



Bachelorarbeit

JSON to Property Graph Mapping

VORGELEGT VON:

Kazzaz Abdulrahman

MATRIKEL-NR.: 218203379

EINGEREICHT AM:

09. September 2021

BETREUER:

Dr.-Ing. Holger Meyer

Dipl.-Inf. Alf-Christian Schering

Inhaltsverzeichnis

1 Einleitung	1
1.1 Motivation	1
1.2 Kurze Vorstellung	2
1.3 Problemstellung und Zielsetzung	3
1.4 ISEBEL-Projekt und WossiDlA-System	5
2 Grundlagen	7
2.1 Property Graph-Konzepte	7
2.2 JSON-Konzepte	12
3 Stand der Technik	18
3.1 JsonPath-basierter Ansatz	20
3.2 X2G-basierter Ansatz	22
3.3 JSON-XML-Ansätze	26
3.3.1 XSLT-basierter-Ansatz	26
3.3.2 XML.Serializer-basierter-Ansatz	29
3.3.3 XML.toString-basierter-Ansatz	31
4 Konzept	32
4.1 Anforderungen	33
4.2 Ablaufdiagramm	34
4.3 J2XML	36
4.3.1 JSON-Daten holen	38
4.3.2 JSON Wohlgeformtheit prüfen	40
4.3.3 JSON zu XML konvertieren	42
4.3.4 XML Wohlgeformtheit prüfen	45
4.3.5 Zu XML-Wohlgeformtheit anpassen	48
4.3.6 XML-Dokument zu X2G übergeben	51
4.3.7 JSONPath zu XPath konvertieren	52
4.4 X2G	54
4.4.1 Importer	55
4.4.2 Evaluator	56
4.4.3 Regelsprache	57

<i>Inhaltsverzeichnis</i>	ii
4.4.4 Graph Erzeuger	63
4.4.5 Exporter	65
4.5 Beispiel Szenario	67
5 Zusammenfassung	72
6 Ausblick	73
Verzeichnisse	73
Abbildungsverzeichnis	74
Literaturverzeichnis	I

Abstract

The aim of this work is the implementation of a tool for the conversion of JSON data in Property Graph data. The tool is implemented in Java. The focus of this work is the analysis of JSON data and the creation of property graphs. The tool allows you to create graphs based on user input, which means that custom rules allow users to control the transformation. Users can decide which nodes and edges from the JSON files should be represented in graphs. Users can select attributes and content from the JSON files. The generated property graph can be saved in a CSV file or in a GEXF file. The work includes the description of the tool structure and the description of the most important parts of the functionality (filtering JSON data, nodes and edges generating and then exporting as a CSV file).

Ziel dieser Arbeit ist die Implementierung eines Tools zur Umwandlung von JSON-Daten in Property Graph-Daten. Das Tool ist in Java-Sprache implementiert. Die Schwerpunkte dieser Arbeit sind die Untersuchung von JSON-Daten sowie die Erstellung von Property Graphen. Das Tool ermöglicht das Erstellen von Graphen basierend auf Eingaben von Benutzern, das bedeutet durch Benutzerdefinierte Regeln können Benutzer die Transformation steuern. Benutzer können entscheiden, welche Knoten und Kanten aus den JSON-Dateien in Graphen dargestellt werden sollen. Benutzer können auch Attribute und Inhalte aus JSON-Dateien auswählen. Das erzeugte Property Graph kann wieder in einer CSV-Datei oder in einer GEXF-Datei gespeichert werden. Die Arbeit umfasst die Beschreibung der Tool-Struktur sowie die Beschreibung der wichtigsten Teile der Funktionalität (Filtern von JSON-Daten, Knoten und Kanten Generieren und Anschließend als CSV-Datei Exportieren).

1 Einleitung

1.1 Motivation

In den letzten Jahren hat sich die Forschung in Graphen-Gebiet rasant entwickelt. Ethnologe und Forscher*innen versuchen, Daten zu analysieren und durchzusuchen, die jedoch in verschiedenen Datenformate wie JSON gespeichert sind. Die Verarbeitung von Daten in JSON-Dateien oder bestimmter Aspekte ist kompliziert, da der Zugriff auf ihre internen Daten nicht einfach ist. Außerdem ist die Verknüpfung von mehreren Dokumenten schwierig, sodass Forscher*innen Schwierigkeiten mit Analysieren, Untersuchen, sowie das Durchsuchen von einer massiven gesammelten Menge von JSON-Dateien begegnen, und daher fehlt die Genauigkeit der Analyse von diesen Daten und die Arbeit an denen wird stark eingeschränkt.

