Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

gedcom7.js

Realisierung einer JavaScript-Bibliothek für das genealogische Austauschformat FamilySearch GEDCOM Version 7

Marius Müller & David Gruber

Bachelor-Projektarbeit

Betreuer: Christian Bettinger

Trier, 28.02.2023

Kurzfassung

In der Kurzfassung soll in kurzer und prägnanter Weise der wesentliche Inhalt der Arbeit beschrieben werden. Dazu zählen vor allem eine kurze Aufgabenbeschreibung, der Lösungsansatz sowie die wesentlichen Ergebnisse der Arbeit. Ein häufiger Fehler für die Kurzfassung ist, dass lediglich die Aufgabenbeschreibung (d.h. das Problem) in Kurzform vorgelegt wird. Die Kurzfassung soll aber die gesamte Arbeit widerspiegeln. Deshalb sind vor allem die erzielten Ergebnisse darzustellen. Die Kurzfassung soll etwa eine halbe bis ganze DIN-A4-Seite umfassen.

Hinweis: Schreiben Sie die Kurzfassung am Ende der Arbeit, denn eventuell ist Ihnen beim Schreiben erst vollends klar geworden, was das Wesentliche der Arbeit ist bzw. welche Schwerpunkte Sie bei der Arbeit gesetzt haben. Andernfalls laufen Sie Gefahr, dass die Kurzfassung nicht zum Rest der Arbeit passt.

Abstract

The same in English.

Inhaltsverzeichnis

Abbildungsverzeichnis

1

Einleitung und Problemstellung

In dieser Ausarbeitung ...

1.1 Anforderungsanalyse & Ziele

Folgende Anforderungen werden an die Bibliothek gestellt:

- AF01: Dateien oder Strings im Format Gedcom7 sollen eingelesen werden können
- AF02: Dateien sollen im Gedcom7 Format ausgegeben werden können
- AF03: Die Syntax von Dateien oder Strings soll gemäß der Gedcom7-Spezifikation überprüfbar sein
- AF04: Die in der Gedcom7-Spezifikation definierten Datentypen sollen unterstützt werden
- AF05: Eingelesene Dateien sollen gemäß der Gedcom7-Spezifikation verändert und erweitert werden können
- AF06: Die Bibliothek soll erweiterbar sein

2

Theoretische Grundlagen

In diesem Kapitel...

2.1 Genealogie und FamilySearch

Genealogie ist ein Überbegriff für die Familien- und Ahnenforschung und beschäftigt sich mit der historischen Herkunft und der Geschichte von Menschen weltweit [Ahn]. Dabei sind insbesondere Abstammungs- und Verwandtschaftsverhältnisse von besonderer Bedeutung, die anhand von Beiweisen aus validen Quellen in Stammbäumen zusammengefasst werden, die aufzeigen, wie eine Generation mit der nächsten verbunden ist. Auf Basis der so erlangten Erkenntnisse kann eine Familiengeschichte erstellt werden, die eine biographische Studie einer genealogisch nachgewiesenen Familie und der Gemeinde in der sie lebten, darstellt [Gen].

Das Aufkommen des Internets stellte einen Wendepunkt in der Genealogie dar.

2.2 GEDCOM Version 7

Das Datenformat FamilySearch GEDCOM 7.0 wurde 2021 von der Kirche Jesu Christi der Heiligen der Letzten Tage entwickelt und stellt ein einheitliches, flexibles Format für den Austausch von genealogischen Daten bereit. Das Ziel besteht darin, eine langfristige Speicherung von genealogischen Informationen zu ermöglichen, die für zukünftige Genealogen und die von ihnen verwendeten System zugänglich und verständlich ist [Fam22]. Die im Rahmen dieser Arbeit verwendete Version 7.0.11 wurde am 01.11.2022 veröffentlicht und stellt die aktuelleste[1] Version des Standards dar.

GEDCOM ist ein UTF-8 kodiertes hierarchisches Containerformat, das die Dateinamenserweiterung *.ged* verwendet. Der erste Character einer GEDCOM-Datei sollte das Byte-Order-Mark (U+FEFF) sein. Der Inhalt einer GEDCOM-Datei ist in sog. *Structures* unterteilt, die aus einem *Structure Type* und einem optionalen *Payload* bestehen und mehrere Substrukturen besitzen können. Hat eine *Structure* eine *Substructure*, dann ist die *Structure* die *Superstructure* der *Structure*. Jede

[1] Stand 31.01.2023

Substructure hat genau eine *Superstructure* und ist so in der Gesamtstruktur eindeutig zugeordnet. Eine *Structure*, die keine *Supertructure* besitzt, heißt *Record*. Alle Records zusammen mit einer *Header*- und einer *Trailer*-Struktur bilden ein *Dataset*, das den Inhalt einer GEDCOM-Datei darstellt. [Fam22]

Der *Payload* einer *Structure* ist eine Zeichenkette eines bestimmten Datentyps, die entweder Informationen für die *Supertructure* bereithält, oder einen Zeiger auf eine andere *Structure* repräsentiert und somit auf diese verweist. GEDCOM v7 definiert 11 verschiedene Datentypen in [Fam22] mit denen Namen, Daten, Uhrzeiten, Texte und vieles mehr dargestellt werden können. Der *Structure Type* ist eindeutig definiert durch eine URI und gibt an, welche Bedeutung und welchen Datentyp die *Structure* besitzt, welche *Substructures* enthalten sein können und mit welcher Kardinalität diese auftreten können. [Fam22]

Kodiert wird der Inhalt einer GEDCOM-Datei in sog. *Lines*, die eine Zeichenkettenrepräsentation einer Struktur (bzw. eines Teils einer Struktur) darstellen und wie folgt aufgebaut sind (eckige Klammern repräsentieren optionale Inhalte):

Level D [Xref D] Tag [D LineVal] EOL

- Level: Eine Line beginnt mit einem Level, das die Verhältnisse der *Structures* untereinander beschreibt. Alle *Structures* mit dem kleinstmöglichen Level 0 sind Records - Level ≥ 1 repräsentieren *Substructures*. Eine *Structure* mit dem Level x ist also die *Superstructure* aller folgenden *Structures* mit dem Level $x + 1$.
- D: *D* steht für *Delimiter*, was englisch für Trennzeichen ist und repräsentiert in diesem Fall das Leerzeichen mit dem Unicode $u + 0020$.
- Xref: Xref ist die Abkürzung für *Cross-Reference Identifier* und fungiert als Adresse für eine *Structure*. Möchte man von einer *Structure* auf eine andere *Structure* verweisen, kann dies über einen Zeiger-Payload auf die entsprechende *Structure* realisiert werden.
- Tag: Der *Tag* kodiert den *Structure Type* einer *Structure*.
- LineVal: Im *LineVal* einer Struktur ist der Payload kodiert.
- EOL: EOL steht für End-Of-Line und kodiert das Ende einer Line. Im Format GEDCOM v7 kann dies entweder durch einen Carriage-Return (Unicode U+000D), Line-Feed(Unicode U+000A) oder einen Carriage-Return gefolgt von einem Line-Feed repräsentiert werden.

Ein Ausschnitt aus einer GEDCOM-Datei ist in 2.1 dargestellt. Dieser Ausschnitt zeigt einen *Record* vom Typ *Family*, in dem Informationen über eine Familie gespeichert werden können. Der Familie wurde der Cross-Reference Identifier *@F1@* zugewiesen, sodass im Dokument auf dieses verwiesen werden kann. Der Ehemann und die Ehefrau der Familie (engl. Husband und Wife) sind die Individuen *I1* und *I2*, die ebenfalls in der Gedcom7-Datei definiert sind. Dieser Zusammenhang wird über die Cross-Reference Identifier *@I1@* und *@I2@* ausgedrückt. Außerdem wird ein Family-Event, nämlich die Hochzeit der beiden Ehepartner aufgeführt und auf den 1.März 1951 datiert. Als letzte Information ist die Anzahl der Kinder (NCHI: Number of Children) mit 2 spezifiziert.

```
0 @F1@ FAM
1 HUSB @I1@
1 WIFE @I2@
1 MARR
2 DATE 1 MAR 1951
1 NCHI 2
```

Listing 2.1: Beispiel für einen Individual Record

Detaillierte Erklärungen, alle Informationen zu *Structure Types*, Datentypen, usw. und viele weitere Beispiele können in [Fam22] nachgelesen werden.

2.3 Nearley

Allgemeines

Nearley.js ist eine JavaScript-Bibliothek zum Parsen kontextfreier Grammatiken (CFGs). Sie bietet einen vielseitigen und effizienten Parsing-Algorithmus, der auf dem Algorithmus von Earley basiert und es ermöglicht, mehrdeutige und rekursive Grammatiken mit Leichtigkeit zu behandeln. Die Bibliothek ist modular aufgebaut, so dass Benutzer ihre eigenen Parser und Lexer definieren und Parser aus externen Quellen wie BNF- und EBNF-Grammatiken erzeugen können. Nearley.js ist in reinem JavaScript implementiert und kann in jeder Umgebung ausgeführt werden, die JavaScript unterstützt, einschließlich Webbrowsern und serverseitigen Umgebungen.

Nearley.js bietet eine Reihe nützlicher Funktionen, darunter JavaScript-Aktionen, genannt *Postprocessor*, bei denen Benutzer Code angeben können, der ausgeführt wird, wenn bestimmte Teile der Eingabe erkannt werden. Nearley.js ist Open-Source und hat eine lebendige Gemeinschaft von Nutzern und Mitwirkenden, was es zu einem zuverlässigen und gut unterstützten Werkzeug für das Parsen komplexer Texte macht.

Grammatik

Im Laufe der Implementierung haben wir uns dazu entschieden, die komplette GEDCOM-Grammatik mittels Nearley abzubilden. Dies hat den Vorteil, dass wir alle Regeln der GEDCOM-Spezifikation in einer Prüfung abdecken können. Außerdem bietet Nearley die Möglichkeit Teile der Grammatik in einzelnen Dateien auszulagern und nur für Teile der Grammatik Parser zu erstellen. Dies ermöglicht es uns eine Teilprüfung von nur einer GEDCOM-Struktur einer ganzen Datei durchzuführen. Dies ist insbesondere bei Manipulationen von einzelnen Strukturen innerhalb einer Datei von Vorteil, da anschließend nur die betroffene Struktur neu geprüft werden muss.

