein Element vom Typ „*row*“. Dies enthält eine ID, ein Token, das Absendedatum, die Sprache, einen Seed und die Antworten. Wenn ein Teilnehmer die Umfrage mehrmals ausfüllt, sind die IDs verschieden, die Tokens aber gleich. Bei den Antworten gibt es für jede Frage bzw. Subfrage ein Element mit folgendem Namensschema: „*-{Survey-ID}X{GID}X{QID}{SQID}?{ext}?*“, wobei „*ext*“ folgendes sein kann („other“|„comment“). Man kann bereits am Format sehen, dass es auch für die „other“ und Kommentar-Felder hier eine separate Antwort gibt.

3.1.7. Erstellung eines XML-Schemas

Da die Struktur der Umfrage nun bekannt ist, wird ein XML-Schema für das LSS-Format erstellt. Das sorgt einerseits für eine wesentlich bessere Übersicht über den Aufbau, das kann für zukünftige Arbeit nützlich sein. Andererseits lässt sich dieses Schema später verwenden, um zu überprüfen, ob es sich bei der Eingabedatei um eine valide LSS-Datei handelt.

3.2. Mapping

Da der Aufbau beider Formate nun bekannt ist, muss überlegt werden, wie man beide LimeSurvey-Formate auf ODM abbilden kann. Im Folgenden werden die verschiedenen Teile von LSS nacheinander abgehandelt und es wird erklärt, wie mit überschüssigen Elementen in ODM ungegangen wird. Der Text wird dabei das Vorgehen beschreiben, die exakten Elementnamen können in den Grafiken nachgesehen werden.

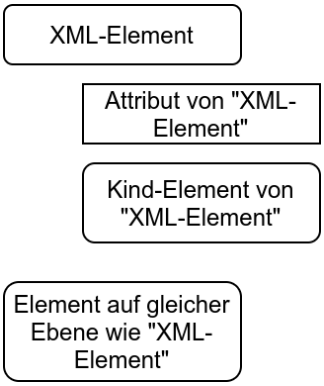


Abbildung 3.1.: Legende für die folgenden Grafiken im Mapping

3.2.1. Dummy Elemente in ODM

ODM besitzt mehr Möglichkeiten, die Abläufe einer Studie oder Umfrage darzustellen, da es als generisches Format natürlich mehr Einsatzzwecke abdecken soll, als das LSS-Format. Dementsprechend gibt es mehrere Wege, LSS auf ODM abzubilden. Durch die hier getroffenen Entscheidungen werden die Elemente „*StudyEvent*“ und „*GlobalVariables*“ überflüssig, da es sich dabei um für eine Studie relevante Felder handelt, es soll aber nur eine Umfrage innerhalb einer Studie erstellt werden.

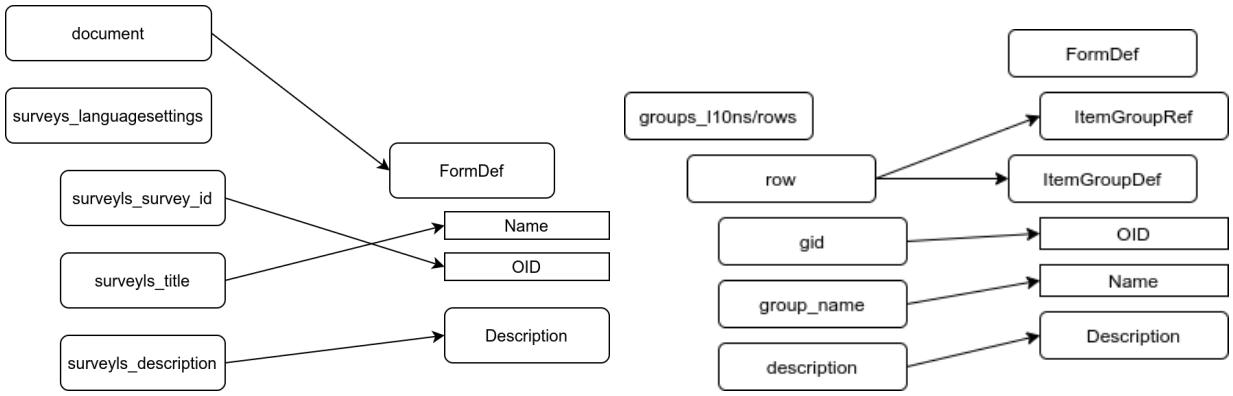
Da diese Elemente dennoch zwingend in ODM vorkommen müssen, werden die globalen Variablen leer gelassen oder mit Dummy-Werten befüllt und es wird ein „*StudyEvent*“ erstellt, welches unabhängig von der Eingabe-Datei immer die gleichen Dummy-Werte hat. Weiterhin wird die „*OID*“ des „*Study*“ Elements auch auf einen Dummy-Wert gesetzt, da wir hier ebenfalls kein Äquivalent in LSS haben. Die „*OID*“ der „*MetaDataVersion*“ soll die Studien-ID behinhalten, da die Version der Metadaten die gesamte Umfrage enthält, es ergibt also Sinn, hier keinen Dummy-Wert einzusetzen sondern eine Referenz auf die Studie mit einzubringen.

3.2.2. Umfrage-Eigenschaften

Innerhalb unseres Dummy-„*StudyEvent's*“ gibt es nun ein Formular, dieses soll die Umfrage repräsentieren. Das „*Repeating*“ Attribut wird auf „No“ gesetzt, da die Umfrage nur einmal pro Teilnehmer ausgefüllt werden soll. Füllt ein Teilnehmer die Umfrage mehrmals aus, gilt das als zwei verschiedene Teilnehmer. Da das Formular die Umfrage repräsentiert, sollen sie dieselbe ID nutzen, als „*Name*“ wird der Titel der Umfrage gesetzt. Die Beschreibung wird ebenfalls in das Formular übernommen.

3.2.3. Fragegruppen

Fragegruppen in LimeSurvey sind fast äquivalent zu „*ItemGroup's*“ in ODM. Entsprechend wird für jede Fragegruppe eine „*ItemGroupDef*“ erstellt, die ID wird direkt in ODM übernommen, der Titel wird als Name verwendet, die Beschreibung kann ebenfalls direkt übernommen werden. Das Attribut „*Repeating*“ wird wieder auf „No“ gesetzt. Entsprechende Referenzen auf die Fragegruppen werden zum Formular hinzugefügt.



(a) Mapping der Umfrage-Eigenschaften **(b)** Mapping der Fragegruppen

Abbildung 3.2.: Diagramme für Fragegruppen und Umfragen

3.2.4. Fragen

Grob gesprochen wird die Frage aus LimeSurveys „*question_l10ns*“ Element bei ODM in „*Question/TranslatedText*“ der Fragendefinition eingetragen, die Sprache aus „*language*“ wird im Attribut „*xml:lang*“ eingetragen. Wie schon bei den Fragegruppen wird die ID übernommen und der Titel wird zum Namen. Der Hilfstext der Frage wird zur Beschreibung in ODM. Die Angabe, ob eine Frage verpflichtend ist, kann ebenfalls übernommen werden, allerdigs wird diese in ODM in der Fragen-Referenz eingetragen, nicht in der Definition.

In ODM gibt es allerdings keine Subfragen, dementsprechend muss eine Frage in LimeSurvey potentiell in mehrere Fragen, eine pro Subfrage, umgewandelt werden, damit diese dann in ODM eingefügt werden können.

Dementsprechend muss jeder Fragetyp aus LimeSurvey einzeln behandelt werden, wobei die Behandlung eines Fragetyps teils in der Behandlung eines anderen Fragetyps enthalten ist. Auch sind die Behandlungen für mehrere Fragetypen identisch. Zum Beispiel werden die Fragetexte der Hauptfrage und der Subfrage konkateniert, um so eine neue Frage zu formulieren. Für die Festlegung der „*OID*“ von Subfragen wird die „*qid*“ der Hauptfrage mit dem Titel der Subfrage konkateniert. Gründe dafür werden in Abschnitt 3.3.7 dargelegt.

Im folgenden werden alle Behandlungen erklärt, mit einer Angabe, für welche Fragetypen diese genutzt werden.

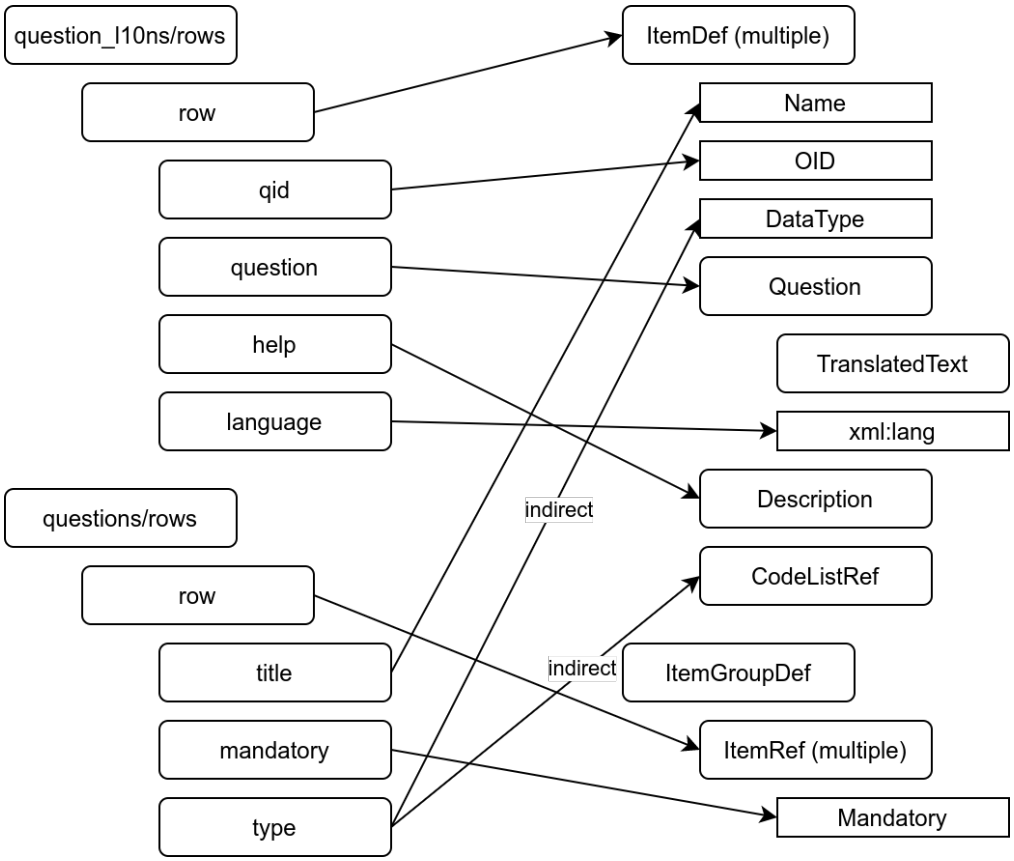


Abbildung 3.3.: Mapping der Fragen

Einfachauswahl

Hier werden die Fragen 1:1 abgebildet, die Antwortmöglichkeiten werden in einer „*CodeList*“ gespeichert. Für den Typ „*Liste mit Kommentar*“ wird eine weitere Frage hinzugefügt, deren „*OID*“ die „*qid*“ der Frage ist, allerdings wird noch „comment“ angehangen. An den Fragetext wird ebenfalls das gleiche angehangen. Bei dem Bild der „*Image-Select-List*“ besteht das Problem, dass dieses nur mittels Link zum Bild auf dem LimeSurvey-Server eingebettet ist. Beim Übertragen der Fragen ist somit auch nur dieser Link enthalten.