Property Graphen bieten hingegen mit ihren Knoten, Kanten und Eigenschaften die Möglichkeit für den Forscher*innen, die gesammelte Daten zu visualisieren und erleichtert ihnen das Durchsuchen und die Analyse von Daten sowie die Möglichkeit, neue Daten zu gewinnen. Außerdem können Forscher*innen die verschiedene Graph-Mining-Algorithmen auf der Graph-Daten umsetzen. Aus diesen Gründen und den vielen Property Graphen-Vorteilen entsteht den Bedarf danach, Tools zu entwickeln, die JSON-Daten zu Property Graph-Daten transformieren.

Der im Laufe dieser Arbeit beschriebener Ansatz (J2G) wurde für die Umwandlung von JSON-Daten zu Graph-daten entwickelt, um Ethnologe und Forscher bei der Daten-Verarbeitung und bei der Visualisierung dieser Daten zu unterstützen.

1.2 Kurze Vorstellung

Heutzutage ist der Analyse und Visualisierung von Daten ein zentrales Thema in der Forschung geworden. Viele Projekte und Systeme beschäftigen sich mit Wissen-Sammlung aus verschiedenen Datenmodelle und Formate und benutzen Graphen einerseits um eine Visualisierung über dieses Wissen darzustellen, andererseits um verschiedene Graph-Mining Algorithmen auf Graph-Daten umzusetzen. Forscher*innen können von diesen Systemen profitieren, indem sie die visualisierte Daten als Property Graphen oder HyperGraphen nutzen, um ihre Analyse auf die Daten anzuwenden. Dieser Nutzen ist nicht nur auf Forscher beschränkt, sondern auch Leser können davon profitieren, da Graphen ein besseres Verständnis von wissen anbietet.

Das ISEBEL-Projekt ist einer der Systeme, das sich mit Wissen-Sammlung befasst. Es zielt darauf ab, eine internationale Suchmaschine zu entwickeln, die Daten aus Volksglaubensdatenbanken sammelt. Es enthält viele digitale Sammlungen, die neueste davon ist die WossidiA von Richard Wossidia aus Mecklenburg. [MSS14]

Ein Teil des Projekts befasst sich mit Data und Graph-Mining [AW10], [CH06], um häufige Muster zu finden, um entweder das Wissen besser zu verstehen oder daraus neues Wissen herauszuholen. Daher werden die Geschichtsdaten über OAI-PMH gesammelt und in einem CKAN-Repository verwaltet. Diese Daten können als JSON-Dokumente von diesem Repository über eine REST-API sowie von der REST-API des WossiDiA-Systems abgerufen werden.

Diese Arbeit bietet eine neue Technik im Rahmen des ISEBEL-Projektes, um Forscher*innen bei der Analyse, dem Durchsuchen, und der Visualisierung bestimmter Aspekte in den gesammelten JSON-Daten mithilfe des Graphenparadigmas und der entwickelten Algorithmen zu unterstützen, die nicht nur auf WossiDiA-Graphdaten be-

schränkt sind. Die entwickelte Technik sorgt dafür, die gesammelten JSON-Daten allgemein in Property Graphen zu transformieren. Die Transformation wird durch benutzerdefinierte Regeln gesteuert, die eine Abbildung von JSON-Schema-Konzepten auf Property Graph-Modell-Konzepte beschreiben.

Mit dieser Technik sind Forscher*innen in der Lage, Regeln zu definieren, die Attribute und Inhalte aus JSON-Dateien auswählen und aus dieser Auswahl werden Knoten und Kanten sowie Beschriftungen erzeugt. Außerdem unterstützt diese Technik die automatische Generierung von Property Graphs auf der Grundlagen eines vom Benutzer vorgegebenes Regelsätze. Nachdem Generierung von Property Graphs bietet die Technik die Speicherung von erzeugten Property Graphs als GEXF-Dateien.