Beispiel

2.4 Mocha

Im Rahmen der Implementierung unserer Bibliothek, war es uns ein zentrales Anliegen, das Testen dieser zu gewährleisten. Zu diesem Zweck haben wir uns auf die Suche nach einem geeigneten Testframework für JavaScript begeben. In diesem Kontext stießen wir auf Mocha, welches eine API bereitstellt, die das einfache Erstellen von Tests ermöglicht. Wie bei vielen anderen Testframeworks können Assertion-Funktionen genutzt werden, um die Tests zu überprüfen.

Wie im Beispiel-Test in Listing 2.2 veranschaulicht wird, wird zunächst ein aussagekräftiger Name für den Test als erster Parameter der Mocha-Funktion *describe* übergeben. Im zweiten Parameter wird eine Callback-Funktion übergeben, die den Code für den Test enthält, beginnend mit dem Setup-Code. Hierbei wird in unserem Fall ein Array mit mehreren Gedcom-Dateinamen erstellt, die sich im angegebenen *path* befinden.

Durch die Verwendung der Funktion *forEach*, welche aus einem Mocha-Plugin namens *mocha-each* stammt, wird über alle Dateinamen iteriert. Für jeden Dateinamen wird ein neuer Test erstellt und der Code in der anonymen Callback-Funktion für jeden *fileName* im Array *gedFiles* ausgeführt.

Der Ablauf für eine Datei im Listing 2.2 ist wie folgt: Zunächst wird die Datei mithilfe der asynchronen Helper-Funktion *readGedFile* eingelesen und in der Variable *beforeParsing* gespeichert. Anschließend wird die Datei mit der Funktion *parseString* des Moduls *gedcomParser* als Dataset geparst. Hierbei wird eine Grammatikprüfung durchgeführt und alle Daten aus der Datei sind nach erfolgreicher Prüfung über die Dataset Klasse und deren Methoden zugänglich.

Im nächsten Schritt wird im aktuellen Verzeichnis eine temporäre Datei namens *temp.ged* erstellt und anschließend mit der Funktion *dataset.write* mit den eingelesenen Daten des Datasets beschrieben.

Schließlich wird mithilfe der Funktion *expect*, welche aus dem Mocha-Plugin *chai* stammt, überprüft, ob die eingelesene Datei und das geschriebene Dataset in der *temp.ged* Datei identisch sind.

Der beschriebene Testmechanismus gewährleistet, dass die Daten, die aus einer GEDCOM-Datei gelesen werden, bei erfolgreichem Parsen nach Schreiben ohne Manipulation am Dataset weiterhin identisch und konsistent sind.

```
// GEDCOM READING AND WRITING TEST
describe('test if gedcom file is equivalent before and after parsing', () => {
   const path = 'test/sampleData/ExampleFamilySearchGEDCOMFiles/';
   //correct files
```

```
  const gedFiles = [
   'escapes.ged',
   'long-url.ged',
   'maximal70_without_extensions.ged',
   'minimal70.ged',
   'remarriage1.ged',
   'same-sex-marriage.ged',
   'voidptr.ged'
  ];

  forEach(gedFiles)
   .it('#%s', async (fileName) => {
   // read Gedcom file as String
   const beforeParsing = await readGedFile(path + fileName);
   // parse Gedcom String and write it to temp.ged
   const dataset = gedcomParser.parseString(beforeParsing);
   await fs.writeFile(path + 'temp.ged', '');
   await dataset.write(path + 'temp.ged');
   // read temp.ged as String and compare it with original file
   const afterParsing = await readGedFile(path + 'temp.ged');
   // expect files to be equal
   expect(diffChars(beforeParsing, afterParsing)).to.have.lengthOf(1);
 });
});
```

Listing 2.2: Beispiel für einen Mocha-Test

3

Related Work

Vor Beginn der theoretischen Ausarbeitung der Gedcom7-Bibliothek haben wir eine umfassende Untersuchung der bestehenden Gedcom-Bibliotheken durchgeführt, darunter auch einige der beliebtesten Bibliotheken, die in Python oder Java entwickelt wurden. Wir haben uns dabei auf zwei Bibliotheken fokussiert und diese als Orientierung für unsere Arbeit herangezogen.

3.1 gedcom7code/js-parser

Die erste Bibliothek ist der js-parser, welcher von Luther Tychonievich, einem Managing Editor von FamilySearch Gedcom, entwickelt wurde und als minimaler Parser für Gedcom7-Zeilen dient. Dieser Parser, der in JavaScript geschrieben wurde, basiert auf einer Regular Expression und liefert die einzelnen Bestandteile der Zeile zurück. Die grundlegende Struktur einer Gedcom-Zeile kann damit ermittelt werden.

Die vorliegenden Bestandteile dienen als rudimentäre Basis für die Properties unserer Structure-Klasse, von welcher alle *Record*-Klassen erben. Dennoch handelt es sich hierbei lediglich um eine Demonstration eines minimalen Parsers und nicht um eine vollständige Bibliothek. Infolgedessen führten wir weitere Recherchen durch.

3.2 python-gedcom

Die zweite Bibliothek, die uns besonders gefallen hat, ist die python-gedcom-Bibliothek. Hier hat uns vor allem die an die Spezifikation angelehnte Klassenhierarchie beeindruckt. Für unsere Zwecke der Gedcom7-Bibliothek bot diese eine ideale Vorlage. Auffällig ist hierbei, dass jeder Gedcom-Record eine eigene Klasse besitzt, die von einer Elternklasse Structure erbt.

Ein weiterer Aspekt, der uns an dieser Bibliothek gefiel, ist der Ansatz, die einzelnen Felder, die eine Zeile eines *Record*s oder *Subrecord*s ausmachen, zu speichern. Dieser Ansatz ermöglicht es uns, Gedcom-Dateien konsistent zu lesen, interpretieren, manipulieren und schreiben.

Allerdings wurde bei der python-gedcom-Bibliothek unserer Meinung nach kaum Wert auf die Prüfung der genauen Spezifikationsvorgaben gelegt. Records können beliebig hinzugefügt werden und nicht spezifikationskonforme Records können erstellt werden.

Da uns die Einhaltung der Spezifikation sehr wichtig ist, haben wir uns entschieden, in Gedcom7.js eine sehr detaillierte Grammatik mittels Nearley.js zu erstellen. Diese Grammatik wird beim Parsen der Gedcom-Datei und nach jeder Operation an einem Dataset verwendet, um das Dataset auf Korrektheit zu prüfen und somit auch eine korrekte Gedcom-Datei zu gewährleisten.

4

Konzept

Die Bibliothek *gedcom7.js* lässt sich wie in Abbildung 4.1 dargestellt in vier logische Teile gliedern. Das zentrale Element ist der Gedcom Parser, mit dem Dateien oder Strings im Format Gedcom7 eingelesen werden und mit Hilfe von *Nearley* auf Korrektheit der Syntax überprüft werden können. Die dafür zugrundeliegende Grammatik wird mit Hilfe eines *Grammatik Generators* generiert, der die in [Fam22] definierte Spezifikation in eine nearley-konforme Syntax überführt. Die so eingelesenen Informationen werden in Gedcom Datenstrukturen gespeichert, die verändert und erweitert werden und anschließend im Format Gedcom7 ausgegeben werden können. In den folgenden Abschnitten werden die vier Teile und das Zusammenspiel dieser in detaillierter Form vorgestellt.

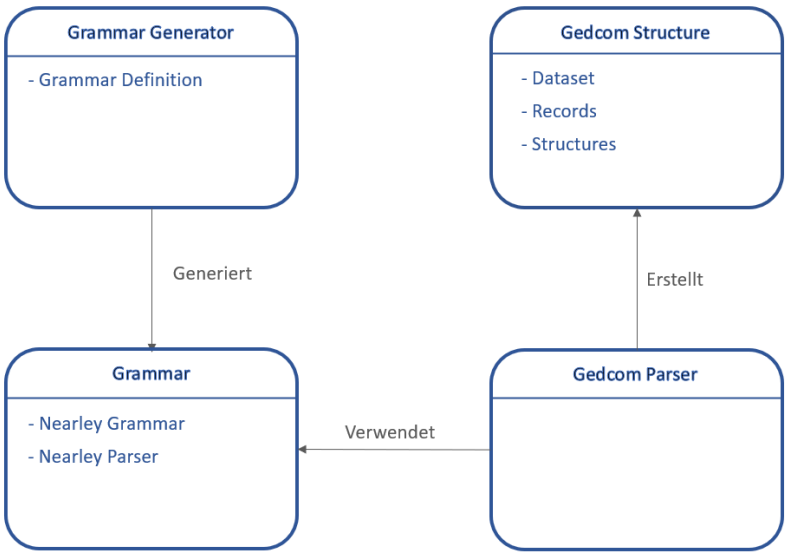

Abbildung 4.1: Allgemeiner Aufbau

4.1 Gedcom Grammatik

Eine wichtige Anforderung für die Bibliothek ist, dass die Syntax von Dateien oder Strings, die eingelesen werden, gemäß der Gedcom7-Spezifikation überprüfbar sein soll. Da die Gedcom Datenstrukturen veränderbar und erweiterbar sein sollen, ist es wichtig, dass eine Syntaxüberprüfung nach Änderungen auf einfache Weise möglich ist. Umgesetzt wird diese Syntaxüberprüfung mit Hilfe des in Kapitel 2.3 vorgestellten JavaScript-Parser-Toolkits *Nearley*. Mit *Nearley* können auf einfach Weise, menschenlesbare Grammatiken erstellt und zu einem *Nearley-Parser* kompiliert werden. Von großem Vorteil ist dabei, dass Features wie Postprozessoren und die Implementierung eines Lexers unterstützt werden.