Matrix

Für die Matrizen mit Zahlen- oder Freitextantworten wird eine Frage pro Zelle der Matrix hinzugefügt. Für alle weiteren Typen dieser Kategorie bis auf die „*Dual Matrix*“ wird eine Frage pro Subfrage hinzugefügt, die gleiche Liste an Antwortmöglichkeiten wird für jede Subfrage verwendet. Die „*Dual Matrix*“

wird behandelt, als würde die gleiche Frage zwei Mal mit unterschiedlichen Antwortmöglichkeiten dargestellt.

Multiple Choice

Hier wird eine Frage pro Subfrage hinzugefügt, gibt es Kommentare, wird noch eine entsprechende Kommentar-Frage pro Subfrage eingefügt. Das Bild beim Typ „*Image Select*“ wird wie schon bei der Einfachauswahl nur auf Umwegen übernommen.

Textfragen

Hier wird der Fragetext wie oben erklärt übertragen, als „*DataType*“-Attribut in „*ItemDef*“ wird „string“ gesetzt. Das funktioniert für alle drei Freitextfragetypen, kurz, lang und riesig so. Im Falle von „*Mehrere Texte*“ wird wieder eine Frage pro Subfrage genutzt. „*Input on Demand*“ und „*Browser Detect*“ werden nicht gemappt. Für Details siehe Abschnitt 4.1.

Maskenfragen

Für den Typ „*Datum/Zeit*“ wird der „*DataType*“ auf „datetime“ gesetzt. Für „*Zahleneingabe*“ wird der „*DataType*“ auf „float“ gesetzt, außer die Option „Nur Integers“ ist angewählt, dann wird stattdessen „integer“ genutzt. Für „*Mehrfache Zahlen*“ wird wieder eine Frage pro Subfrage erstellt. Für beide Zahlen-Fragen werden Bedingungen, wie den maximalen oder minimalen Antwort-Wert als „*RangeCheck*“ übernommen. Beim Typ „*Gleichung*“ handelt es sich nicht um eine Frage, er wird nicht gemappt (siehe Abschnitt 4.1). Es gibt mehrere Typen, bei denen es feste Antwortmöglichkeiten gibt und daher eine CodeList implizit aus dem Typ erstellt wird: „*Ja/Nein*“, „*Geschlecht*“. „*Ranking*“, „*Textanzeige*“, „*Dateiupload*“ und „*Sprachumschaltung*“ werden nicht gemappt. Für weitere Details siehe Abschnitt 4.1.

3.2.5. Antwortmöglichkeiten

Grundsätzlich wird jedes Mal, wenn es eine vordefinierte Liste an Antwortmöglichkeiten gibt, eine CodeList genutzt, um diese Liste in ODM darzustellen. Für die Fragen, wo man mit 1 bis 5 oder 1 bis 10 antworten kann, wird eine simple Liste genutzt, für alle weiteren Fragen eine komplexe Liste. Das liegt unter anderem daran, dass LimeSurvey fast alle Antworten mit einem Buchstaben darstellt und die tatsächlichen Antworten aus mindestens einem Wort bestehen. So werden komplexere Umwandlungen vermieden und die existierende Struktur wird übernommen. Für selbstdefinierte Antwortmöglichkeiten ist ohnehin eine komplexe Liste notwendig, da der Ersteller der Umfrage beliebige Kombinationen für Code und Antwort definieren kann.

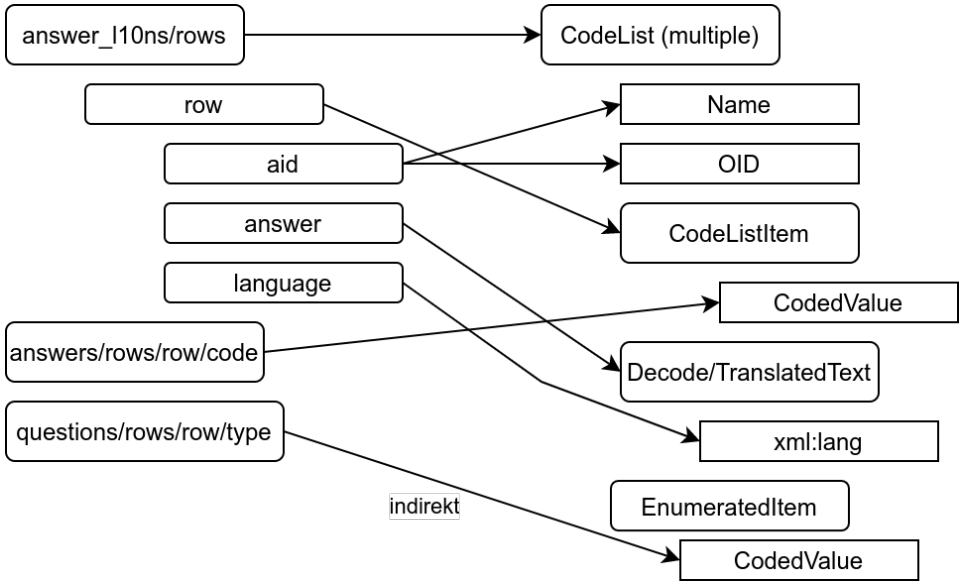


Abbildung 3.4.: Mapping der Antwortmöglichkeiten

3.2.6. Antworten

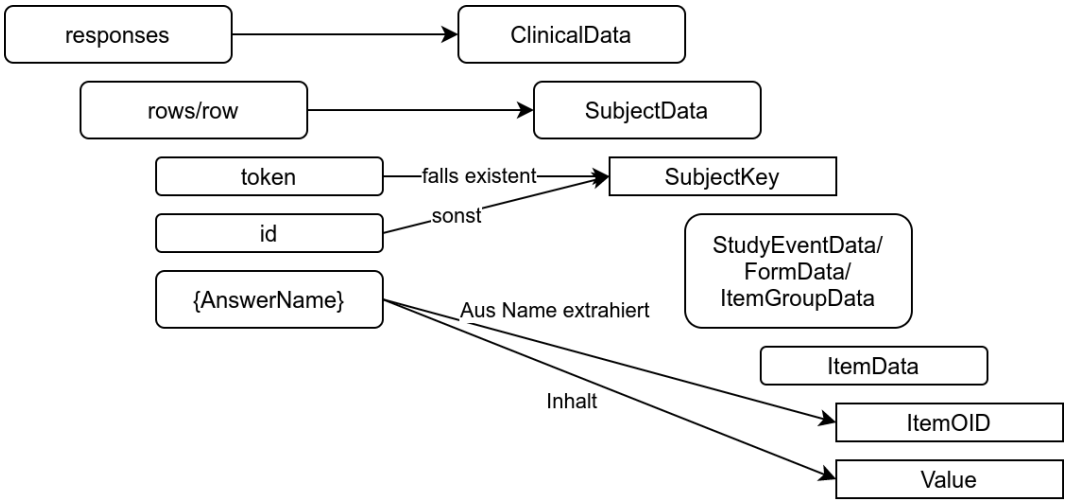


Abbildung 3.5.: Mapping des LSR-Formates

Als „*SubjectOID*“ wird ein Token, falls es ein solches gibt, verwendet. Das liegt daran, dass ein Token pro Teilnehmer eindeutig ist, während es pro Teilnehmer mehrere IDs in LimeSurvey geben kann. Gibt es kein Token, wird die ID aus LimeSurvey als ID on ODM genutzt. Ein passender „*RepeatKey*“ wird ebenfalls festgelegt, sodass verschiedene Ausfüllungen der Umfrage durch den gleichen Teilnehmer unterschieden werden können.

Als „*StudyOID*“ und „*MetaDataVersionOID*“ der „*ClinicalData*“ werden die entsprechenden OIDs referenziert. Aus jeder Reihe in LimeSurvey werden hier

die Daten eines Teilnehmers gemacht. Die ID wird übernommen, für die OIDs des „*StudyEvent's*“ und des Formulars werden die bereits erstellten Werte eingetragen. Als ID der Fragegruppe wird die „*gid*“ gesetzt. Als „*ItemOID*“ des „*ItemData*“ Elements wird ein Teil des Elementnamens genutzt, nämlich „{qid}{sqid}{ext}“, in „*Value*“ wird der Text des Elements eingetragen. Da die OIDs der Fragen vorher bereits so gewählt wurden, dass sie mit dieser Stuktur übereinstimmen, haben wir bereits funktionierende Referenzen, ohne weitere Umwandlungen vornehmen zu müssen.

3.2.7. Themes und Frageattribute

Die Themes werden nicht in ODM übernommen. Weitere Informationen gibt es in Abschnitt 4.2. Auch von den Frageattributen werden fast keine übernommen, da die meisten der visuellen Darstellung dienen.

3.3. Implementierung

3.3.1. Java

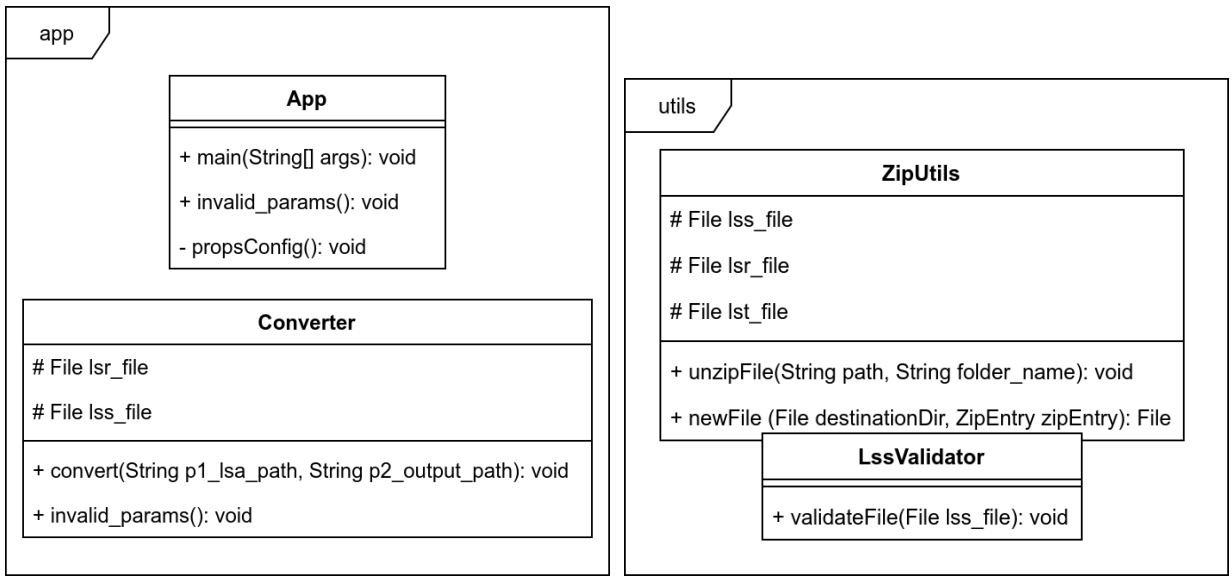
Nun stellt sich die Frage, welche Programmiersprache genutzt werden soll, um den Konverter zu implementieren. Java bietet sich dabei aus verschiedenen Gründen an. Einerseits gibt es viele Bibliotheken für Java, welche das Bearbeiten von XML stark vereinfachen, diese Bibliotheken sind auch sehr mächtig und können quasi alles, was man brauchen könnte, um die Aufgabe zu erfüllen. Darunter sind zum Beispiel Validatoren, die XSD 1.1 unterstützen und Parser, welche mit mehreren Attributen in einem Element umgehen können. Andererseits wird die Sprache bereits viel am IMI genutzt, der Konverter kann also sehr leicht als JAR in andere Programme eingebunden und von dort aus aufgerufen werden.