Diese Arbeit ist wie folgt aufgebaut: Zunächst wird im Rest dieses Kapital das gegebene Problem sowie Ziel dieser Arbeit vorgestellt, welche Aspekte in dieser Arbeit betrachtet sind, und welche Probleme durch die neue Technik gelöst werden soll in Abschnitt 1.3. Ein Überblick auf Das ISEBEL-Projekt und WossiDiA-System ist im Abschnitt 1.4. Kapital 2 zeigt die Struktur von Property Graphs und JSON-Schema und wie sie aufgebaut sind und welche Konzepte sie haben. Die bereits vorhandenen Techniken sind im Kapital 3. Das Konzept ist im Kapital 4 zu sehen.

1.3 Problemstellung und Zielsetzung

Ziel dieser Arbeit ist die Entwicklung eines Tool zur Umwandlung von JSON-Daten in Property Graph-Daten, um Forscher bei der Analyse, dem Durchsuchen und der Visualisierung bestimmter Aspekte in den gesammelten Daten zu unterstützen. Durch dieses Tool können die JSON-Dateien allgemein in Property Graphen umgewandelt und visualisiert werden, wobei die Transformation durch benutzerdefinierte Regeln gesteuert

werden kann. Mit benutzerdefinierten Regeln ist gemeint, dass die Eingaben von Benutzer berücksichtigt sind und den Eigenschaftsgraph wie erwünscht erstellt werden soll, indem der Benutzer die Attribute und Inhalte aus den JSON-Dateien auswählen kann und aus dieser Auswahl werden Knoten, Kanten, und Beschriftungen erzeugt. Das Filtern von JSON-Dokumenten oder von anderen Datenformat sowie die Erzeugung von gewünschten Property Graphs aus den gefilterten Dokumenten ist schwierig und begegnet viele Probleme, diese Probleme wurde durch im Rahmen dieser Arbeit entwickelten Technik betrachtet und gelöst. Die Probleme, die bei der Entwicklung bemerkt wurden, sind die folgenden Probleme:

- Welche Attribute und Inhalte werden aus den JSON-Dokumenten extrahiert.
- wann soll ein neuer Knoten/Kante erzeugt werden.
- wie werden die Duplikate entfernt, wenn ein Knoten erzeugt wird, der bereits im Graph existiert.
- wie werden die Beziehungen(Kanten) zwischen den Knoten erstellt.
- wie werden die Eigenschaften der Knoten oder Kanten zugeordnet.
- was passiert, wenn verschiedene Knoten oder Kanten gemeinsame Eigenschaften haben.
- wenn mehrere Knoten viele Beziehungen zwischen einander haben, wie wird das im Graph dargestellt, wenn der Property Graph das erlaubt.

Schließlich wird der gewählte Property Graph generiert und kann als CSV-Datei gespeichert werden. Der Prozess zur Beantwortung dieser Fragen wird im nächsten Kapitel vorgestellt.

1.4 ISEBEL-Projekt und WossidA-System

Wie bereits erwähnt zielt das ISEBEL-Projekt darauf ab, eine internationale Suchmaschine zu entwickeln, die in der Lage ist, Daten aus Volksglaubensdatenbanken zu sammeln. Das anfängliche Projekt konzentriert sich aber auf Glaubenslegenden [UDNS16], die in den drei bekannten digitalen Sammlungen von Evald Tang Kristensen aus Dänemark (etkspase), Richard Wossidlo aus Mecklenburg (wossidia) und mehreren Sammlern und Erzählern aus den Niederlanden (verhaalenbank) gefunden wurden.

Das WossidA-System [MSS14] ist eine der von ISEBEL geernteten Datenbanken und hat zum Ziel, den Nachlass des mecklenburgischen Volkskundlers Richard Wossidlo (1859-1939) im Internet zu veröffentlichen. Diese Sammlung umfasst große Mengen von handschriftlichen Notizen, die nahezu alle Bereiche der Volkskultur und der Lexik der niederdeutschen Sprache abdecken. Eine Besonderheit von Wossidlos Sammlung ist ein hochkomplexer Thesaurus, der dem Wissenschaftler als semantisches Netz für seine umfangreichen Notizen und Korrespondenzen gedient hat. Dieser Thesaurus wird aus der Perspektive der europäischen Ethnologie überarbeitet und mit einem Standardregelwerk kompatibel gemacht. Ein Beispiel von WossidA-Webanwendung ist in der Abbildung 1.1 zu sehen.