4.1.1 Pre- und Postprozessor

Standardmäßig packt ein *Nearley-Parser* jedes Zeichen, das mit einer Regel übereinstimmt, in ein Array [Cha]. Bei komplexeren Grammatiken wie der Gedcom7 Spezifikation führt dies dazu, dass sehr viele Arrays innereinander verschachtelt werden, sodass schnell zweistellige Verschatlungsgrade erreicht werden, was ein weiterarbeiten mit den Ergebnissen erschwert. Mit Hilfe von Postprozessoren können jeder *Nearley Regel* Verarbeitungsanweisungen zugewiesen werden, sodass die Ergebnisse beispielsweise im JSON-Format zurückgegeben werden. Auf diese Weise können die eingelesenen Dateien bereits bei der Syntaxüberprüfung in eine passende Darstellungsform gebracht werden, sodass eine leichte Überführung in die passende Gedcom Datenstruktur möglich ist.

Desweiteren kann ein Lexer verwendet werden, um die Arbeit mit *Nearley* zu optimieren. Ein *Nearley-Parser* teilt die Eingabedaten standardmäßig in einen Strom von einzelnen Zeichen, die sequentiell abgearbeitet werden, was auch als *Scannerless Parsing* bezeichnet wird. Ein Lexer ist eine Art Preprozessor, der die Eingabedaten in größere Einheiten, die sog. *Tokens* zusammenfasst [Cha]. Auf diese Weise wird der Aufwand beim Parsen verringert und die Interpretation der Eingabedaten fällt oft leichter. Ein einfaches Beispiel hierfür ist eine Regel die einen Zahlenwert erwartet. Ist der Eingabewert beispielsweise "137", würde ein *Nearley-Parser* standardmäßig jede Ziffer einzeln einlesen und im Postprozessor müsste definiert werden, dass die aufeinanderfolgenden Ziffern als ein Zahlenwert interpretiert werden sollen. Mit Hilfe eines Lexers könnte eine einfache Regel definiert werden, die den kompletten Zahlenwert als ein Token vorverarbeitet. Im Rahmen dieser Arbeit wurde der JavaScript Lexer *Moo.js* [Rad] verwendet. *Moo.js* zeichnet sich durch seine Geschwindigkeit[1] aus und wird von *Nearley* als Lexer unterstützt.

4.1.2 Nearley-Parser für Gedcom7

Da die Gedcom7- sowie die Nearley Syntax beide auf EBNF-Sprachkonzepten basieren, lässt sich die Gedcom7 Spezifikation ohne weiteres in eine Nearley Grammatik übersetzten, die dann zu einem Nearley-Parser kompiliert werden kann. Wird

[1] Laut den Entwicklern ist Moo.js der schnellste JavaScript-Lexer und ∼2-10 mal schneller als herkömmliche Lexer [Rad].

diesem Nearley-Parser eine Gedcom7-Datei (.ged) kodiert als UTF-8 Zeichenkette übergeben, erfüllt dieser die folgenden zwei Aufgaben:

1. Überprüfung der Gedcom7 Syntax

Der Nearley-Parser überprüft den übergebenen Gedcom7-String Line für Line, indem er alle Zeichen (bzw. Tokens) sequentiell liest, bis ein End-Of-Line (EOL) Token gefunden wird. Nach jedem Zeichen das eingelesen wird, überprüft der Parser, welche in der Grammatik definierten Regeln durch das neu eingelesene Zeichen nicht mehr mit der Zeichenkette übereinstimmen und verwirft diese. Wird ein EOL Token gelesen werden die Postprozessoren aller übereinstimmenden Regeln ausgeführt und ein Array mit den Ergebnissen dieser Postprozessoraufrufe als Ergebnis der Line zurückgegeben. Da die Gedcom7 Grammatik nicht mehrdeutig ist, findet der Parser bei korrekter Gedcom7 Syntax immer ein eindeutiges Ergebnis[2] (d.h. beim Erreichen des EOL Tokens ist maximal eine übereinstimmende Regel übrig). Werden bei diesem Prozess alle Regeln der Grammatik ausgeschlossen, bevor ein EOL Token gelesen wird, ist die Syntax des übergebenen Gedcom7-Strings nicht korrekt und ein Syntaxfehler kann erzeugt werden. Da die Zeichenkette sequentiell abgearbeitet wird, kann bei auftretendem Fehler genau aufgezeigt werden, welche Line und welches Zeichen fehlerhaft sind.

2. Extrahieren der Strukturinformationen

Eine weiterer Aufgabe des Nearley-Parsers ist es, die Strukturinformationen des Gedcom7-Strings zu extrahieren, sodass im nächsten Schritt eine einfache Überführung in entsprechende Gedcom Datenstrukturen möglich ist. Durch den sequentiellen Aufbau einer Gedcom7-Datei wird eine Struktur stets vor seinen Substrukturen definiert. Da das erste Token jeder Line stets das Level der Line repräsentiert, kann der Nearley-Parser die Abhängigkeiten der Lines zueinander zuordnen und es ist zu jedem Zeitpunkt eindeutig, welcher Superstruktur eine Struktur zugeordnet werden soll. Folgende Informationen können also durch den Nearley-Parser extrahiert werden:

- **URI**: Auch wenn bestimtme Structuretypes denselben Tag besitzen, kann aus der Kombination von Level und Tag die eindeutige Gedcom URI bestimmt werden
- **Datentyp**: Sofern ein Payload in der Line vorhanden ist, kann mit Hilfe der URI der Datentyp des Payloads bestimmt werden
- **Superstruktur**: Zu jeder Line kann die entsprechende Superstruktur angegeben werden, sofern es sich nicht um einen Record (Structure mit Level 0) handelt, die keine Superstructure besitzen
- **Substrukturen**: Hat eine Sturktur eine oder mehrere Substrukturen, können diese auf Basis des Levels der Lines bestimmt werden

[2] Hier Kapitel ansprechen in dem über ambigious grammar geredet wird, bei level problem

4.2 Grammatik Generator

In der Gedcom7 Spezifikation werden 181 Structuretypes verteilt auf 7 Records definiert, die alle in einer Line der Form

Level D [Xref D] Tag [D LineVal] EOL

dargestellt werden. Sollen diese Structuretypes in eine Nearley Grammatik überführt werden, muss für jede dieser Strukturen und jede mögliche Kombination an Substrukturen eine Regel erstellt werden. Da dies eine sehr repetitive Aufgabe ist und sich die Regeln nur an bestimmten Stellen unterscheiden, lässt sich die Grammatikerstellung durch einen Grammatik Generator automatisieren. Dazu können Definitionsdateien erstellt werden, die die für alle Structuretypes die folgenden Informationen bereithalten:

- **URI**: Die URI des Structuretypes wird benötigt, um eine Struktur eindeutig zuordnen zu können
- **LineType**: Der LineType gibt an, wie die Line aufgebaut ist (Cross-Reference-Identifier vorhanden? Payload vorhanden?)
- **Datatype**: Sofern ein Payload in der Line vorhanden ist, kann über den Datatype die Syntax des Payloads ermittelt werden
- **Tag**: Der Tag wird benötigt, damit die Nearley Regeln eindeutig sind
- **Substructures**: In der Gedcom7 Spezifikation sind für alle Structuretypes alle möglichen Substructures definiert. Mit dieser Information können alle korrekten Fälle in Nearley Regeln abgebildet werden
- **Level**: Um eine eindeutige Grammatik zu generieren, müssen die Level mit denen ein jeweiliger Structuretype auftreten kann, zwingend mit angegeben werden. Da in der gedcom7 Spezifikation TAGs mehrfach für verschiedene Typen verwendet werden, kann nicht einfach ein generisches Level für die Regeln verwendet werden, das ganzzahlige Werte akzeptiert, da die entstehende Grammatik damit mehrdeutig wäre. Ein Beispiel hierfür sind die Structures *g7:HEAD-DATE* und *g7:DATE-exact* im Gedcom Header. Mit einem generischen Level wären die Regeln für beide Structuretypes identisch mit

  Level D "DATE"D DateExact EOL

  Wird eine solche Line als Substructure eines Header Records von dem Nearley Parser gelesen, kann dieser nicht entscheiden, ob es sich um ein *g7:HEAD-DATE* oder ein *g7:DATE-exact* handelt und würde somit zwei Ergebnisse aufrecht erhalten. Um diese Mehrdeutigkeit zu verhindern, wird das Level in der Definition angegeben.

Anhand dieser Informationen kann der Grammatik Generator automatisiert Nearley Regeln formulieren. Diese Regeln können zu einer Grammatik zusammengefasst und anschließend vom Generator zu einem Nearley-Parser kompiliert werden. Anhand des LineTypes kann der Generator den Regeln die passenden Postprozessoren zuweisen, die für das Extrahieren der Strukturinformationen zuständig sind. Auf diese Weise kann ein voll funktionaler Nearley-Parser automatisiert generiert

werden, der die Gedcom7-Syntax vollständig parsen und alle für die weitere Verarbeitung benötigten Informationen extrahieren kann.

Ein weitere großer Vorteil an dieser Automatisierung ist, dass zur Erfüllung der Anforderung der einfachen Erweiterbarkeit der Bibliothek beigetragen wird. Sollte die Bibliothek in zukünftigen Projekten durch neue Strucutretypes o.ä. erweitert werden, ist dies auf einfache und verständliche Weise durch das Hinzufügen neuer Einträge in die Strukturdefinitionen möglich. Desweiteren bildet der Grammatik Generator ein Fundament für einen wichtigen Use-Case, der in weiterführenden Arbeiten adressiert werden sollte: der Möglichkeit Extensions zu definieren. Die Gedcom7 Spezifikation definiert die wichtigsten Strukturen zur Speicherung genealogischer Informationen - für alle Informationen die über diese Standardstrukturen hinausgehen, müssen Extensions definiert werden. Da genealogische Informationen sehr vielfältig sein können, sind Extensions ein probates Mittel, dass in vielen Anwendungen genutzt wird. Mit Hilfe des Grammatik Generators kann die Definition von Extensions umgesetzt werden, indem eine Schnittstelle zum Generator entwickelt wird, die dem Benutzer zur Verfügung gestellt wird. Über diese Schnittstelle kann die Strukturdefinition erweitert werden und anschließend die Grammatik neu generiert und kompiliert werden. Auf diese Weise könnte die Bibliothek auf die Anforderung aller Benutzer angepasst werden.