3.3.2. Programmaufbau

Das Programm trägt den Namen „lsa2odm“ und besteht aus vier Paketen, *app*, *utils*, *parser* und *writer*, wobei *parser* in zwei weitere Pakete aufgeteilt ist, *lss* und *lsr*. in *app* befindet sich die Klasse *App*, welche die *main*-Methode enthält. Diese erstellt eine Instanz des Konverters *LsaConverter*, auch die Methoden aus *ZipUtils* und *LssValidator* werden hier gebraucht.

3.3.3. Eingabe

Das Programm erwartet entweder eine oder zwei Eingaben. Der erste Parameter muss dabei der Pfad zum LSA-Archiv sein, der zweite, optionale Parameter ist dabei ein Pfad, wo die ODM-Datei am Ende geschrieben werden soll. Anschließend wird der erste Parameter mit dem Regulären Ausdruck



(a) Klassendiagramm für das Paket *app* **(b)** Klassendiagramm für das Paket *utils*

Abbildung 3.6.: Klassendiagramme für mehrere Pakete, welche am Anfang der Ausführung gebraucht werden

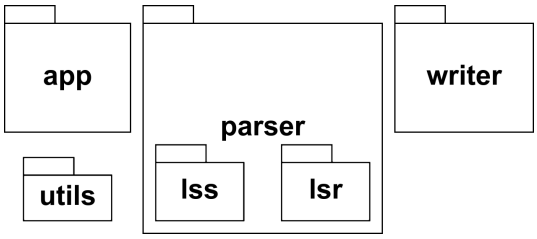


Abbildung 3.7.: Paketdiagramm für lsa2odm

„ˆ/?[.*/]*(.*?)\\.lsa$“ getestet. Ist der Parameter ein gültiger Pfad zu einer Eingabedatei, wird *unzipFile* auferufen. Die Funktion beinhaltet relativ generischen Code, welcher ein Zip-Archiv entpackt. Wenn die Funktion fertig ist, kann man auf alle benötigten Dateien zugreifen und es kann mit den nächsten Aufgaben weiter gemacht werden.

3.3.4. Properties

Es gibt eine Reihe an Strings, welche im Programm gesetzt werden, besonders die Namen der Dummy-Elemente. Aber auch Suffixe für manche Namen oder der Name der IMI-Syntax ist nicht festgelegt. Daher sollen all diese Dinge von außerhalb des Codes geändert werden können. Gibt es noch keine, wird eine Properties-Datei erstellt, welche all diese Informationen beinhaltet.

3.3.5. XSD-Validierung

Das in Abschnitt 3.1.7 erstellte Schema wird nun genutzt, um zu überprüfen, ob die Eingabedatei valide ist. Das Ergebnis der Prüfung wird dem Nutzer angezeigt, allerdings ist es nicht bindend, auch eine invalide Datei wird konvertiert. Das liegt vor allem daran, dass sich das Format schnell ändern kann, mehr dazu in Abschnitt 4.4. Da diese Änderungen aber oft klein sind, besteht eine hohe Chance, dass der Konverter den Großteil des Dokumentes dennoch verarbeiten kann. Der Anwender bekommt dennoch die Information, dass der Konverter potentiell nicht mit dieser LimeSurvey-Version funktioniert.

3.3.6. Parsing der LimeSurvey Struktur

Zuerst wird eine Instanz der Klasse *LssParser* mit der entsprechenden LSS-Datei erstellt. Die Klasse *Survey* dient zum speichern aller gesammelten Informationen, alle für die ODM-Datei benötigten Daten werden hier abgelegt, das Objekt wird später weitergegeben.

Zunächst werden die drei in Abschnitt 3.1.5 angesrochenen Elemente mit Metadaten der Studie in die *survey* übernommen. Dann werden die Fragegruppen übernommen, hierbei werden alle für das Mapping relevante Informationen gespeichert.

Parsing der Fragen

Als nächstes werden die Fragen in *survey* übernommen. Dabei werden diese bereits gemäß des Mappings umgewandelt. Das wird gemacht, um die Gesamtstruktur so früh wie möglich zu simplifizieren und nicht für jede Frage eine gesonderte Klasse erstellen zu müssen. Da sich viele Fragen im Aufbau stark ähneln, ist dies ein simpler und schneller Weg, das Ziel zu erreichen. Am Ende dieser Verarbeitung soll es noch vier Fragetypen geben:

T Eine Frage, auf die mit einem Freitext geantwortet werden kann

N Bei dieser Frage muss mit einer Zahl im float-Format geantwortet werden

I Bei dieser Frage muss mit einer Zahl im integer-Format geantwortet werden

A Bei dieser Frage muss aus einer vordifinierten Liste an Antwortmöglichkeiten gewählt werden

D Diese Frage hat ein Datum und eine Uhrzeit als Antwort

Zuerst wird eine Liste aller „*row*“ Elemente des Frage-Elements erstellt, durch welche im Anschluss iteriert wird. Die Klasse *Question* soll alle Informationen über eine Frage speichern, entsprechend wird diese genutzt, um die Informationen jeder Reihe zu speichern. Eine Liste der Daten ist in Abschnitt 3.3.6 zu sehen. Nicht in jedem Fall wird diese Frage auch zur Umfrage

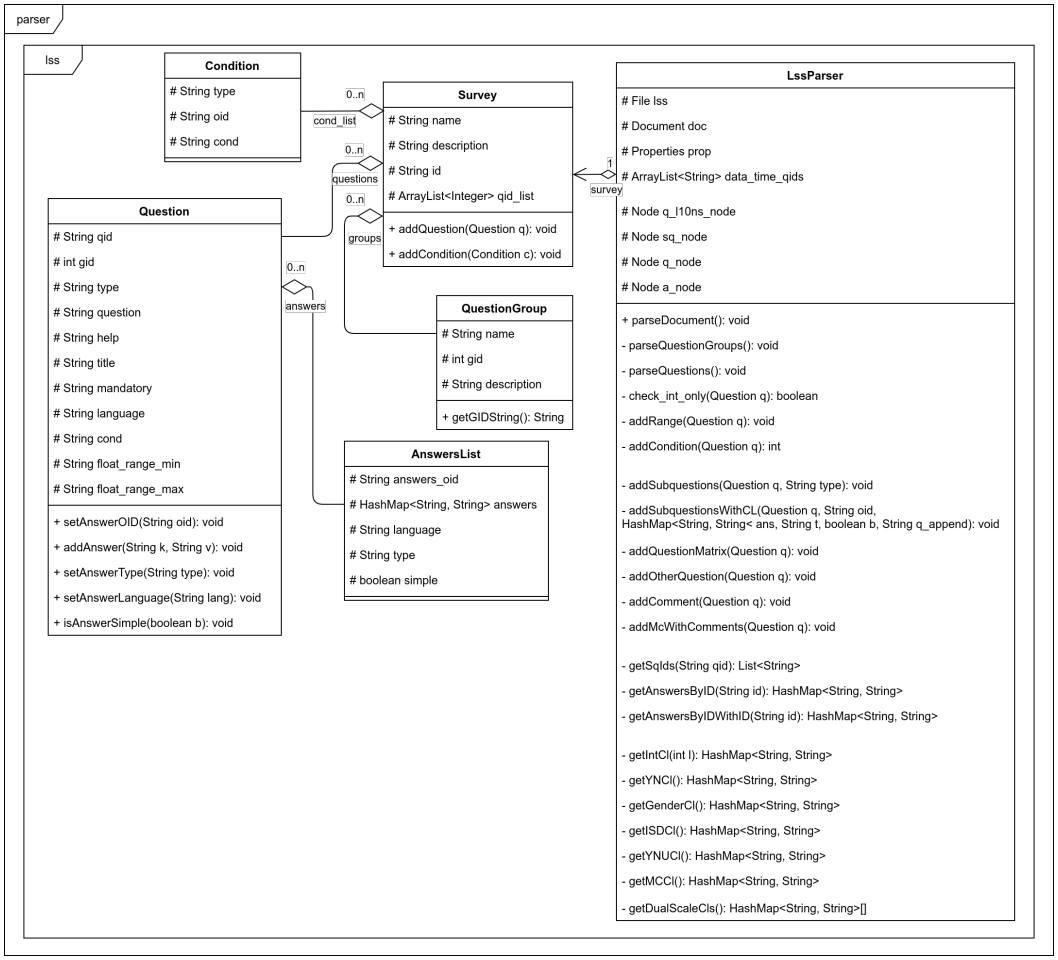


Abbildung 3.8.: Klassendiagramm für das Paket *lss*

hinzugefügt, wenn es sich zum Beispiel um eine Matrix-Frage handelt, wird diese Instanz nur zum Erstellen neuer Fragen genutzt. Dann werden potentiell existierende Bedingungen hinzugefügt, dieser Vorgang wird in Abschnitt 3.3.6 beschrieben.

Anschließend gibt es ein *switch*-Statement, welches einen passenden Weg zum Parsen der Frage abhängig vom Typ ausführt.

Textfragen

Für Textfragen wird hier nur der Typ geändert, sodass alle Arten von Textfragen „T“ als Typ haben.

Datum/Zeit

Bei Datum/Zeit-Fragen wird die „*qid*“ zu einer Liste hinzugefügt, welche später in Abschnitt 3.3.7 gebraucht wird.

Numerische Eingaben

Wenn es sich um eine Frage mit numerischer Antwort handelt, wird geprüft, ob die Antwort nur ganzzahlige Werte enthalten darf. In dem Fall wird der Typ auf „I“ gesetzt. Dann wird überprüft, ob es Minimal- und Maximalwerte für die Antwort gibt, wenn ja werden entsprechende Werte in der Frage-Klasse gesetzt.

Listen

Handelt es sich um eine Frage vom Typ „*Liste mit Kommentar*“, wird zuerst eine weitere Frage hinzugefügt, welche den Kommentar speichern soll. Dann wird mit der Behandlung für die anderen „*Listen*“-Fragen fortgefahren. Bei selbiger wird geschaut, ob die Option für *Anderes* aktiviert ist, wenn ja, wird auch dafür eine Frage hinzugefügt. Dann wird eine Liste mit Antwortmöglichkeiten an die eigentliche Frage angehangen und selbige wird hinzugefügt.

Restliche Einfachauswahl

Für die Fragetypen „*Fünf Punkte Auswahl*“, „*Ja/Nein*“ und „*Geschlecht*“ wird der Typ geändert und eine entsprechende Antwort-Liste mittels der dafür erstellten Hilfsfunktionen generiert.

Multiple Antworten

Wird nach Zahlen verlangt, werden erst potentielle Constraints ermittelt und hinzugefügt, dann wird der Typ angepasst, je nach Gleitkomma- oder Ganzzahl-Antworten. Dann wird bei beiden die Funktion *addSubquestions* aufgerufen. Diese sucht die IDs aller Subfragen und fügt dann entsprechende Kopien der Hauptfrage ein, Änderungen werden wie oben beschrieben durchgeführt.

Duale Skalen

Geht es um diesen Typ, werden erst beide Listen mit Antworten angelegt, dann wird die Funktion *addSubquestionsWithCL* zwei Mal aufgerufen, einmal mit jeder Skala. Beim ersten Mal wird „-0“ an die Frage-ID angehangen, beim zweiten Mal „-1“.

Matrizen (Text, Numerisch)

In beiden Fällen werden hier die Antwortmöglichkeiten auf den Achsen gesammelt, dann wird für jede Kombination eine neue Frage erstellt, der Inhalt wird angepasst.