Das WossidA-System verwendet typisierte, gerichtete Hypergraphen [MSH17] zur Darstellung der Sammlungen von Richard Wossidlo. Hypergraphen sind ein ungewöhnlich geeignetes Modell für komplexe kulturgeschichtliche Phänomene wie die Volkskunde. Im Hypergraphen-Modell des Wossidlo Digitalarchiv können Knoten Geschichten, Erzähler, Orte, Sammler, benannte Entitäten, Schlüsselwörter und Akteure der Geschichte darstellen

Das Sammeln aller zugehörigen Informationen aus dem Papierbelegnetzwerk erfolgt durch Anwendung graphbasierter Operationen wie inhalts-, struktur- und typbasierte Graphfilterung in Kombination mit Hyperedge-Kontraktion, Aggregation und Zusammenfassung. Der resultierende Graph kann dann auf ein logisches Dokument abgebildet werden.

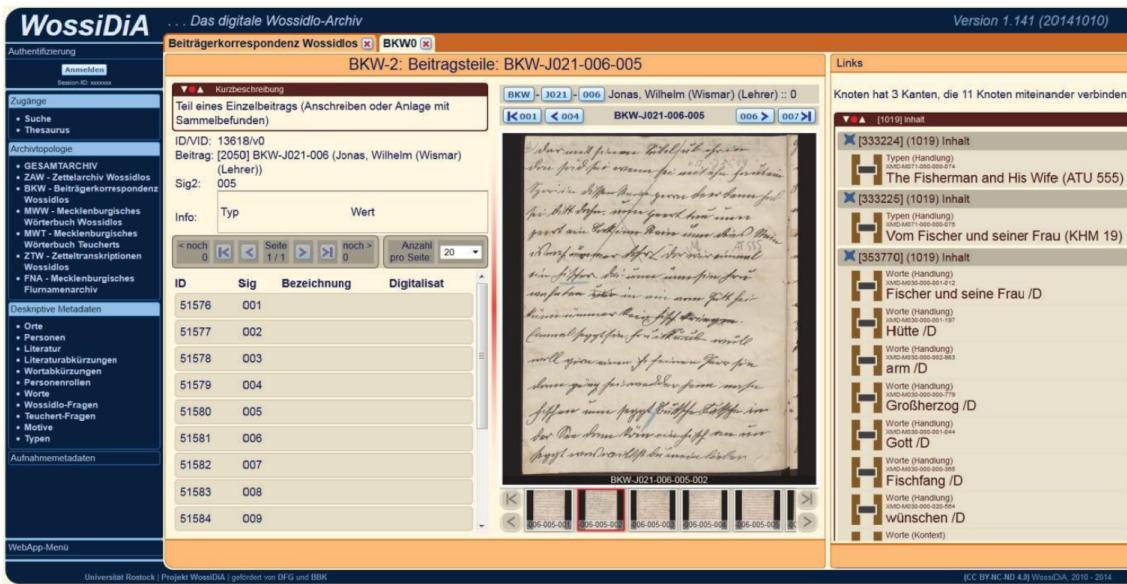


Abbildung 1.1: Beispiel von WossiDiA-Webanwendung, Quelle: [MSS14]