4.3 Gedcom Strukturen

Die zentrale Struktur in einer Gedcom7 Datei ist das sog. *Dataset*. Jedes Dataset muss mit einem Header Structure beginnen, der Metadaten über das gesamte Dataset beinhaltet und dabei u.a. Aussagen über den Ort und Zeitpunkt der Erstellung und den Ersteller des Datasets selbst machen kann. Die Mindestanforderung an den Header ist, dass die verwendete Gedcom Version in einer dafür vorgesehenen Structure spezifiziert ist. Abgeschlossen wird jedes Dataset mit einer Trailer Line, die das Ende des Datasets repräsentiert. Eine minimales Gedcom7 Dataset sieht also wie folgt aus:

```
0 HEAD
1 GEDC
2 VERS 7.0
0 TRLR
```

Listing 4.1: Minimales Gedcom7 Dataset

Alle weiteren genealogischen Informationen können in einem oder mehreren Records festgehalten werden. Folgende Records sind in der Gedcom7 Spezifikation definiert:

- **Family (FAM)**: Der Family Record wurde ursprünglich so strukturiert, dass er eine Familie mit einem männlichen Ehemann und einer weiblichen Ehefrau

repräsentiert. Um die Migration von bestehenden Gedcom-Dateien auf Gedcom7 zu erleichtern, wurde die Benamung der Strukturen beibehalten. Trotzdem sollen in Gedcom7 Familien, Heirat, Zusammenleben und Adoption unabhängig vom Geschlecht der Partner angegeben werden können und daher das Geschlecht und die Rollen von Partnern nicht aus der Husband- bzw. Wife Struktur abgeleitet werden.

- **Individual (INDI)**: Zusammenstellung von Fakten und Hypothesen über eine Person. Diese können aus verschiedenen Quellen stammen, die durch Quellenangaben dokumentiert werden können.
- **Multimedia (OBJE)**: Eine Referenz zu einer oder mehrerer digitaler Dateien, angereichert mit Informationen über den Inhalt und den Typ der Datei.
- **Repository (REPO)**: Beinhaltet Informationen über Personen oder Institutionen, die eine Sammlung von Quellen besitzen.
- **Shared Note (SNOTE)**: Eine Sammlung von Informationen, die nicht vollständig in andere Strukturen passen. Beispiele wären Forschungsnotizen, alternative Interpretationen oder Argumentationen
- **Source (SOUR)**: Beschreibt eine Quelle, indem auf bestimmte Dokumente oder Verzeichnisse verwiesen wird.

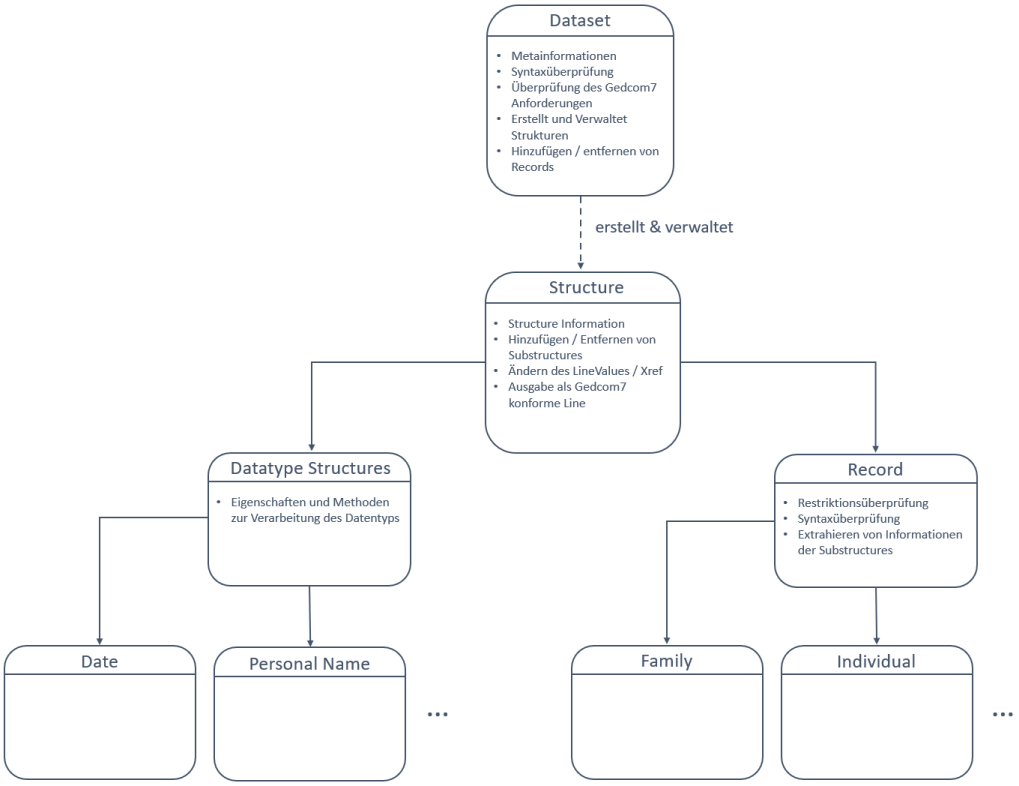

Abbildung 4.2: Gedcom Strukturen

- **Submitter (SUBM)**: Beschreibt eine Person oder eine Institution, die im Dataset enthaltene Informationen begesteuert hat.

4.3.1 Structure

In der Bibliothek *gedcom7.js* wird jede Line, die vom Parser gelesen wird, in Form einer Structure abgebildet (siehe Abbildung 4.2). Diese Datenstruktur hält alle Informationen wie Level, Tag und LineValue der Line bereit und enthält zudem die Superstructure und alle Substructures. Eine wichtige Anforderung ist zudem, dass die eingelesene Gedcom7 Datei veränderbar und erweiterbar sein soll. Daher enthält die *Structure* Datenstruktur Methoden mit der Substructures hinzugefügt bzw. entfernt werden können und mit denen der Payload der Structure verändert werden kann. Um die veränderte Datenstruktur als Gedcom7 Datei speichern zu können, muss jede Structure als Gedcom7 konforme Line ausgegeben werden können. Außerdem werden zwei spezielle Structures definiert, die Datatype Structures und Records.

1. Records

Die oben aufgeführten Records, also Structures mit Level 0 wie z.B. der Family Record, sind die Superstructures aller weiteren Structuretypes und haben daher besondere Anforderungen. Daher werden in *gedcom7.js* alle Records in Form einer eigenen Record Datenstruktur abgebildet, die Structure erweitert. Dabei sollen Methoden bereitgestellt werden, mit denen die Informationen, die in den Substructures enthalten sind, extrahiert werden können. Ein Beispiel hierfür wäre eine Methode, die alle Informationen über die Residenz einer Familie, die über viele Substructures verteilt sind, zusammenfasst und in übersichtlichem Format zurückgibt. In einem Gedcom7 Record kann zudem über eine *Restriction Structure* der Zugriff auf Informationen dieses Records eingeschränkt werden. Die Record Datenstruktur sollte diese Einschränkungen verwalten und die Ausgabe von Informationen dementsprechend anpassen. Außerdem besitzen alle Records Möglichkeiten zur Syntaxüberprüfung des Records und all seiner Substructures, sodass nach dem Hinzufügen einer Structure in einen Record, die Syntax des Records überprüft werden kann, um zu entscheiden ob das Hinzufügen syntaktisch korrekt war. Dies bietet den Vorteil, dass nicht jedes Mal die Syntax des gesamten Datasets überprüft werden muss, obwohl sich nur ein Record verändert hat.

2. Datatype Structures

Der Payload von Lines kann verschiedene Datentypen haben, die in der Gedcom7 Datei als Zeichenkette kodiert sind. Handelt es sich dabei um einen einfachen Linevalue vom Datentyp *Text* ist es ausreichend, diesen als Zeichenkette in der Structure zu hinterlegen. Bei komplexeren Datentypen wie *Date* ist es jedoch notwendig, die Structure um Methoden und Eigenschaften zu erweitern, um die weitere Verarbeitung zu erleichtern. Daher werden für komplexere Datentypen wie *Date* oder *Age* spezielle *Datatype Structures* bereitgestellt, mit denen beispielsweise das Da-

tum eines Events in eine JavaScript konformen Datumsstruktur überführt werden kann.

Soll der in Listing **??** vorgestellte Family Record eingelesen werden, würde sich folgende Baumstruktur ergeben:

4.3.2 Dataset

Die in Abschnitt 4.3.1 beschriebenen Structures werden in einer *Dataset* Datenstruktur zusammengefasst, die alle genealogischen Informationen einer Gedcom7 Datei enthält. Die Hauptaufgabe des Datasets besteht darin, Structures zu erstellen und zu verwalten. Wird eine Gedcom7 Datei mit korrekter Syntax mit dem in 4.1.2 vorgestellten Nearley Parser eingelesen, extrahiert dieser alle Structure Informationen. Anschließend kann ein *Dataset* erstellt werden, das all diese Informationen einliest, daraus Structures erstellt und die Zusammenhänge zwischen diesen Structures modelliert, sodass, wie in Abbildung 4.3.1 beispielhaft gezeigt, eine Baumstruktur mit allen Records entsteht. Um ein Dataset mit neuen genealogischen Informationen anzureichern, werden Methoden zum Hinzufügen bzw. zum Entfernen von Records bereitgestellt. Da das Hinzufügen bzw. Entfernen von Strukturen zu einer inkorrekten Gedcom7 Syntax führen kann, müssen Methoden zur Syntaxüberprüfung implementiert werden. Des Weiteren stellt das Dataset Metainformationen über eine Gedcom7 Datei zur Verfügung und überprüft bestimmte Anforderungen, die in der Gedcom7 Spezifikation angegeben werden. Beispiele hierfür sind, dass jede Gedcom7 Datei mit dem *Byte-Order-Mark*[3] beginnen sollte oder dass alle Structure, auf die über einen Cross-Reference-Identifier verwiesen wird, definiert seien müssen, bevor auf diese verwiesen wird.