Arrays

Für alle Array-Fragen wird die Funktion *addSubquestionWithCL* aufgerufen, die Liste an Antworten, deren Name un der Datentyp wird jeweils angepasst.

Mehrfachauswahl

Hier werden auch alle Subfragen mit einer Antwortliste eingefügt, die Liste enthält „Y“ für Ja und énquote für Nein. Dann wird noch eine potentielle *Anderes*-Frage eingefügt. Hat die Auswahl noch Kommentare, wird eine spezielle Funktion aufgerufen. Diese hat fsat gleichen Inhalt, allerdings wird noch ein Kommentar für jede Frage eingefügt. Eine passende Bedingung wird auch eingefügt, sodass die Kommentar-Frage nur angezeigt wird, wenn die Möglichkeit auch ausgewählt wurde.

Dabei wird die Frage-ID wie folgt aufgebaut: „{Parent_QID} + {Title(y-Axis)}_{Title(x-Axis)}“.

Parsing der Anzeige-Bedingungen

Da die Bedingungen innerhalb des LSS-Formates ohnehin als Elemente gespeichert werden und nicht als ExpressionScript, ergibt es mehr Sinn, diese nicht wieder in ExpressionScript umzuwandeln, sondern direkt in die Syntax des IMI. Die in LimeSurvey mit ExpressionScript formulierten Bedingungen geben an, wann eine Frage angezeigt werden soll. Die in ODM definierten Bedingungen müssen zu „True“ evaluieren, wenn eine Frage nicht angezeigt werden soll. Der fertige Ausruck muss am Ende also negiert werden. Dann werden alle Bedingungen rausgesucht, welche zur „*qid*“ gehören. Diese sollen nun durch ein logisches „Und“ verknüpft werden.

Die Frage aus „*cfieldname*“ wird mittels des regulären Ausdrucks

$$\hat{}\backslash\backslash d+X(\backslash\backslash d+)X(.+?)\$$

so verarbeitet, dass wir die darin enthaltene „*gid*“ und „*qid*“ der Frage enthalten. Mit diesen, der Dummy-ID für das *StudyEvent* und der Umfragen-ID wird dann ein Pfad gemäß der IMI-Syntax erstellt.

Handelt es sich nicht um einen Regulären Ausdruck, der als Operator genutzt wird, werden folgende drei Teile in dieser Reihenfolge aneinander gehangen:

$$\{\text{PATH}\}\ \{\text{OPERATOR}\}\ \{\text{VALUE}\}$$

wobei das Element „*method*“ den Operator enthält und „*value*“ den Wert. *PATH* ist der vorher erstellte Pfad.

Handelt es sich um einen regulären Ausdruck, wird das führende Leerzeichen entfernt und die Bedingung in folgender Form aufgeschrieben:

$$\text{MATCH}(\{\text{REGEX}\}, \{\text{PATH}\})$$

Soll eine Antwort leer bleiben, wird „NULL“ als Wert verwendet. Bei beidem handelt es sich nicht um einen Teil der IMI-Syntax, weiteres dazu in Abschnitt 4.7.

Abschließend wird der Bedingung eine OID der Form „{*qid*}{*ex*}“ gegeben, wobei „*ex*“ in der Properties-Datei festgelegt werden kann. Diese wird auch in der Frage eingetragen, damit man später eine Referenz auf die Bedingung hat.

3.3.7. Parsing der LimeSurvey Antworten

Die Klasse *LsrParser*, zusammen mit den Klassen *Response* und *Answer* wurde erstellt, um die Antworten zu verarbeiten. Zuerst wird in *createDocument* ein neuer SAXReader erstellt, welcher die LSR-Datei in eine Instanz der Klasse *Document* einliest. Selbige wird dann in *parseAnswers* weiterverwendet.

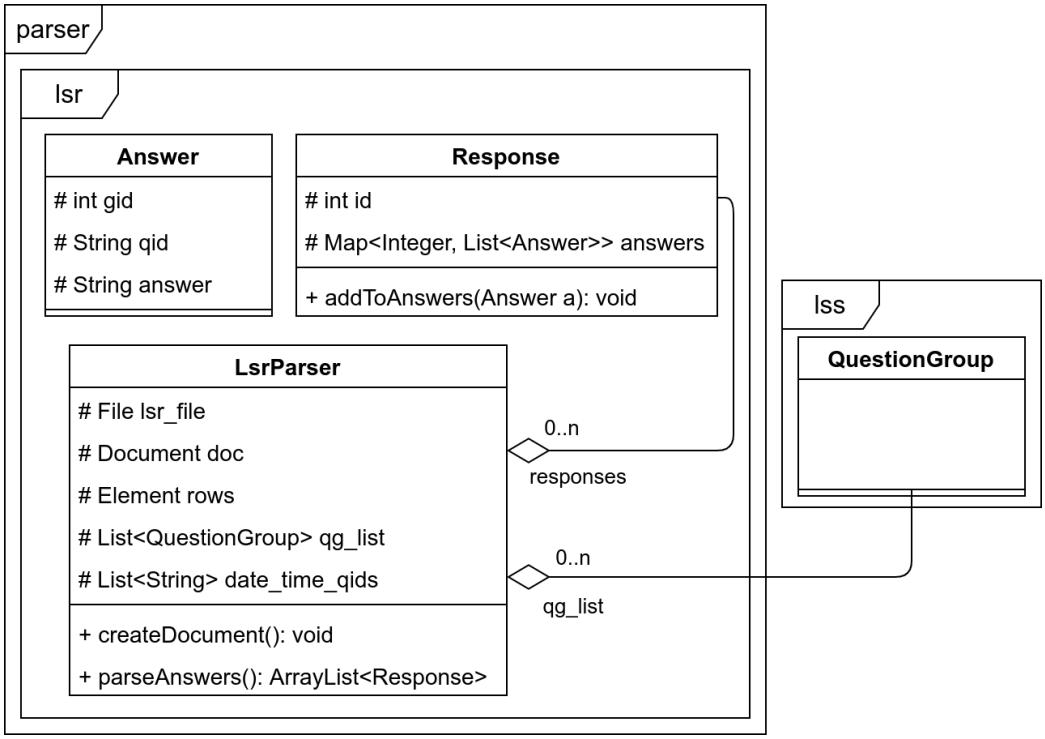


Abbildung 3.9.: Klassendiagramm des Paketes *lsr*

Alle für das Parsing relevanten Felder stehen in „*document/responses/rows*“. Dort gibt es eine Reihe pro Beantwortung des Fragebogens. Die Klasse *Response* dient zum Speichern einer Reihe, daher werden Instanzen erstellt, um alle Informationen der LSR-Datei zu übertragen. Als nächstes wird über alle Kind-Elemente der Reihe iteriert und es wird nach Antworten gesucht. Diese sind an der Struktur des Element-Namens zu erkennen, sie besitzen fast den gleichen Aufbau wie der Inhalt des Elements „*cfieldname*“ aus Abschnitt 3.3.6, allerdings gibt es hier zusätzlich noch einen führenden Unterstrich. Wir nutzen also

den regulären Ausdruck „^_\\d+X(\\d+)X(.+?)$“, um pasende Elemente zu finden und mittels der Capture Groups die *gid* und *qid* zu erhalten.

Dann wird geprüft, ob die Antwort leer ist, in dem Fall wird sie ignoriert. Ist sie nicht leer, wird geschaut, ob es sich um eine „*Datum/Zeit*“ Frage handelt, dazu wird die in Abschnitt 3.3.6 erstellte Liste genutzt. Dies ist notwendig, damit das Format dem Datentyp „xsd:dateTime“ entspricht, welcher später gefordert wird. Anschließend wird eine neue Instanz der Klasse *Answer* erstellt, diese soll die Antwort auf eine Frage sichern. Befüllt wird sie mit den vorher gewonnenen Informationen über die Frage.

Schließlich wird die Antwort mittels der Methode *addToAnswers* zu *answers* hinzugefügt. Diese Methode sortiert die Antwort dabei passend ein, sodass am Ende alle Antworten zu Fragen aus der gleichen Fragegruppe in einer Liste sind.

Dann wird die *Response* in die ensprechende Liste hinzugefügt. Als Teil des Objektes *survey* wird sie später an den *ODMWriter* weitergegeben, damit die Antworten dort übertragen werden können.

3.3.8. Ausgabe als ODM-Datei

Mittels der Klasse *ODMWriter* soll eine neue ODM-Datei erstellt werden. Der Konstruktor nimmt dabei die beim Parsing erstellte *Survey* entgegen. Die Methode *createODMFile* dient vor allem als Caller für andere Methoden, welche die eigentlichen Funktionen ausführen.

Studienstruktur

Zuerst erstellt *createODMRoot* das Wurzelelement „*ODM*“ mit allen benötigten Attributen. Als nächstes wird *addStudyData* aufgerufen. Hier werden die Elemente „*Study*“, sowie die globalen Variablen mittels der Dummy-Werte aus der Properties-Datei erstellt. Weiterhin werden die Elemente „*MetaDataVersion*“„*Protocol*“, „*StudyEventDef*“ und „*Form*“ erstellt, alle relevanten oder benötigten Attributewerte werden gesetzt. Die *meta_data_oid* besteht dabei aus dem Prefix in der Properties-Datei und der Studien-ID. Die Attributwerte für die „*StudyEventDef*“ stammen aus der Properties-Datei, das Formular wird gemäß des Mappings befüllt.

Fragetypen

Nun werden die Fragegruppen mittels *addQuestionGroups* hinzugefügt. Wie in Abschnitt 3.2.3 beschrieben wird aus jeder Fragegruppe eine „*ItemGroupDef*“. Im Formular wird die entsprechende Referenz auf die Fragegruppe eingefügt. Gibt es eine Beschreibung, wird sie in eine Beschreibung in der „*ItemGroupDef*“ eingefügt. Die Fragen brauchen später Referenzen in den Fragegruppen, daher werden Referenzen auf alle Fragegruppen gespeichert, sodass man später einfach Elemente hinzufügen kann.

Fragen

Danach werden alle Fragen in der Methode *addQuestions* eingefügt. Zuerst wird ein zweites Dokument mit einem Wurzelelement erstellt, um alle „*CodeList*“ Elemente zu speichern. Dies wird gemacht, da die Listen mit den Antworten hinter allen Fragen stehen sollen, eine Frage zwischen der letzten Frage und der ersten Antwortliste einzufügen ist allerdings nicht unbedingt einfach. Theoretisch kann man mittels der Methode *add* einen Index angeben und so ein Element an einer beliebigen Stelle einfügen. Da sich der Index hier allerdings mit jedem Einfügen einer neuen Frage verändert, ist es simpler, die Listen einfach erst auszulagern und später mit *appendContent* einfach alle Elemente zu übertragen.

Anschließend wird über den Typ entschieden, welche Methode zum Einfügen der Frage aufgerufen wird. Dabei gibt es hier nur noch die fünf Typen, auf welche alle Fragen in Abschnitt 3.3.6 reduziert wurden. Bei einer Array-Frage wird *addQuestionWithCL* aufgerufen, was die Frage und die Antwortliste einfügt. In allen anderen Fällen wird *addQuestion* aufgerufen, der übergebene Datentyp variiert dabei entsprechend. Zuletzt wreden alle Antwortlisten übertragen.

Klinische Daten

Zuerst muss das Element für Klinische Daten im Wurzelelement erstellt werden. Anschließend ist *createODMFile* fertig, die Antworten werden jetzt durch eine öffentliche Methode hinzugefügt. Diese wird im LSR-Parser aufgerufen, die Rationale dafür wurde bereits in Abschnitt 3.3.7 erklärt.