2 Grundlagen

2.1 Property Graph-Konzepte

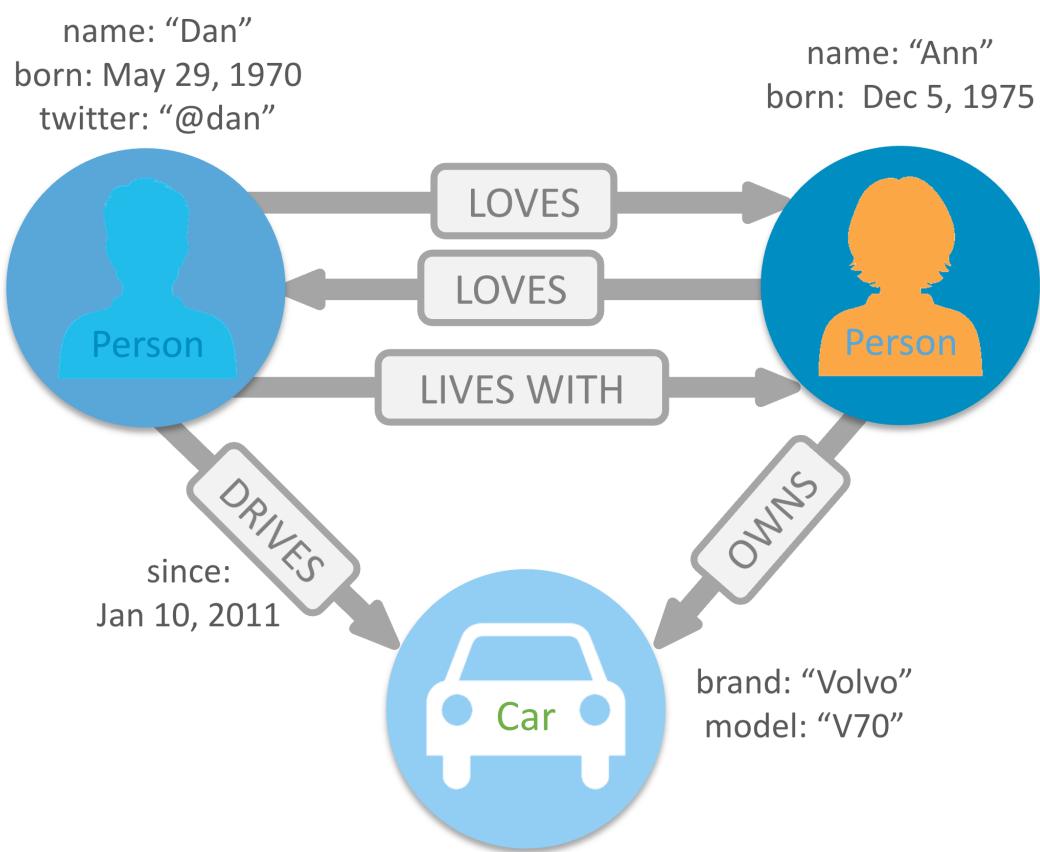


Abbildung 2.1: Beispiel (1) von Property Graph, Quelle: [neob]

Eigenschaftsgraphen (Property Graphs auf Englisch) sind speziell Fall von Graphen und dienen zur Visualisierung von Daten aus verschiedenen Daten-Modelle, sie werden verwendet, um Daten besser analysieren zu können [CYM20] [TAS⁺19].

Der Begriff Eigenschaftsgraph wurde von Rodriguez und Neubauer eingeführt [RN10]. Es ist möglich, viele Variationen von der grundlegenden Definition zu finden [BFVY18] [AAB⁺17] [Har14] [TP18], die meisten von ihnen beziehen sich auf die Unterstützung mehrerer Beschriftungen für Knoten und Kanten wie in der Abbildung 2.1 oder das Auftreten von mehrwertigen Eigenschaften. Im allgemein ist ein Eigenschaftsgraph ein gerichteter, beschrifteter Mehrfachgraph mit der besonderen Eigenschaft, dass jeder Knoten oder jede Kante einen Satz von Eigenschaft-Werte-Paare haben kann. Der Eigenschaftsgraph hat vier wesentliche Konzepte [neoa], [gra] (Knoten, Kanten, Beschriftungen, Eigenschaften), wobei jedes Konzept eigene Merkmale hat.

Knoten:

- Knoten beschreiben Entitäten oder Elemente.
- Knoten können null oder mehr Beschriftungen (Label) haben, um zu klassifizieren, welche Art von Knoten es sich handelt oder so zu sagen, den Knoten-Typ zu bestimmen.
- Knoten werden auch als Scheitelpunkte oder Punkte bezeichnet.
- Knoten können Eigenschaften (Schlüssel-Wert-Paare) aufweisen, die sie weiter beschreiben.

Kanten:

- Kanten (Beziehungen) beschreiben eine Verbindung zwischen einem Quellknoten und einem Zielknoten.
- Kanten haben immer eine Richtung, die als Pfeilspitze bezeichnet wird.
- Kanten können auch Null oder mehr Beschriftungen (Typ) haben, um zu klassifizieren, welche Art von Beziehung sie bestimmen.

- Kanten werden auch als Beziehungen, Verknüpfungen oder Linien bezeichnet.
- Kanten können Eigenschaften (Schlüssel-Wert-Paare) aufweisen, die sie weiter beschreiben.