4.4 Gedcom Parser

Die in diesem Kapitel vorgestellten Konzepte und Datenstrukturen werden alle im Gedcom Parser vereinigt, der die zentrale Instanz der Bibliothek *gedcom7.js* darstellt. Abbildung 4.3 zeigt ein Sequenzdiagramm, das den allgemeinen Ablauf beim parsen einer Gedcom7 Datei mit dem Gedcom Parser zeigt.

Der Gedcom Parser ließt eine Gedcom7 Datei ein und konvertiert diese in eine Zeichenkette. Die Zeichenkette kann dann an einen Nearley Parser übergeben werden, der Line für Line ließt, die Syntax überprüft und dabei die Structure Informationen extrahiert. Ist die Syntax der Gedcom7 Datei korrekt, werden die gesammelten Structure Informationen an den Gedcom Parser zurückgegeben - anderenfalls wird das Einlesen mit einer Fehlermeldung beendet. Anschließend überprüft der Gedcom Parser die Kardinalität der eingelesenen Structures (beispielsweise darf nur eine *HUSB*-Struktur pro Family Record enthalten sein).[4]. Sofern keine Fehler bei der Kardinalitätsüberprüfung gefunden werden, wird ein neues Dataset

[3] Erklärung

[4] Die Kardinalitätsüberprüfung wurde in den Gedcom Parser ausgelagert, da Nearley ein Streaming-Parser ist und somit zu keinem Zeitpunkt weiß, ob noch weitere Eingaben zu erwarten sind. Daher werden Konzepte wie Kardinalitätsüberprüfungen nicht von Nearley unterstützt. [Cha]

erstellt und die von Nearley extrahierten Structure Informationen an das Dataset übergeben. Das Dataset erstellt den Header und den Trailer, der in jedem Dataset vorhanden sein muss und erstellt anschließend alle Structures auf Basis der Structure Informationen. Dazu wird jeder Eintrag der Structure Informationen auf den Structure Type untersucht (Record, Datatype Structure oder allgemeine Structure) und auf Basis dessen ein Structure mit allen Informationen erstellt. Diese Structure wird dann in das Dataset eingegliedert, indem die entsprechende Superstructure und alle Substructures zugewiesen werden. Sind alle Structures erstellt, wird überprüft, ob dass alle Cross-Reference-Identifier, auf die im Dataset verwiesen wird auch innerhalb des Datasets definiert sind. Ist dies der Fall, wird das Dataset zurückgegeben. Dieses Dataset kann dann wie in Abschnitt 4.3 beschrieben verändert und erweitert werden.

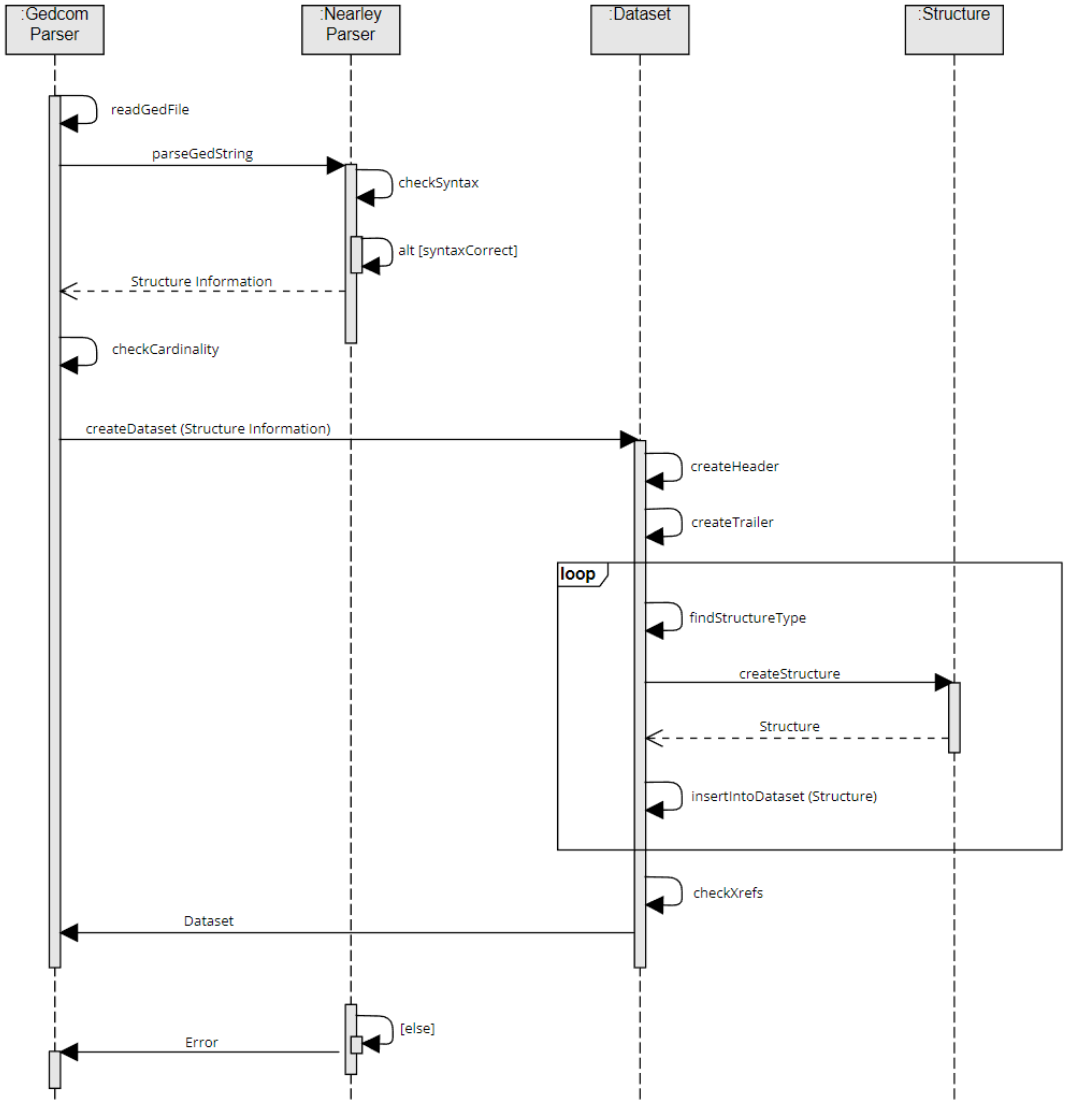

Abbildung 4.3: Ablauf Gedcom Parser

5

Implementierung & Test

In diesem Kapitel wird beschrieben wie das im Kapitel 4 vorgestellte Konzept in der Bibliothek *gedcom7.js* implementiert wird.

5.1 Gedcom Grammatik

5.1.1 Gedcom7 Syntax in Nearley

Da die Gedcom7- sowie die Nearley Syntax beide auf EBNF-Sprachkonzepten basieren, lässt sich die Gedcom7 Spezifikation ohne weiteres in eine Nearley Grammatik übersetzten. Um Nearley Regeln für eine *Gedcom Line*[1] zu definieren, können die folgenden Tokens für das Leerzeichen, den *Cross-Reference Identifier* und die End-Of-Line Zeichenfolge in Form von regulären Ausdrücken definiert werden:

```
D    : /[ ]/
Xref : /\@[A-Z0-9\_]+\@/
EOL  : /(?:\r\n?|\n)/
```

Listing 5.1: Tokens für eine Gedcom Line, definiert als regulärer Ausdruck

Diese regulären Ausdrücke werden in der Vorverarbeitungsphase vom Moo-Lexer verwendet, um zusammenhängende Zeichen zu Tokens zu gruppieren, die dann in der Nearley Grammatik über den Tokennamen mit einem vorangestellten %-Zeichen angesprochen werden können. Soll nun die erste Line eines Family-Records geparsed werden, könnte dies mit der folgenden Nearley-Regel umgesetzt werden:

```
record_FAM -> "0" %D %Xref %D "FAM" %EOL
```

Listing 5.2: Nearley Regel zum parsen eines Family Records

Diese Regel akzeptiert eine Line mit dem Level 0, einem syntaktisch korrekten

[1] siehe Kapitel GEDCOM Version 7

Cross-Reference-Identifier, dem Tag *FAM* gefolgt von einem EOL-Token. Getrennt werden die Bestandteile durch ein Leerzeichen.

Sollen nun ebenfalls HUSB- und WIFE Structures als Substructures des Family Records akzeptiert werden, könnte die Nearley Grammatik wie folgt erweitert werden:

```
record_FAM
    -> "0" %D %Xref %D "FAM" %EOL
    | record_FAM record_FAM_Substructs:+

record_FAM_Substructs
    -> "1" %D "HUSB" %D %Xref %EOL
    | "1" %D "WIFE" %D %Xref %EOL
```

Listing 5.3: Nearley Regel zum parsen eines Family Records mit HUSB- und WIFE Substructures

Auf diese Weise nimmt würde der Nearley Parser einen Family Record ohne Substructures und einen Family Record mit beliebig vielen Substructures (in diesem Fall HUSB- und WIFE Structures) als Eingabe akzeptieren. Sollen nun die weiteren Lines aus Listing 2.1 ebenfalls in die Grammatik aufgenommen werden, müssen Regeln für die Datentypen der Payloads des MARR-Events und der NCHI-Structure definiert werden. Die Anzahl der Kinder wird als *Integer* Datentyp kodiert, also ein Folge von Dezimalziffern. Nach der Gedcom7 Spezifikation dürfen *Integer* Werte nicht leer sein und führende Nullen sind erlaubt, sollten aber vermieden werden. Eine Regel für den Datentyp *Integer* kann also dargestellt werden als

```
digit   -> [0-9]
Integer -> digit:+
```