Ausgabe

Mittels *writeFile* wird nun das vorher erstellte Dokument geschrieben. Der Name der Ausgabe-Datei ist „{Survey-ID}.xml“, sie wird am Ausgabepfad geschrieben, falls einer angegeben wurde. Zusätzlich wird das Dokument formatiert, sodass es auch von Hand lesbar ist. Dazu reicht in *dom4j*

$$OutputFormat format = OutputFormat.createPrettyPrint();$$

Das erstellte Format kann dann an den *XMLWriter* gegeben werden.

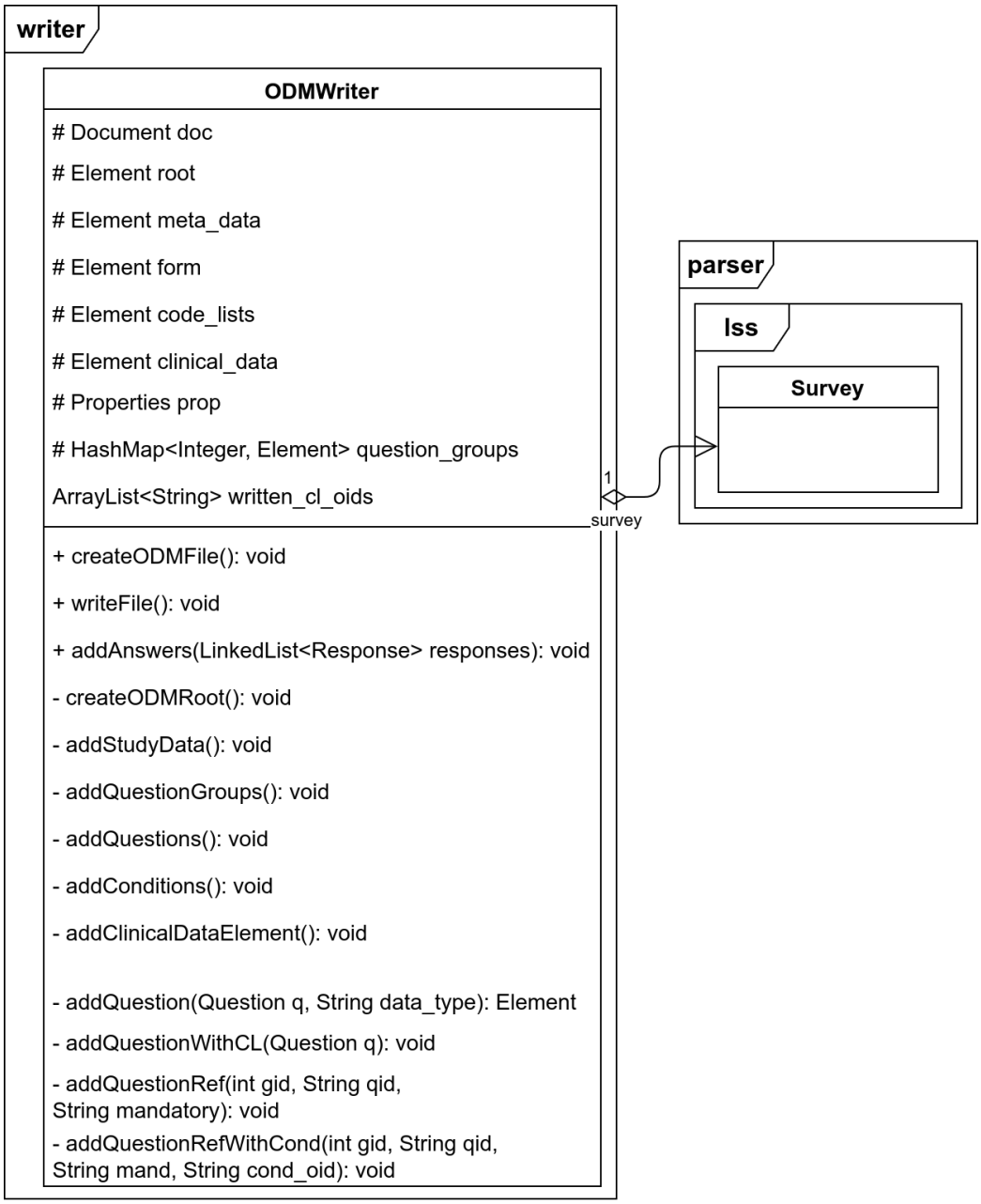


Abbildung 3.10.: Klassendiagramm des Pakets writer

4. Diskussion

4.1. Weglassen von Fragetypen

Auch wenn mit diesem Konverter eine vollständige Umsetzung der LimeSurvey Archiv-Daten angestrebt wird, so wurden doch einige Fragetypen bewusst nicht umgesetzt. Im Folgenden soll erläutert werden, welche Fragetypen nicht konvertiert wurden, wie diese Fragetypen hätten umgesetzt werden können und warum die Entscheidung getroffen wurde, dies nicht zu tun.

4.1.1. Datei-Upload

Der Fragetyp „*Datei-Upload*“ kann genutzt werden, um den Nutzer auf eine Frage mit einer Datei antworten zu lassen. In ODM hätte man diese Datei einbinden können, indem man sie in „*hexBinary*“ umwandelt, ein Datentyp in ODM, welcher Stream-Daten in einem hexacodierten Binärformat sammelt. Trotzdem wurde dieser Fragentyp im Konverter nicht umgesetzt. Das liegt zum einen daran, dass die hochgeladenen Dateien nicht Teil des Archives sind (siehe Abschnitt 2.1.5) und andererseits daran, dass die Praktikabilität dieses Vorgehens eher fragwürdig ist. Unter anderem wird eine Rekonstruktion zur Originaldatei schwer, da es zum Beispiel keine Informationen über den ursprünglichen Dateityp gibt, auch wird die XMl-Datei nur noch sehr unangenehm von Hand lesbar, wenn man diese Stream-Daten in Antworten einbinden würde. Auch wird der Fragetyp nicht häufig genutzt, was die Umsetzung noch unattraktiver macht.

4.1.2. Browser-Detection, Language-Switch

Beide Fragetypen sammeln Informationen über den Benutzer, man könnte also argumentieren, es sei sinnvoll, beides als Fragen zu übernehmen. Technisch wäre das ebenfalls kein Problem, man kann einfach einen festen Fragetext vorformulieren und die gesammelten Informationen als Antwort eintragen.

Das wäre allerdings irreführend, da dies impliziert, der Nutzer habe ausdrücklich auf eine Frage geantwortet, wenn er in Wirklichkeit eventuell nicht einmal wusste, dass unter anderem Informationen über seinen Brower aufgenommen wurden. Auch eine Frage zur Sprache hätte weniger Sinn, gerade weil es auch hier eigentlich keine Antwort auf eine Frage war, sondern eher eine Einstellung in der Umfrage. Das Ziel einer Umfrage ist es ja eher, die Meinung oder Ansichten eines Teilnehmers einzuholen, Metainformationen wie die Sprache

gehören da eher nicht zu. Keine Frage zur Sprache zu erstellen ist allerdings eher eine Designentscheidung, man könnte hier durchaus auch anderes machen und es wäre ein Informationsgewinn, zumal der Nutzer die Sprache ja auch wirklich selber eingestellt hat.

4.1.3. Text-Display

Dieser „Fragetyp“ stellt nur einen Text dar. Es werden keine Informationen gesammelt. Da man den Inhalt des Textes auch nicht automatisch, irgendeiner Frage oder Fragegruppe als Beschreibung zum Beispiel, zuordnen kann, wird er gar nicht übernommen.

4.1.4. Gleichung

Da eine Gleichung keine neuen Informationen erfragt, sondern nur aus vorherigen Antworten einen Wert berechnet und diesen anzeigt, ist es keine Frage, sondern nur eine zusätzliche Information zum Darstellen. Diese werden entsprechend der Philosophie der ganzen Arbeit nicht übernommen.

4.2. Visuelle Darstellung der Fragen

In LimeSurvey gibt es zahlreiche Möglichkeiten, die Darstellung seiner Umfrage zu verändern. Darunter sind Themes für die ganze Umfrage, sowie Attribute für einzelne Fragen. Teils kann man auch ganze CSS-Klassen angeben, welche das Aussehen der Umfrage bestimmen. All diese Angaben sind selbstverständlich auch Teil des LSA-Archives, werden vom Konverter allerdings nicht übernommen.

Das ist prinzipiell nicht optimal, da die Art und Weise, wie eine Frage dargestellt wird, auch eine Auswirkung auf die Antwort haben kann. Ein bestimmtes Antwortverhalten lässt sich also potentiell nicht mehr nachvollziehen, da die Darstellung verloren gegangen ist.

Dennoch ergibt es Sinn, die Informationen nicht zu übenehmen. Erstens hat der ODM-Standard keinen Weg, irgendwelche Attribute zur visuellen Darstellung zu speichern. Eine Übernahme erfordert also ein selbst definiertes Prinzip, was wiederum von anderen Programmen, welche ODM Dateien nutzen, nicht verstanden werden kann. Zweitens handelt es sich hierbei nicht um Informationen, welche von den Teilnehmern gewonnen wurden, das ist wohl auch einer der Gründe, warum es in ODM keinen Standard für visuelles gibt. Auch spielt die Darstellung zwar eine Rolle, jedoch keine übergeordnete, man verliert durch das Auslassen der Informationen keinen wesentlichen Teil der in LimeSurvey gewonnenen Informationen. Eigenschaften wie die Formulierung der Frage sind hier relevanter und diese wird übernommen.

4.2.1. Timings

Auch die Timings sind vorwiegend ein Weg, die Art und Weise zu beeinflussen, wie eine Frage oder Informationen über diese beim Teilnehmer dargestellt werden. Eigenschaften wie die Warnungen zu bestimmten Zeiten oder die Limitierung der Antwortzeit haben zwar einen Einfluss auf die Länge der Antwort zum Beispiel, aber dennoch sind sie nicht wirklich Teil der Frage im klassischen Sinn.

Da ODM auch hier keinen Standard zum Darstellen der Informationen besitzt, werden Timings ebenfalls nicht übernommen.

4.3. Formatierung

Da LimeSurvey erheblich mehr Möglichkeiten hat, die Darstellung von Fragen zu beeinflussen, musste sich im Mapping überlegt werden, wie man diese Fragen so formatieren kann, dass sie in ODM übernehmbar sind. Mittels der „*Presentation*“ kann eine Darstellung von Fragen definiert werden. Da hier allerdings auch keine standardisierte Form gibt, soll diese Möglichkeit zuerst nicht genutzt werden. Entsprechende Syntax könnte nämlich von keinem bestehenden Programm erkannt werden.

4.3.1. Fragen mit Bildern im Fragetext

Bei dem Bild der „*Image-Select-List*“ besteht das Problem, dass dieses nur mittels Link zum Bild auf dem LimeSurvey-Server eingebettet ist. Beim Übertragen der Fragen ist somit auch nur dieser Link enthalten.

4.3.2. Arrays

Arrays werden in Einfachauswahl-Fragen auseinander gezogen. Die originale Struktur geht damit verloren, das ist problematisch. Unter anderem liegt das daran, dass sich das Antwortverhalten bei Umfrage-Teilnehmern unterscheiden kann, wenn die Fragen anders gestellt werden.

Aus eigener Erfahrung beeinflussen die anderen Antworten im Array die weiteren Antworten, zum Beispiel indem gewisse Muster beim Antworten befolgt werden. Auch haben die bisher gegebenen Antworten oft einen Einfluss, wenn man sich zwischen zwei Antwortmöglichkeiten entscheiden muss. Dieser Effekt ist so allerdings nicht wissenschaftlich nachgewiesen.