Beschriftungen (Labels):

- Beschriftungen sind sozusagen den Typ für Knoten und Kanten
- Beschriftungen machen die Knoten und Kanten eindeutig.

Eigenschaften (Properties):

- Eigenschaften sind Schlüssel-Wert-Paare, und sie dienen zur weiteren Beschreibung von Knoten und Kanten.
- Jeder Eigenschaft kann beliebige Datentypen haben wie String, Integer, Double, Boolean ... etc.

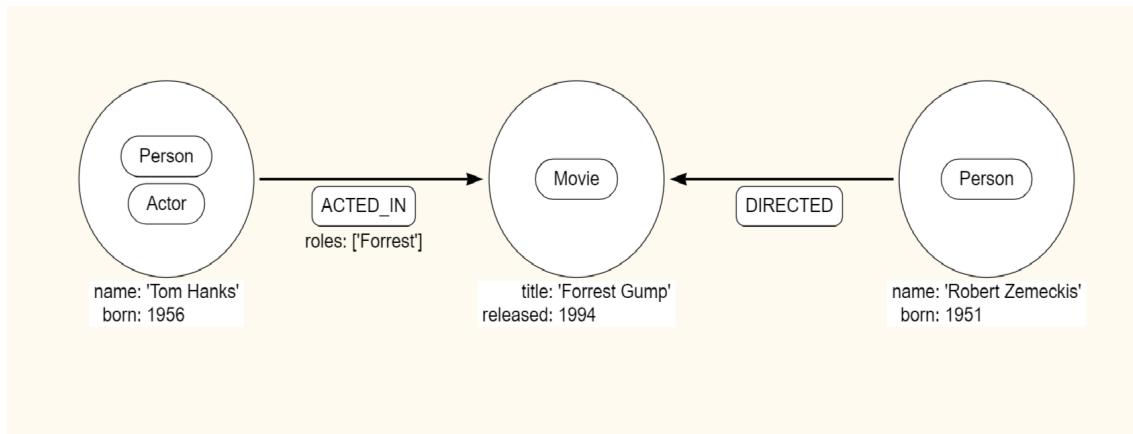


Abbildung 2.2: Beispiel (2) von Property Graph [neoa]

Die Abbildung 2.2 zeigt ein minimales Graph mit drei Knoten und zwei Kanten, die jeweils beschriftet sind. Dieser Graph beschreibt einen Film und dazugehörigen Elementen und deren Beziehungen. Die linkere Knoten hat zwei Beschriftungen (Person, Actor)

und zwei Attribute (Eigenschaften), deren Attribut-werte (Name und Born) von Typ String und Integer sind. Der mittlere Knoten hat eine einzige Beschriftung (Movie) und zwei Attribute (Title und Released). Der rechte Knoten hat die Beschriftung (Person) und die gleiche Attribute wie der linkere Knoten mit anderen Werten. Die linkere Kante hat den Typ (Acted in) und einen einzigen Attribut, diese Kante verbindet die linkeren Knoten (Quellknoten) mit der mittleren Knoten (Zielknoten) durch einen Pfeilspitze, mit dieser Verbindung wird beschrieben, dass der Schauspieler "Tom Hanks" den Rolle "Forrest's Film "Forrest Gump" gespielt hat. Die rechte Kante hat den Typ (Directed) und hat keine Attribute. diese Kante ist für die Verbindung zwischen dem rechten Knoten und dem mittleren Knoten. Mit dieser Verbindung wird beschrieben, dass "Robert Zemeckis" den Film "Forrest Gump" gedreht hat.

Jeder Eigenschaftsgraph kann mit den Graph-Programme visualisiert werden, dafür müssen Graph-Daten in Graph-Dateiformat umgewandelt werden [tlc], da die Graph-Programme nur Graph-Dateiformate akzeptieren. Es gibt viele Dateiformate zum Austausch von Graphdaten, diese Formate werden in der entsprechenden Graph-Software eingelesen und visualisiert.

Beispiele von Graphdaten-Formaten sind:

- GRAPHML (GraphML Format)
- GEXF (Graph Exchange XML Format)
- Graphson
- CYS
- XGMML
- CSV

- GRAM
- GML etc....