Listing 5.4: Nearley Regel für den Datentyp *Integer*

Für das MARR-Event, also die Hochzeit der Ehepartner der Familie, ist eine *DATE Structure* zum Festhalten des Datums der Hochzeit hinterlegt. Dieses Datum wird mit dem Datentyp *DateValue* kodiert, der im Gegensatz zum *Integer* wesentlich mehr Regeln umfasst. Ein *DateValue* kann auf vier verschiedene Weisen dargestellt sein:

1. *date*: Ein mehr oder weniger genau spezifiziertes Datum, z.B. “JULIAN 13 MAR 1998 BCE”
2. *datePeriod*: Ein Zeitintervall, dass von einem Startdatum bis zu einem Enddatum angegeben wird, z.B. “FROM 15 FEB 2001 TO 23 MAR 2001”

3. *dateRange*: Ein ungenaueres Zeitintervall, bei dem nur Grenzen angegeben werden, z.B. “BET 15 FEB 2001 AND 23 MAR 2001”
4. *dateApprox*: Eine Schätzung des Datums (ABT x: genaues Datum unbekannt, aber nahe x), z.B. “ABT 15 FEB 2001”

Diese Zusammenhänge ergeben die folgenden Nearley Regeln für die Definition des Datentyps *DateValue*:

```
DateValue  ->  (date | DatePeriod | dateRange | dateApprox):?

date       ->  (calendar D):?
               ((day D):? month D):?
               year
               (D epoch):?
datePeriod ->  ("FROM" D date D):? "TO" D date
dateApprox ->  ("ABT" | "CAL" | "EST") D date
dateRange  ->  "BET" D date D "AND" D date
           |   "AFT" D date
           |   "BEF" D date

calendar -> "GREGORIAN" | "JULIAN" | "FRENCH_R" | "HEBREW"
day      -> Integer
year     -> Integer
month    -> Tag
epoch    -> "BCE" | Tag

Tag      -> upperCaseLetter | digit | underscore
```

Listing 5.5: Nearley Regel für den Datentyp *DateValue*

Werden all diese Regeln zusammengefasst lässt sich die folgende Grammatik definieren, die den Family Record aus Listing 2.1 als Eingabe akzeptiert:

```
record_FAM_Substructs
    -> "0" %D %Xref %D "FAM" %EOL
    | record_FAM record_FAM_Substructs:+

record_FAM_Substructs
    -> "1" %D "HUSB" %D %Xref %EOL
    | "1" %D "WIFE" %D %Xref %EOL
    | "1" %D "NCHI" %D Integer %EOL
    | structure_MARR

structure_MARR
    -> "1" %D "MARR" %EOL
    | structure_MARR

structure_DATE
    -> "2" %D "DATE" %D DateValue %EOL
```

Listing 5.6: Nearley Grammatik für den Family Record aus Listing 2.1

Mit diesem Vorgehen können Nearley Regeln für alle Datentypen, Structures und Records definiert werden, die zu einer Grammatik für die Syntaxüberprüfung von Gedcom7 Dateinen zusammengesetzt werden können.

5.1.2 Nearley Postprozessor

Der Nearley Postprozessor für die Bibliothek *gedcom7.js* enthält 3 Funktionen:

joinAndUnpackAll():

Wie in Abschnitt 4.1.1 beschrieben, überführt ein *Nearley-Parser* jedes Zeichen, das mit einer Regel übereinstimmt, in ein Array. Bei komplexeren Grammatiken wie der Gedcom7 Spezifikation führt dies dazu, dass sehr viele Arrays innereinander verschachtelt werden, sodass schnell hohe Verschatlungsgrade erreicht werden. Ein Beispiel hierfür wäre der in Abschnitt 5.1.1 definierte Datentyp *DateValue*. Hier würde jeder Bestandteil eines DateValues in ein eigenes Array verschachtelt werden. Wird beispielsweise das Datum

13 MAR 1998 BCE

ohne Postprozessoren verarbeitet, wird das Array

[13, , [MAR, , [1998, , [BCE, ,]]]]

zurückgegeben, dass eine Weiterverarbeitung sehr umständlich macht. Daher wird der Postprozessor joinAndUnpackAll() implementiert, der über die JavaScript

Funktion *flat()* alle Elemente des Arrays rekursiv verkettet und anschließend über die Funktion *join()* zu einer Zeichenkette zusammenfügt. Wird dieser Postprozessor einem Datentyp wie *DateValue* zugewiesen, wird jedes syntaktisch korrekte Datum als Zeichenkette zurückgegeben und kann so direkt als LineValue für die weitere Verarbeitung verwendet werden. Die in Listing 5.5 definierte Regel würde sich ergeben zu

```
DateValue
    -> (date | DatePeriod | dateRange | dateApprox):?
        {% postprocessor.joinAndUnpackAll %}
```

Listing 5.7: Erweiterung der Nearley Regel für den Datentyp *DateValue*

createStructure():

Der Postprozessor createStructure() wird verwendet, um die gelesene Line mit Structure Informationen anzureichern. In der Nearley Regel wird die Line selbst, der Typ und die in der Gedcom7 Spezifikation definierte URI der Line und die Structures bei denen eine Kardinalitätsüberprüfung notwendig ist an den Postprozessor übergeben. Für den in Listing 5.6 Family Record ergibt sich der Postprozessoraufruf wie folgt:

```
record_FAM
    -> "0" %D %Xref %D "FAM" %EOL
    {% (line) => postprocessor.createStructure({
        line: line,
        uri: "g7_record_FAM",
        type: "FAM_RECORD",
        checkCardinalityOf: {
            "1_g7_FAM_HUSB":"0:1",
            "1_g7_FAM_WIFE":"0:1",
        }
    }) %}
```

Listing 5.8: Nearley Regel zum parsen eines Family Records mit Postprozessor

Im Parameter *checkCardinalityOf* werden die URIs alle Structures angegeben, bei denen eine Kardinalitätsüberprüfung notwendig ist und die in der Gedcom7 Spezifikation definierte Kardinalität als Wert zugewiesen. Kardinalitätsüberprüfungen sind bei allen Substructures erforderlich, die für die SUperstructure als notwendig

definiert wurden (1:1 und 1:M) oder für die eine Maximale Anzahl festgelegt ist (also 0:1 und 1:1). In der Funktion createStructure() werden alle Informationen abhängig vom übergebenen Type zusammengefasst und als JavaScript Objekt an den Parser zurückgegeben. Für einen Family Record ergibt sich die Funktion zu:

```
createStructure: (params) => {
    // create line object depending on type of line
    let lineObject = {};
    lineObject = {
        level: line[0],
        xref: line[2],
        tag: line[4],
        lineVal: '',
        EOL: line[5]
    };

    // return data object with structure information
    return {
        uri: params.uri,
        line: lineObject,
        type: params.type,
        lineValType: params.lineValType || null,
        superstructFound: false,
        substructs: [],
        checkCardinalityOf: params.checkCardinalityOf
    };
}
```

Listing 5.9: Funktion createStructure() für einen Family Record

addSubstructure():

Die zusammengesetzte Regel ergibt sich zu

```
record_FAM
    -> "0" %D %Xref %D "FAM" %EOL
    {% (line) => postprocessor.createStructure({
        line: line,
        uri: "g7_record_FAM",
        type: "FAM_RECORD",
        checkCardinalityOf: {
            "1_g7_FAM_HUSB":"0:1",
            "1_g7_FAM_WIFE":"0:1",
        }
    }) %}
```

Listing 5.10: Vollständige Nearley Regel zum parsen eines Family Records mit Substructures

5.2 Grammatik Generator

Der in Abschnitt 4.2 vorgestellte Grammatik Generator wird über die Klasse GrammarGenerator, wie in Abbildung 5.1 dargestellt, implementiert. Über die Klassenmethode *build(path)* kann eine Instanz von GrammarGenerator erstellt werden, der die Gedcom Grammatik an dem mit dem Parameter *path* spezifizierten Pfad erzeugt. Bei der Instanzerzeugung wird im ersten Schritt der Nearley-Header, der in der Nearley Datei *NearleyHeader.ne* spezifiziert ist, eingelesen und als Instanzvariable in Form einer Zeichenkette gespeichert. Dieser Nearley-Header stellt den obersten Eintrag jeder Nearley-Datei dar, die vom GrammarGenerator erzeugt wird und enthält die Include-Statements für Datentypen, Postprozessoren, etc. und den Aufruf des Moo-Lexers. Anschließend werden die Gedcom Grammatik Definition, die in Form von JavaScript Objekten gespeichert sind, gelesen und gespeichert. Anschließend kann diese Gedcom Grammatik Definition mit der Funktion *generateGrammar()* in Nearley-Dateien überführt werden, die dann zu Nearley Parsern kompiliert werden können. In den folgenden Kapitel wird auf diese Schritte im Detail eingegangen.