Auf der anderen Seite ist dieser Weg - Unter Beachtung der Limitierungen von ODM - die Möglichkeit mit den geringsten Nebeneffekten. Fragetexte, Hilfen und Antwortmöglichkeiten werden korrekt übernommen. Auch ist nicht klar, inwiefern der oben beschriebene Effekt überhaupt regelmäßig auftritt.

4.3.3. Multiple Choice Fragen

Hier werden aus einer Multiple-Choice-Frage mehrere Ja/Nein-Fragen, das hat den Nebeneffekt, dass der Teilnehmer sich über einzelne Antwortmöglichkeiten potentiell weitaus mehr Gedanken macht, als in der ursprünglichen Darstellung.

Wenn man z.B. aus der Frage: „Nennen sie ihre Lieblingsfarben: Rot, Grün, Blau, Orange“ die Frage „Ist Orange ihre Lieblingsfarbe?“ macht, wird sich der Teilnehmer bei letzterem weit genauer überlegen, ob er Orange wirklich so gerne mag, während man in der ersten Darstellung weit eher dazu geneigt ist, das mit seiner eigentlichen Lieblingsfarbe, Grün zum Beispiel, anzukreuzen, weil die Farbe ja auch ganz nett ist und man eh mehrere nehmen kann.

Dieses Verhalten kann dazu führen, dass man weniger Optionen anklickt (Hier zum Beispiel, hat man ja bereits eine Lieblingsfarbe angegeben, wieso noch eine zweite anwählen?) und dass man weniger Optionen überlesen kann, gerade bei Fragen mit vielen Antwortmöglichkeiten ist so die Gefahr geringer, eine Option zu überspringen.

Mit den Limitierungen von ODM ist dies allerdings immer noch ein besserer Weg, als die Frage z.B. pro Antwortmöglichkeit einmal als Einfachauswahl einzufügen. Das würde wesentlich mehr Wiederholung bedeuten, die Auswertung unnötig kompliziert machen und wäre eine aus Teilnehmerperspektive sehr seltsame Darstellung.

4.4. Versionsabhängigkeit

In dieser Arbeit wurde mit der LimeSurvey-Datenbank-Version *443* gearbeitet. Die Art und Weise, wie Fragen gespeichert werden, ändert sich allerdings potentiell immer wieder. So wurde in Version *441* die Matrix noch anders gespeichert, es gab einen anderen Weg, zu unterscheiden, welche Subfrage auf der X-Achse und welche Subfrage auf der Y-Achse liegt. Derartige Änderungen würden den Konverter natürlich kaputt machen, dementsprechend kann es notwendig werden, versionsabhängige Arbeitsschritte in den Code einzubauen.

Bisher wurde das nicht gemacht, unter anderem deshalb, weil es schwer ist, an LSA-Archive in alten Versionen zu kommen, hier müsste man potentiell weitere LimeSurvey-Instanzen aufsetzen. In dieser Hinsicht wäre es sicherlich sinnvoll, den Code noch besser und eindeutiger zu kommentieren, als es ohnehin geboten ist, sodass jeder später schnell und einfach kleinere Änderungen in Abhängigkeit der Datenbank-Version einbauen kann.

Auch muss man sich überlegen, für welche Versionen der Konverter tatsächlich funktionieren soll, bei zu vielen Versionen und Änderungen wird der Code potentiell sehr unübersichtlich.

4.5. XSD Defintion

4.5.1. Element-Inhalte

Auffällig war, dass in der LSS-Datei nur CDATA-Werte als Text verwendet werden. So ist niemals klar, welcher Datentyp genau nun in ein bestimmtes Feld gehört. Das Schema erzwingt hier teils genauere Datentypen, wenn der Inhalt offensichtlich ist, allerdings ist dies nicht immer möglich. Trotzdem wird so nicht nur ein Maß an struktureller Korrektheit sondern auch an inhaltlicher Korrektheit erzwungen. Prinzipiell sollten diese Datentypen einer aus LimeSurvey exportierten Datei nie im Wege stehen, daher ist die Einführung dieser kein Problem.

4.5.2. Design

Auch das Design der hardgecodeten Elemente ist nicht optimal, da so in der Zukunft zum LSS-Format hinzugefügte Elemente als invalide erkannt werden. Man könnte sicherlich einen dynamischeren Weg kreieren, indem man die Liste an möglichen Elementen in dem „*fields*“-Element und Features von XSD 1.1 nutzt. Allerdings widerspricht das der Art und Weise, wie XSD verwendet werden sollte, das Festhalten der genauen Elementnamen ist dort vorgesehen. Auch ist es kein großes Problem, da der Konverter diese Felder dann sowieso nicht kennt, so wird man ebenfalls vor potentiellen Inkompatibilitäten gewarnt.

4.6. Implementierung

4.6.1. Java

Es gibt eine Vielzahl an Gründen, warum Java als Sprache für dieses Projekt gewählt wurde. Einerseits soll der Konverter später am IMI eingesetzt werden, deren Systeme basieren größtenteils auf Java. Weiterhin ist Java dank der JVM unabhängig von Betriebsystem und es gibt eine Menge an Bibliotheken, welche das Programmieren vereinfachen. Mit *log4j* bekommt man einen einfachen Überblick über alle Nachrichten des Programms, weiterhin spart man sich dank *lombok* das Schreiben von Gettern, Settern und vielen Konstruktoren. Die Sprache ist schnell und es gibt sehr viele mächtige Werkzeuge, um mit XML arbeiten zu können.

Natürlich gibt es noch weitere Sprachen, mit welchen man den Job hätte erledigen können, wie Python oder Rust. Diese haben allerdings alle eigene Nachteile, Python ist oft langsamer als Java und die Bibliotheken in Rust haben teils gravierende Mängel, wie eine Unfähigkeit mit mehreren Attributen in einem Element umgehen zu können. Sie hätten aber auch Vorteile, mit Features wie optionalen Parametern hätten einige Funktionen übersichtlicher und kompakter gestaltet werden können.

4.6.2. Switch-Statement

Man könnte sagen, das *switch*-Statement, welches Fragen anhand des Typs abhandelt, sei nicht der optimale Weg, um das Problem in einer objektorientierten Sprache zu lösen. Auch ist es unübersichtlich, wenn man wie hier fast 30 Fälle im gleichen *switch* hat. Allerdings erfüllt es die benötigten Anforderungen perfekt:

Manche Fragen benötigen die selbe Verarbeitung und teils sind die Verarbeitungsschritte einer Frage Teil der Verarbeitungsschritte einer anderen Frage. Durch *switch* wird das Problem effizient behandelt.

4.6.3. JAXB

Mit JAXB kann man aus XML-Elementen automatisch Klassen erstellen. Das ist ein schneller und wenig aufwändiger Weg, die gesamte Eingabedatei in Java zu übernehmen. Allerdings ist das hier nicht unbedingt zielführend, da die LSS-Struktur ja erheblich komplexer ist, als das auszugebende ODM. Daher war es Ziel, die LSS-Struktur so schnell wie möglich zu vereinfachen, das wurde direkt beim Einlesen des Archives gemacht.

4.7. Erweiterung der IMI-Syntax

In der IMI-Syntax wird keine Möglichkeit vorgestellt, reguläre Ausdrücke zu evaluieren. Daher wird in dieser Arbeit vorgeschlagen, eine Funktion

MATCH(REGEX, PATH)

einzuführen, welche reguläre Ausdrücke mit beginnendem und endendem Schrägstrich und potentiell Flags hinter dem endenden Schrägstrich entgegennimmt und prüft, ob die Antwort dem Muster entspricht. Mittels einer Custom-Funktion ließe sich so eine Syntax in *expr-eval* und *EvalEx* integrieren.

Auch wird keine Möglichkeit geboten, die Abwesenheit einer Antwort zu überprüfen. Dafür wird ein Vergleich mit „NULL“ vorgeschlagen. Das wird allerdings nur von *EvalEx* unterstützt, nicht durch *expr-eval*.

4.8. Verwandte Arbeiten

Es gibt eine ganze Reihe an Konvertern, welche entweder von ODM zu einem anderen Format oder von einem anderen Format zu ODM konvertieren können. Viele dieser wurden von Mitarbeitern des IMI geschrieben, entsprechende Veröffentlichungen gibt es unter ihren Publikationen.

Zuerst gibt es einen Konverter von ODM zu Simply.MDR [9] und einen Konverter zwischen ODM und openEHR[10]. Beide mappen Metadaten von ODM

in das jeweils andere Format und nutzen XSLT, um die Konvertierung zu realisieren. Das ist der simpelste Weg, da beide Formate in XML geschrieben sind und die Formate große Ähnlichkeiten besitzen. Entsprechend mussten auch nur wenige Elemente weggelassen werden, wie die Protokolle und Studienevents in openEHR.

Dann gibt es noch zwei weitere Konverter, welche Java nutzen. Der erste wandelt ODM Metadaten in FHIR [11] um, der zweite wandelt die Ergebnisse einer Studie, die „Study of Health in Pomerania“ (SHIP) zu ODM um[12]. Beide sind in ihrer Arbeit erfolgreich gewesen, auch wenn das SHIP-Format eine nur vage ähnliche Struktur zu ODM hatte. Der FHIR-Konverter nutzt aus dem ODM-XML-Schema gewonnene Klassen zum Zwischenspeichern der Daten.

Weitere zwei Konverter wurden nur in Theorie implementiert, es gibt also nur ein Mapping. Dazu zählt die Konvertierung zwischen ODM und EN 13606 EHR[13], was mit kleinen Limitierungen möglich war, sowie die Konvertierung von ODM zu OpenClinica[14]. OpenClinica besitzt eine sehr ähnliche Struktur, ist aber kein XML-Format, sondern ein Excel-Template. Auch hier werden nur Metadaten gemappt.

Zuletzt wurde noch ein Konverter zwischen der „Clinical Document Architecture“ (CDA) und ODM in *R* implementiert[15]. Dieser ist ebenfalls Open-Source, unter der GPL-Lizenz.

Weiterhin gibt es noch ein Werkzeug zur Erstellung von Konvertern, die „ODMToolBox“[16]. Dies ist ein in Java geschriebenes Programm, welches mittels Spring und einer REST-API Möglichkeiten bereitstellt, eigene Konverter zu bauen. Dabei wird bevorzugt ein direktes Mapping verwendet, ist das nicht möglich, wird nach Workarounds gesucht.

5. Fazit

Eine Umwandlung von LimeSurvey-Archiven in das Operational Data Model ist grundsätzlich mit kleineren Einschränkungen möglich. Das in dieser Arbeit erstellte Mapping wurde implementiert und seine Praxistauglichkeit damit bewiesen.

Das Arbeiten mit dem LSA-Format, was außerhalb von LimeSurvey durch mangelnde Dokumentation früher anstrengend gewesen sein muss, wurde nun erleichtert. Einerseits kann jeder nun sehr schnell ein Verständnis für den Format-Aufbau erlangen, da selbiger in dieser Arbeit präzise analysiert wurde, andererseits kann jeder den implementierten Konverter nutzen, um seine Daten direkt im ODM-Format vorliegen zu haben.

Da ODM eigentlich dazu gedacht ist, wesentlich komplexere Studien-Strukturen darzustellen, bleiben einige Features hier ungenutzt, gleichzeitig besitzt LimeSurvey erheblich mehr Möglichkeiten, die visuelle Darstellung und andere Eigenschaften einer Frage zu beeinflussen. Das macht es notwendig, vieles entweder zu simplifizieren oder wegzulassen, wenn man eine Konvertierung vornehmen möchte.