Der Gephi Software wurde entwickelt, um Graphen darzustellen, Gephi unterstützt viele Dateiformate [sub] wie:

- GEXF.
- GraphML.
- CSV.

Während die Framework Cytpscape die folgenden Dateiformate unterstützt:

- CYS
- XGMML

Die Graphdaten werden in Graphdatenbanken gespeichert. Zum Beispiel speichert die Graphdatenbank Neo4j die Graphdaten in GRAM-Dateiformat.

2.2 JSON-Konzepte

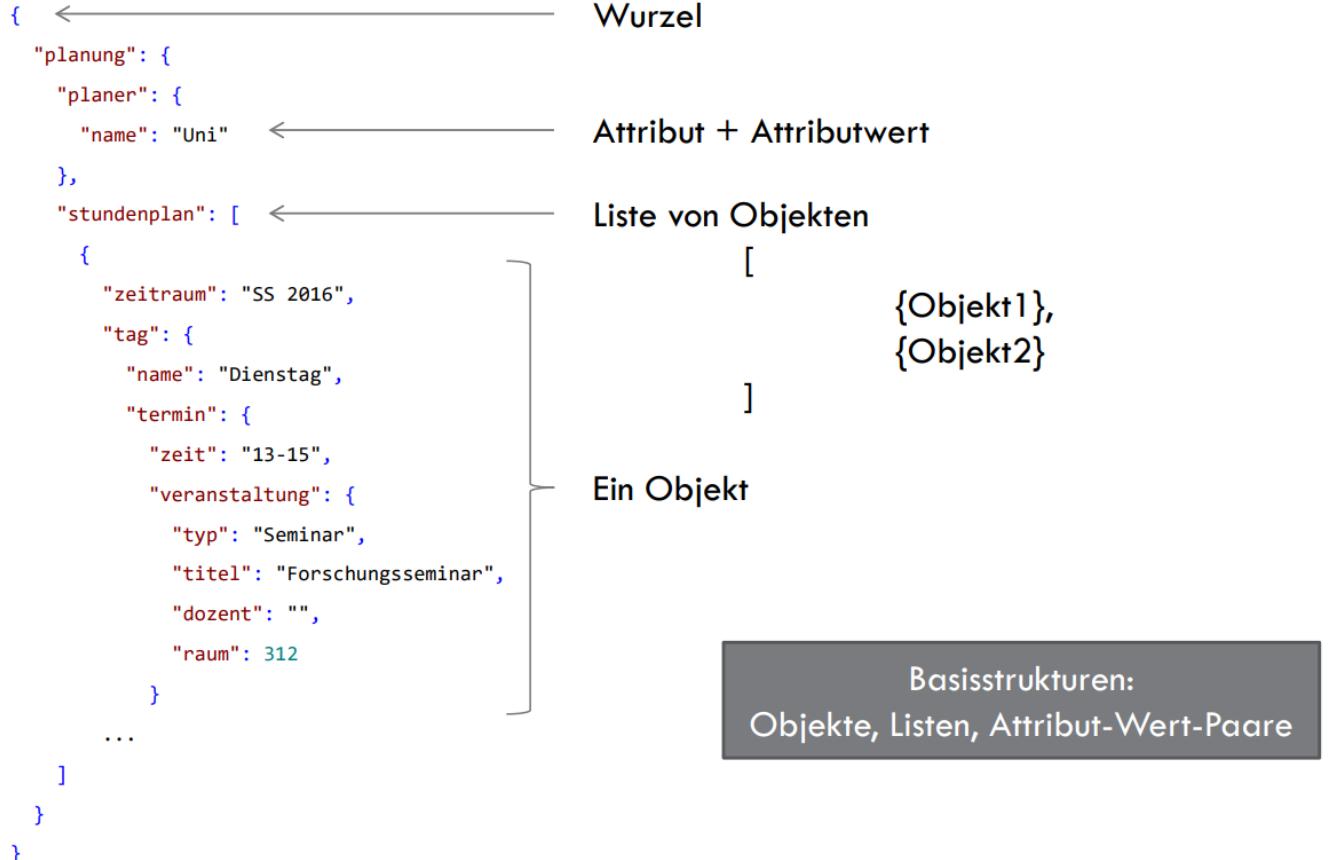


Abbildung 2.3: JSON-Konzepte

JavaScript Object Notation oder Kurz JSON [Bra14], [CM17], [di] ist ein beliebtes Textdatenformat, das zum Austauschen von Daten in modernen Webanwendungen und mobilen Anwendungen verwendet wird. JSON wird auch zum Speichern von unstrukturierten Daten in Protokolldateien oder NoSQL-Datenbanken wie Microsoft Azure Cosmos DB verwendet. Viele REST-Webdienste geben Ergebnisse zurück, die als JSON-Text formatiert sind, oder akzeptieren Daten, die im JSON-Format formatiert sind. Beispielsweise verfügt das WossiDiA-System über REST-API, die JSON-Daten zurückgibt. JSON ist auch das hauptsächliche Format zum Austauschen von Daten zwischen Webseiten und Webservern über AJAX-Aufrufe.

JSON erfüllt ähnliche Aufgaben wie XML. Das Format speichert Daten, die so in einem sowohl für Menschen als auch für Maschinen lesbarer Form strukturiert sind. Dazu verwendet es Namen wert-Paare und eine Formatierung mit geschweiften Klammern, der Struktur ist in der Abbildung 2.3 zu sehen.

Jedes JSON-Dokument besteht aus vier verschiedenen Konzepte [Wik22] (Wurzel, Attribute, Objekte, Arrays).

Wurzel:

- Das JSON-Dokument beginnt mit einer Wurzel, der nicht unbedingt ein Elementname enthalten muss, wie in der Abbildung 2.4 zu sehen
- Der Wurzel kann anhand einer geschweiften Klammer dargestellt werden.

```