5.2.1 Definition der Grammatik

Die Definition der Gedcom Grammatik erfolgt in Form von JavaScript Objekten. Für jede Structure, die in der Gedcom7 Spezifikation definiert ist, wird eine Grammatik Definition erstellt. Folgende Parameter sind in diesen Objekten hinterlegt:

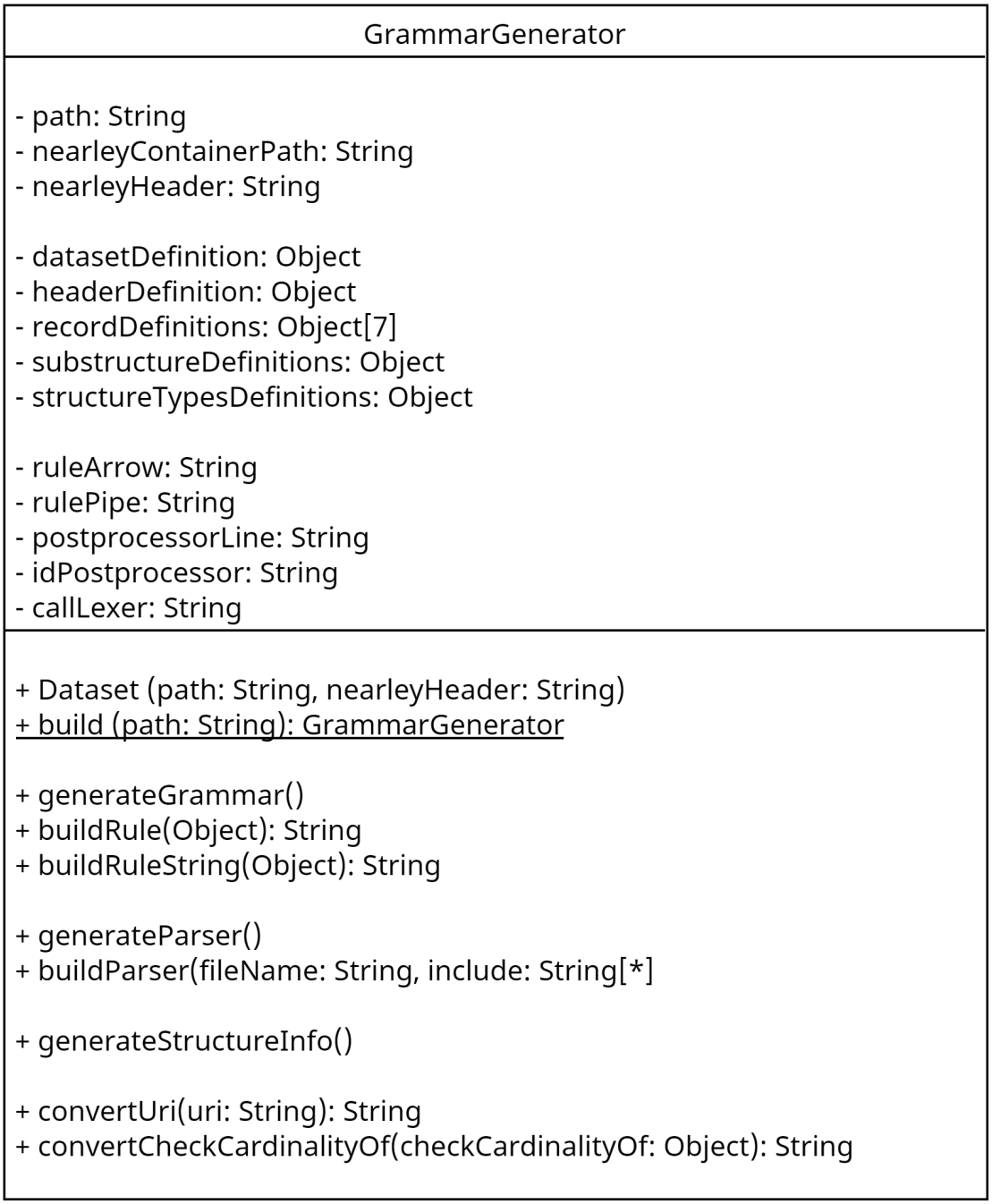

Abbildung 5.1: UML Klassendiagramm GrammarGenerator

URI	Die in der Gedcom7 Spezifikation für diese Structure hinterlegte URI.
lineType	Der Typ der Line einer Structure (hier wird beispielsweise hinterlegt, ob die Structure einen Cross-Reference-Identifier enthält oder auf andere Structuren verweisen darf).
Info	In der Info wird ein Informationstext zu jeder Strucutre hinterlegt. Dieser kann verwendet werden um bei Verwendung der Bibliothek dem Benutzer Informationen über die Bedeutung der Structures zukommen zu lassen.
Level	Die Levels unter denen die Structure in einer Gedcom7

Die Definition für einen Family Record sieht wie folgt aus. Anders als bei den Beispielen aus Abschnitt 5.1 bei denen nur ein Teil der Substructures betrachtet wurde, handelt es sich hierbei um eine vollständige Definition:

```
{
    uri: 'g7:record-FAM',
    lineType: lineTypes.FAM_RECORD,
    info: 'Structure Info coming soon!',
    level: [0],
    tag: 'FAM',
    substructs: {
        'g7:RESN': '0:1',
        FAMILY_ATTRIBUTE_STRUCTURE: '0:M',
        FAMILY_EVENT_STRUCTURE: '0:M',
        NON_EVENT_STRUCTURE: '0:M',
        'g7:FAM-HUSB': '0:1',
        'g7:FAM-WIFE': '0:1',
        'g7:CHIL': '0:M',
        ASSOCIATION_STRUCTURE: '0:M',
        'g7:SUBM': '0:M',
        LDS_SPOUSE_SEALING: '0:M',
        IDENTIFIER_STRUCTURE: '0:M',
        NOTE_STRUCTURE: '0:M',
        SOURCE_CITATION: '0:M',
        MULTIMEDIA_LINK: '0:M',
        CHANGE_DATE: '0:1',
        CREATION_DATE: '0:1'
    }
}
```

Listing 5.11: Grammatik Definition eines Family Records

5.2.2 Grammatikgenerierung mit generateGrammar()

Nach dem in Abschnitt 5.2.1 beschriebenen Vorgehen werden Grammatik Definitionen für alle Structuretypes, Substructures und Records, sowie für das gesamte Dataset erstellt. Mit der Funktion *generateGrammar()* werden diese eingelesen,

in eine nearley-konforme Zeichenkette konvertiert und anschließend in Form einer Nearley Datei (.ne) gespeichert. Die Regeln werden mit der Funktion *buildRuleString()* erzeugt und mit den in der Klasse GrammarGenerator definierten Building-Konstanten zusammengefügt. Ein Beispiel für eine solche Konstante ist der *ruleArrow* der zur Definition einer Regel verwendet wird und als Zeichenkette "`\n\t` –>" kodiert ist. Die so erzeugten Nearley Dateien sind somit einfach lesbar und liegen in der in Abschnitt 5.1.1 Form vor. Alle so erstellten Nearley Dateien werden im mit *path* spezifizierten Pfad im Verzeichnis "*path/nearley/*" abgelegt.

5.2.3 Parsergenerierung mit generateParser()

Mit der Funktion *generateParser()* werden die Nearley Grammatiken für alle Records und das gesamte Dataset zu Nearley Parsern kompiliert. Dazu stellt Nearley die Funktion

nearleyc *inputPath* -o *outputPath*

bereit, mit eine Nearley Datei eingelesen und im spezifizierten Pfad kompiliert werden kann. Das Erstellen von Parsern für die Records ist notwendig, da die Nearley Parser für die Syntaxüberprüfung nach Änderung eines Records verwendet werden. Würde nur ein allgemeiner Dataset-Parser erstellt werden, müsste nach jeder Änderung das komplette Dataset überprüft werden, obwohl nur ein Record verändert wurde.

Im ersten Schritt der Funktion *generateParser()* werden die Include-Statements vorbereitet, die z.B. die Definition der Datentypen enthalten. Anschließend wird für die Record- und Dataset-Grammatiken die Funktion *buildParser()* aufgerufen, die in Listing 5.12 dargestellt ist. Hier werden die mit *generateGrammar()* erzeugte Grammatik, die Include-Statements und der Nearley Header in einer Container Datei *NearleyContainer.ne* zusammengefügt. Diese Container Datei wird anschließend zu dem entsprechenden Parser kompiliert und im mit *path* spezifizierten Pfad im Verzeichnis "*path/parser/*" abgelegt.

```
// build nearley-file with include statements and NearleyHeader inside of NearleyC
async buildParser (fileName, include) {
    // string representation of the grammar to be compiled
    let fileStr = '';

    // add given include statements
    for (const file of include) {
        fileStr += `@include "../grammar/nearley/${file}"\n`;
    }

    // add nearley header
    fileStr += this.nearleyHeader;

    // overwrite content of NearleyContainer file
    await fs.writeFile(this.nearleyContainerPath, fileStr);

    // read grammar of given file
    const grammar = await fs.readFile(`${this.path}nearley/${fileName}.ne`, { enco

    // append grammar to NearleyContainer file
    fs.appendFile(this.nearleyContainerPath, grammar);

    // compile composed NearleyContainer.ne file to nearley parser
    await exec(`npx nearleyc ${this.nearleyContainerPath} -o ${this.path}parser/${
}
```

Listing 5.12: Funktion buildParser() des Grammatik Generators

5.3 Gedcom Struktur

Die Struktur einer Gedcom7 Datei wird in der Bibliothek *gedcom7.js* mit Hilfe der Klasse DATASET abgebildet, die alle Gedcom Structures verwaltet, die in Form der gleichnamigen Klasse STRUCTURE verwaltet werden. Structures werden in *gedcom7.js* entweder als allgemeine Instanz der Klasse STRUCTURE (z.B. eine HUSB-Structure), als RECORD (z.B. ein Family Record) oder als DATATYPE STRUCTURE (z.B. eine DATE-Structure) repräsentiert.

5.3.1 Klasse *Structure*

Die Klasse STRUCTURE ist die Überklasse aller Klassen zur Structure-Verwaltung, d.h. Records und Datatyp Structures erben alle Methoden und Eigenschaften von STRUCTURE. Diese Methoden und Eigenschaften sind in Abbildung 5.2 abgebildet.

Eine Instanz der Klasse STRUCTURE hält alle Informationen über eine Line bereit (URI, Level, Tag, Xref, lineValue, Typ des LineValues, EOL-Zeichen). Außerdem werden Referenzen zu allen Substructures, der Superstructure, dem Record mit dem die Structure assoziiert wird, sowie zum Dataset in dem die Structure enthalten ist bereitgestellt. Außerdem übernimmt die Klasse STRUCTURE vier zentrale Aufgaben zur Verwaltung von Gedcom7 Informationen, die von der Klasse DATASET angestoßen werden können:

1. Finden von Substructures

Eine der wichtigsten Aufgaben der Klasse STRUCTURE ist es, eigene Substructures zu suchen und zu finden. Dazu werden die Methoden *getSubstructuresByUri*, *getSubstructuresByTag* und *getSubstructuresByLineVal* bereitgestellt, die alle Sbstructures zurückgeben, die einem Suchkriterium genügen, das abhängig von der Methode eine Gedcom7 URI, ein Gedcom7 Tag oder einen LineValue darstellen. Über den Parameter *recursive* kann spezifiziert werden, ob nur direkte Substructures (also mit einem um 1 inkrementierten Level) gesucht werden sollen oder ob ebenfalls alle Substructures von Substructures rekursiv durchsucht werden sollen. Sollen einfach alle Substructures einer Structure ohne Suchkriterium zurückgegeben werden, kann die Methode *getSubstructures* verwendet werden.