Dennoch ist es in dieser Arbeit gelungen, alle relevanten Metadaten und klinischen Daten aus LimeSurvey zu extrahieren und ohne einen größeren Verlust an Informationen in ODM einzupflegen. Das hier erstellte Formular kann nun entweder für sich genutzt werden oder in anderen ODM-Dateien mit größeren Studien eingebunden werden.

Dadurch wird die Interoperabilität erheblich erhöht und der Austausch von Daten zwischen verschiedenen Systemen stark simplifiziert. Dieser Konverter reiht sich in eine längere Reihe an Konvertern ein, welche andere Formate zum Speichern medizinischer und klinischer Daten entweder zu oder von ODM umwandeln. Durch dieses sich ständig erweiternde Netzwerk wird es in Zukunft immer einfacher, ein beliebiges Datenformat in ein anderes umzuwandeln, indem man einen Mittelweg über ODM geht. Auch wird das Speichern der Daten immer einfacher, wenn alles in einem einzigen Format gespeichert werden kann. Dafür ist unter anderem auch die Qualität des Konverters entscheidend, da der Datenverlust beim Konvertieren minimal sein soll.

Die bisherige Implementierung macht hier bereits einen guten Job, allerdings kann man, mit genügend Zeit und wenn man die Notwendigkeit sieht, noch weitere Features hinzufügen:

- Mehrere Umfragen in einer LSS-Datei unterstützen

- Fragen in verschiedenen Sprachen in einem ItemDef Element unterbringen
- Unterstützung für mehr Formate bei „*Datum/Zeit-Fragen*“
- Übernahme von regulären Ausdrücken zur Validierung der Antwort-Struktur als „*RangeCheck*“
- Übernehmen, wie viele Ziffern eine Zahl bei einer Antwort lang sein darf

A. Vollständiger Aufbau des LSS-Formates

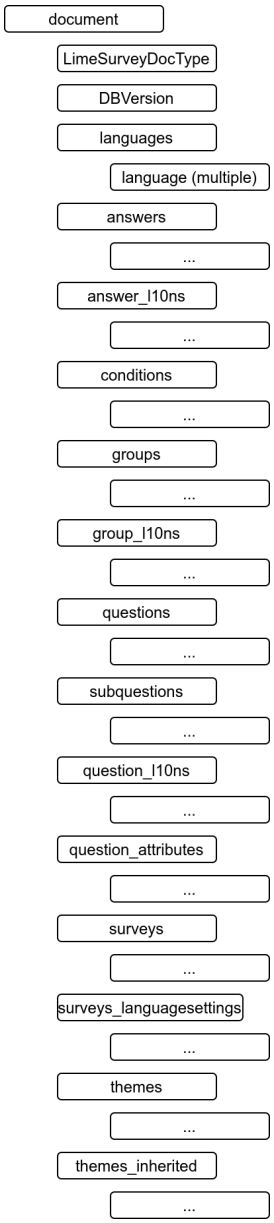
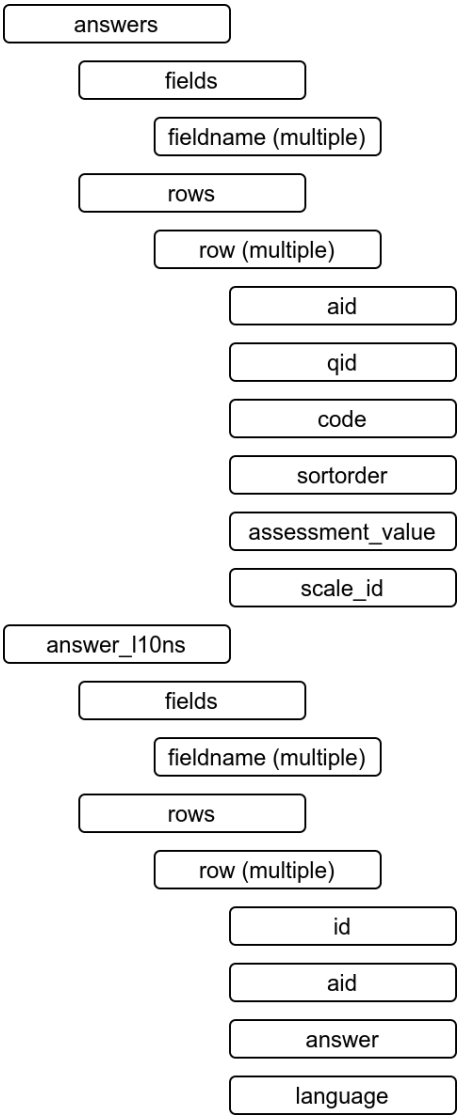
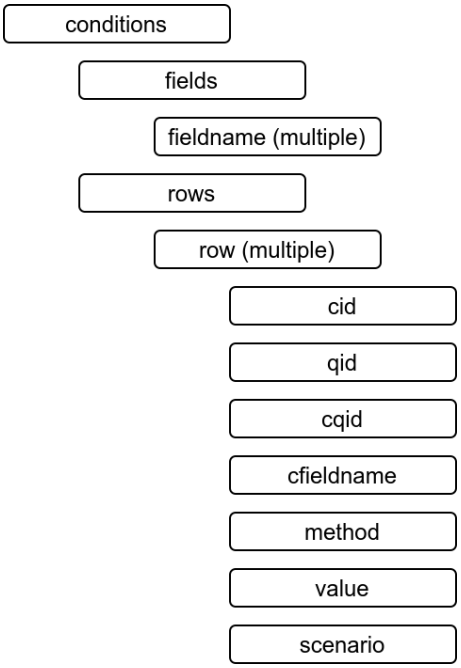


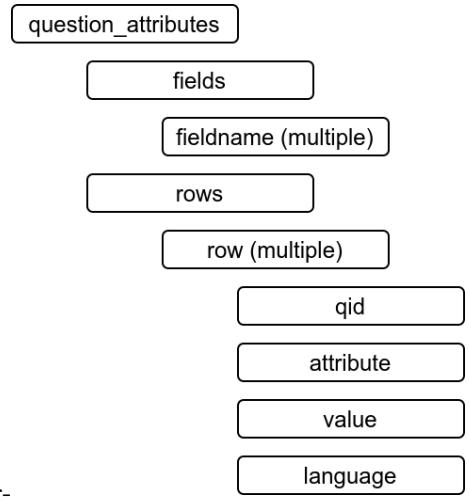
Abbildung A.1.: Aufbau des Wurzel-Elements „*document*“



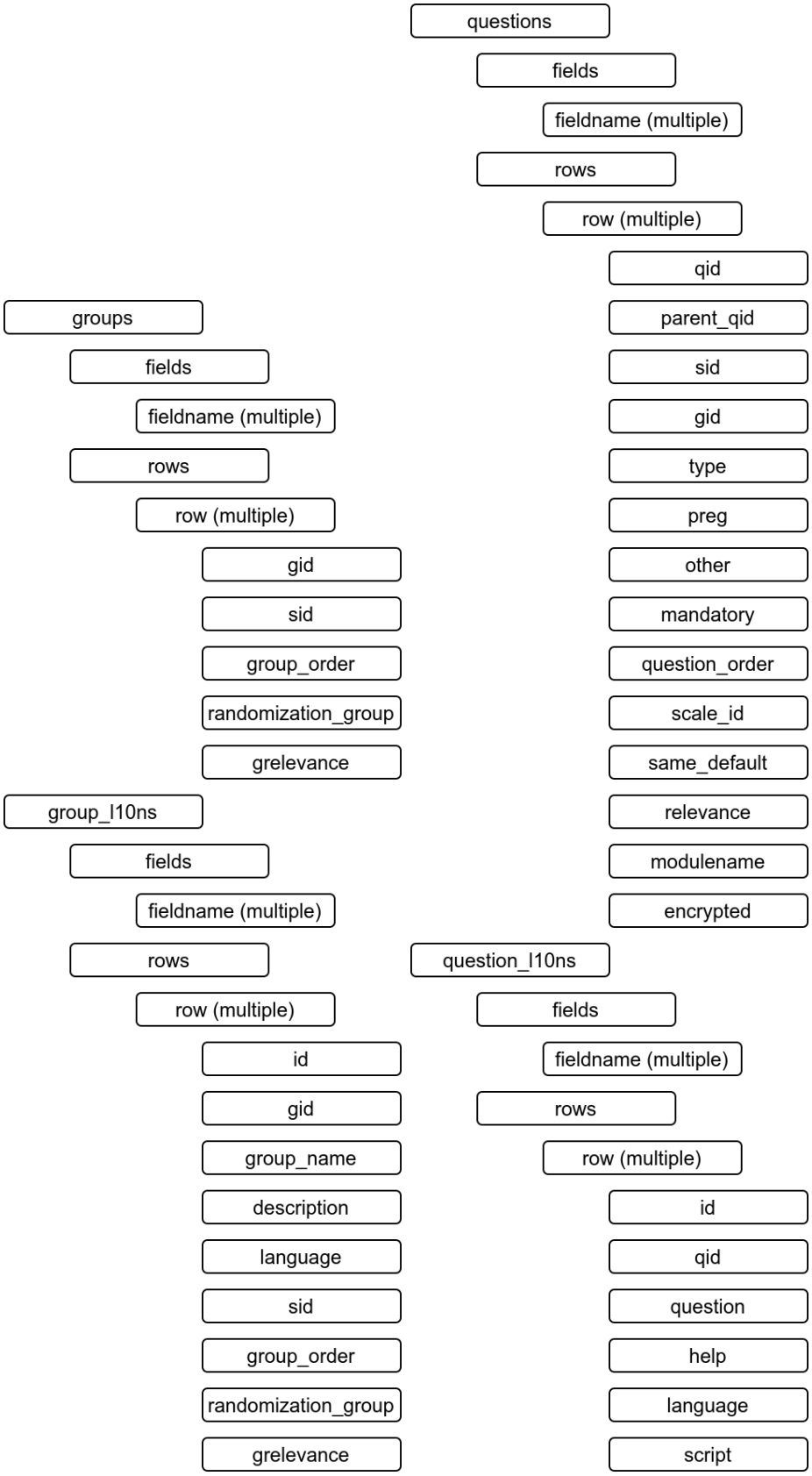
(a) Aufbau der Elemente für die Antworten



(b) Das Element für die Bedingungen



(c) Die Frageattribute im LSS-Format



(a) Aufbau von Fragegruppen **(b)** Aufbau der Frageelemente

Abbildung A.3.: Diagramme für Fragegruppen und Fragen

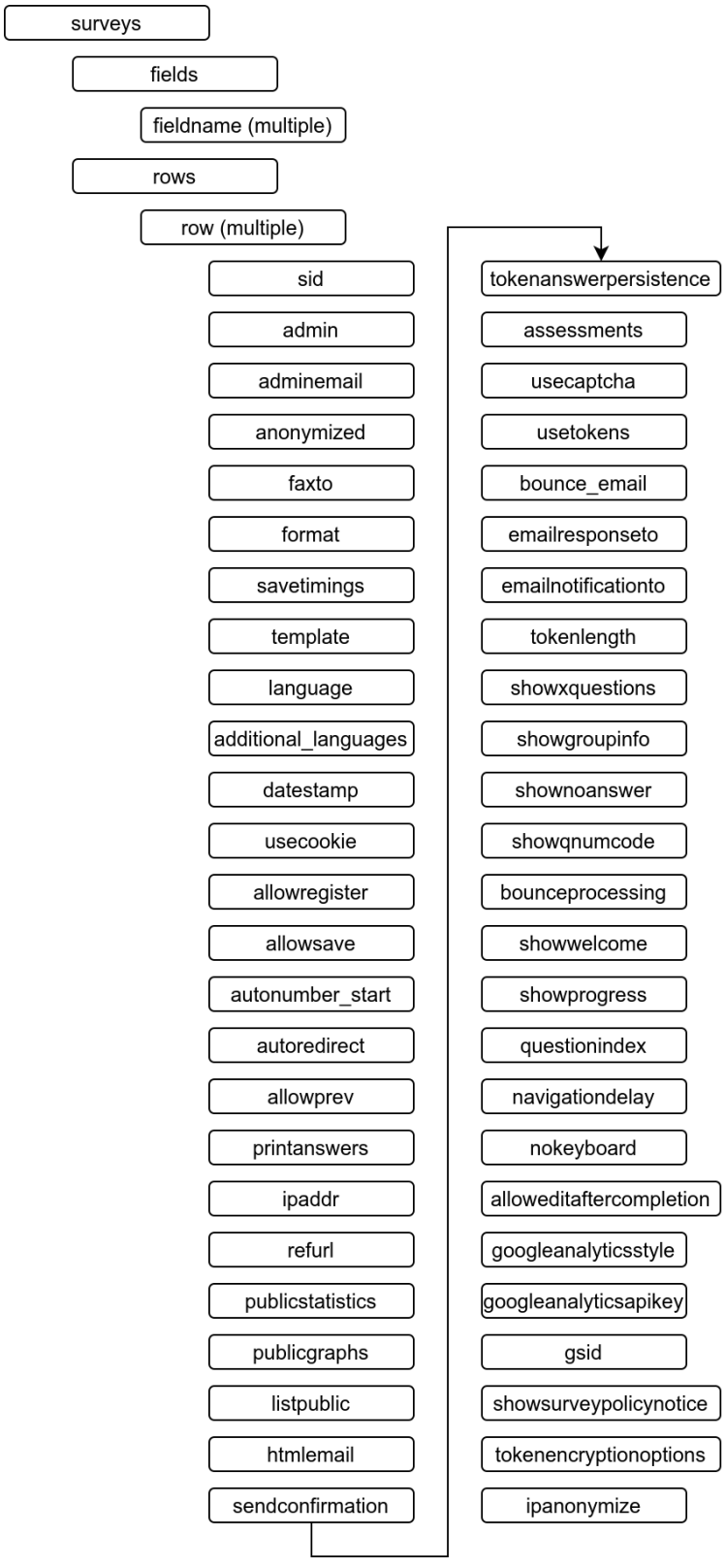


Abbildung A.4.: Aufbau der Metadaten einer Umfrage

(a) Aufbau von weiteren Metadaten der Umfrage

(b) Aufbau eines Themes

Abbildung A.5.: Diagramme für Metadaten der Umfrage und des Designs

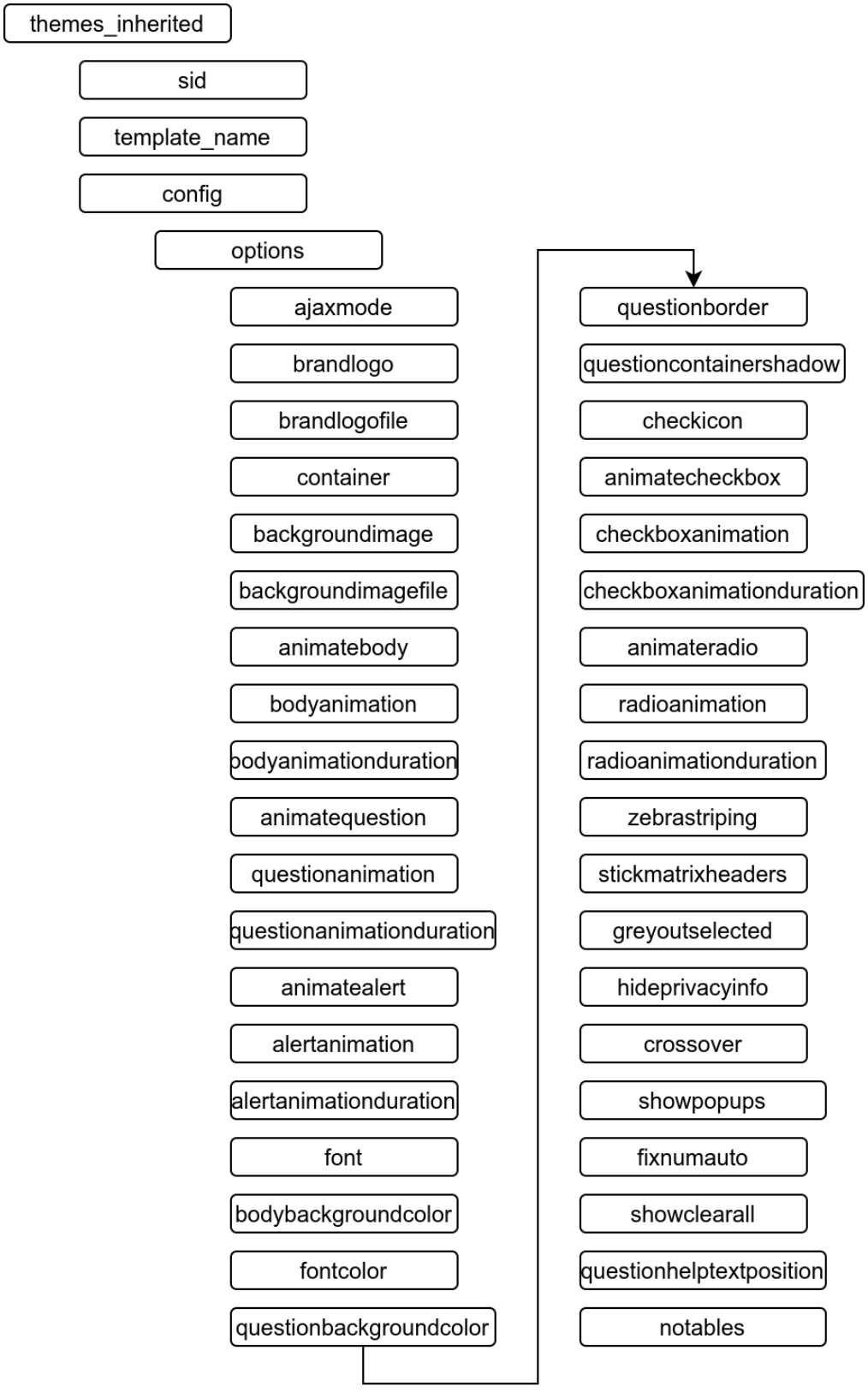


Abbildung A.6.: Aufbau der geerbten Designs

Literatur

[1] *Bessere Therapien dank Medizininformatik*. 10. Juli 2017. URL: https://www.bmbf.de/de/bessere-therapien-dank-medizininformatik-4473.html.

[2] M Dougas. *Medical Data Models (MDM) Portal*. 2021. URL: https://medical-data-models.org/.

[3] *LimeSurvey*. 28. Apr. 2021. URL: https://www.limesurvey.org/de.

[4] *Docker-Hub: acspri/limesurvey*. 2021. URL: https://hub.docker.com/r/acspri/limesurvey.

[5] *LimeSurvey Manual*. 28. Apr. 2021. URL: https://manual.limesurvey.org/.

[6] S Semler. *CDISC – ein neuer IT-Standard (nicht nur)für die medizinische Forschung*. 2006. URL: http://www.medizin-edv.de/ARCHIV/cdisc_medizinische_Forschung.pdf.

[7] *dom4j Documentation*. URL: https://dom4j.github.io/javadoc/2.1.3/.

[8] *Regex101*. URL: https://regex101.com/.

[9] A Kock-Schoppenhauer u. a. „Compatibility Between Metadata Standards: Import Pipeline of CDISC ODM to the Samply.MDR“. In: (2018), S. 221–225. DOI: 10.3233/978-1-61499-852-5-221. URL: https://ebooks.iospress.nl/publication/48786.

[10] P Bruland und M Dugas. *Transformations between CDISC ODM and openEHR Archetypes*. 2021. URL: https://ebooks.iospress.nl/volumearticle/37718.

[11] J Doods, P Neuhaus und M Dugas. „Converting ODM Metadata to FHIR Questionnaire Resources“. In: (2016), S. 456–460. DOI: 10.3233/978-1-61499-678-1-456. URL: https://ebooks.iospress.nl/publication/44653.

[12] S Hegselmann u. a. „Automatic Conversion of Metadata from the Study of Health in Pomerania to ODM“. In: (2017), S. 88–96. DOI: 10.3233/978-1-61499-759-7-88. URL: https://ebooks.iospress.nl/publication/46464.

[13] A Tapuria u. a. „Comparison and transformation between CDISC ODM and EN13606 EHR standards in connecting EHR data with clinical trial research data“. In: (2018). DOI: 10.1177/2055207618777676. URL: https://journals.sagepub.com/doi/10.1177/2055207618777676?url_ver=Z39.88-2003&rfr_id=ori:rid:crossref.org&rfr_dat=cr_pub%20%200pubmed.

[14] S Gessner u. a. „Automated Transformation of CDISC ODM to OpenClinica“. In: (2017), S. 95–99. DOI: 10.3233/978-1-61499-808-2-95. URL: https://ebooks.iospress.nl/publication/47514.

[15] M Dugas. „ODM2CDA and CDA2ODM: Tools to convert documentation forms between EDC and EHR systems“. In: (2015). DOI: 10.1186/s12911-015-0163-5. URL: https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-015-0163-5.

[16] I Soto-Rey u. a. „Standardising the Development of ODM Converters: The ODMToolBox“. In: (2018), S. 231–235. DOI: 10.3233/978-1-61499-852-5-231. URL: https://ebooks.iospress.nl/volumearticle/48788.

A.1. Akronyme

EDC „*Electronic Data Capture*“ Das Sammeln und Verarbeiten von Daten

RegEx „*Regular Expression*“ Ein Ausdruck, der genutzt werden kann, um Zeichenketten auf eine bestimmte Struktur zu überprüfen

IMI „*Institut für Medizinische Informatik*“ Eines der Institute der WWU. Diese Arbeit ist in Kooperation mit ihnen entstanden

XML „*eXtensible Markup Language*“ Eine Auszeichnungssprache zum Speichern hierarchisch strukturierter Daten

XSD „*XML Schema Definition*“ Eine Datei, welche beschreibt, wie ein XML-Dokument aufgebaut sein sollte, um einer bestimmten Definition zu entsprechen.

SAX „*Simple API for XML*“ Ein Standard, welcher beschreibt, wie man ein XML-Dokument parsen kann. Dieses wird sequentiell eingelesen und für definierte Ereignisse wird eine vorgegebene Rückruf-Funktion aufgerufen. Ein Programm kann eigene Funktionen registrieren und so das Dokument verarbeiten.

DOM „*Document Object Model*“ Bietet die Möglichkeit, die Hierarchie der XML-Knoten in Baumform darzustellen und so zu navigieren/ den Baum zu bearbeiten

CDISC „*Clinical Data Interchange Standards Consortium*“ Eine Non-Profit-Organisation, welche Standards zum Austausch von Daten aus klinischen Studien entwickelt

CDISC ODM „*Operational Data Model*“ von CDISC entwickeltes XML-Format (siehe Abschnitt 2.2)

lsa Abkürzung für und Dateiendung des LimeSurvey Archives (siehe Abschnitt 2.1.5)

lsr Abkürzung für und Dateiendung der LimeSurvey Response Datei (siehe Abschnitt 2.1.5)

lss Abkürzung für und Dateiendung der LimeSurvey Struktur Datei (siehe Abschnitt 2.1.5)

Eidesstattliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit über *„Titel“* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Vorname Nachname, Münster, 6. Juli 2021

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

Vorname Nachname, Münster, 6. Juli 2021