2. Hinzufügen von Substructures

asd

3. Entfernen von Substructures

asd

4. Ändern des Payloads

asd

5.3.2 Klasse *Record*

5.3.3 Klasse *Family*

5.3.4 Klasse *Dataset*

5.4 Gedcom Parser

Abbildung 5.2: UML Klassendiagramm Structure

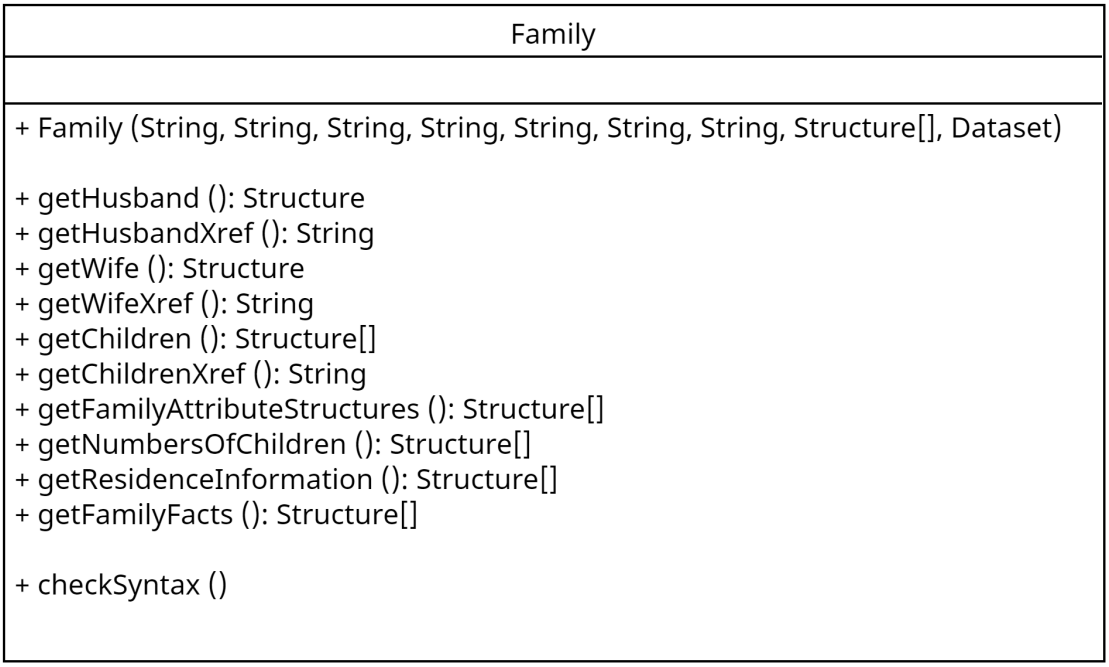

Abbildung 5.3: UML Klassendiagramm Record

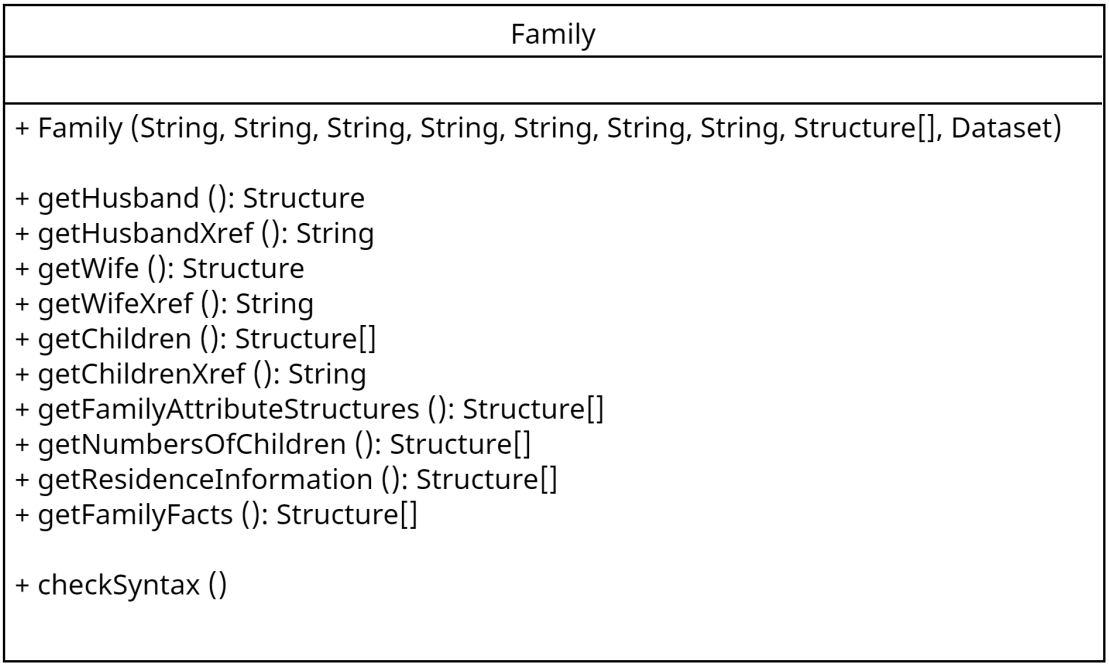

Abbildung 5.4: UML Klassendiagramm Family

Dataset
- header: Object - trlr: Object - records: Object[] - xrefMap: Object - substructsWithXrefDatatype: Object - BOM: String - BOMset: boolean - EOL: Object - multipleEOLCharacters: boolean
+ Dataset (Structure, Structure[], Structure, boolean) + createEmptyDataset (String): Dataset + createStructure (Object, Structure, Structure, boolean): Structure + createStructureFromStructureParameter (Object, String): Structure + addRecord (Object): Structure + removeRecord (Structure, boolean?) + addStructure (Structure, Structure, Structure) + removeStructure (Structure, bolean?) + searchAndRemoveEmptyStructure (Structure) + checkSyntax () + getEOL () + convertUri (String): String + addXrefToXrefMap (Structure) + addSubstructWithXrefDatatype (Structure) + removeSubstructWithXrefDatatype (Structure) + checkForNotDefinedXrefs () + isXrefDefined (String, Structure) + getRecordByXref (String) + getRecrodsByConstructor (Object) + getHeaderRecord (): Structure + getFamilyRecords (): Structure + getIndividualRecords (): Structure + getMultimediaRecords (): Structure + getRepositoryRecords (): Structure + getSharedNoteRecords (): Structure + getSourceRecords (): Structure + getSubmitterRecords (): Structure + toString ()

6

Zusammenfassung und Ausblick

In dieser Abhandlung wird unsere Bibliothek gedcom7.js als Hauptthema präsentiert. Diese Bibliothek enthält als Hauptkomponenten einen Parser, Grammatiken, einen Grammatik-Generator und die Structure-Klassen und Methoden. Diese können verwendet werden, um Gedcom7-Dateien unter Verwendung von node.js zu lesen, zu verarbeiten, zu manipulieren, zu erstellen und zu schreiben.

Bei der Ausarbeitung und Implementierung wurde ein besonderer Fokus auf die konforme Handhabung von Gedcom7-Dateien gelegt. Durch die Grammatikprüfung beim Einlesen eines Datasets wird sichergestellt, dass die Datei den Gedcom7-Spezifikationen entspricht. Die Grammatikprüfung wird auch nach Änderungen durch die Structure-Klassen-Methoden durchgeführt, um sicherzustellen, dass die Daten nach der Manipulation noch den Spezifikationen entsprechen.

Das Kernmodul der Grammatikprüfung ist der GrammarGenerator. Mit seinem spezifikationsnahen Interface kann der GrammarGenerator redundante Strukturen kompakt abbilden und daraus effizient Nearley-Grammatiken erstellen, aus denen JavaScript-Parser generiert werden. Der GrammarGenerator greift hierbei auf die URI-Bezeichner der Gedcom-Spezifikation zurück.

Dank des modularen Aufbaus des GrammarGenerators können auch Features wie Gedcom-Extensions implementiert werden, bei denen eigene "Tags"definiert werden können. Nach der Erweiterung des GrammarGenerators können neue grammatikbasierte Parser erzeugt werden.

Insgesamt bietet die Bibliothek gedcom7.js eine robuste und spezifikationskonforme Möglichkeit, Gedcom7-Dateien in JavaScript zu verarbeiten. Die Grammatikprüfung und der GrammarGenerator sind besonders nützliche Features, die die Erstellung und Manipulation von Gedcom7-Dateien erleichtern und für eine Weiterentwicklung des Projekts eine sehr solide Grundlage bietet.

Literaturverzeichnis

Ahn. Ahnenforschung: *Genealogie*. Abgerufen am 02.02.2022 von `https://www.ahnenforschung.de/themen/genealogie/`.

Cha. Chandra, Kartik: *Documentation: nearley.js*. Abgerufen am 10.02.2022 von `https://nearley.js.org/`.

Fam22. Family History Department: *The FamilySearch GEDCOM Specification 7.0.11*. The Church of Jesus Christ of Latter-day Saints, 15 East South Temple StreetSalt Lake City, UT 84150 US, 7.0.11 Auflage, November 2022.

Gen. Genealogists, Society Of: *Genealogy or Family History*. Abgerufen am 02.02.2022 von `https://www.sog.org.uk/learn/hints-tips/genealogy-or-family-history`.

Rad. Radvan, Tim: *Documentation: moo.js*. Abgerufen am 10.02.2022 von `https://github.com/no-context/moo`.

A

Glossar

GEDCOM	GEnealogical Data COMunication
URI	Uniform Resource Identifier

B

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Seminararbeit ohne fremde Hilfe verfasst und nur die im Literaturverzeichnis angegebenen Quellen verwendet habe.

Datum	Unterschrift der Kandidatin/des Kandidaten