{ -> Die Wurzel-Element
  "name": "Miller John",
  "mobile": "897654321",
  "age": 45,
  "address": {
    "city": "New York",
    "country": "USA"
  }
}
```

Abbildung 2.4: JSON-Wurzel, Quelle: [gee]

Attribute:

- Das JSON kann Attribute enthalten, die als Attribut-Wert-Paare dargestellt sind.
- Jedes Attribut hat einen Namen und dazugehörigen Wert, die durch einen Doppelpunkt getrennt sind (Z. B. "Wohnort": "Berlin").

Objekte:

- Das JSON-Dokument enthält Objekte (Elemente), die durch Doppelpunkt und geschweifte Klammern spezifiziert werden können, wie in der Abbildung 2.5 zu sehen.
- Objekte können Attribute oder eine Liste von Objekten enthalten

```
"train": {  
    "date": "07/04/2016",  
    "time": "09:30",  
    "from": "New York",  
    "to": "Chicago",  
    "seat": "57B"  
}
```

Abbildung 2.5: JSON-Objekt, Quelle: [cob]

Arrays:

- Das JSON-Dokument enthält Arrays, die weiterhin Attribute und Objekte enthalten können, wie in der Abbildung 2.6.
- Die Liste kann auch anhand eines Doppelpunktes und Ecke klammern spezifiziert werden.

```
{
  "glEntries": [
    {
      "generalLedgerId": 1,
      "accountId": 34,
      "amount": 32334.23,
      "description": "desc1",
      "debit": "Yes"
    },
    {
      "generalLedgerId": 2,
      "accountId": 35,
      "amount": 323.23,
      "description": "desc",
      "debit": "Yes"
    },
    ...
  ]
}
```

Abbildung 2.6: JSON-Liste, Quelle: [stab]

Infolgedessen basiert ein JSON-Dokument auf drei Basisstrukturen: Objekte, Listen und Attributwerte-Paare, wie in der Abbildung 2.3 zu sehen. Kurz gesagt besteht jedes JSON-Dokument aus zwei Strukturen:

- Menge von Name/Wert-Paare (Objekt) wie in der Abbildung 2.7.

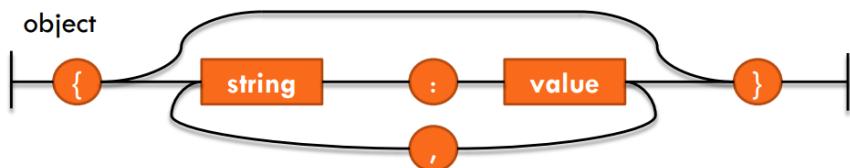


Abbildung 2.7: JSON-Objekt-Value, Quelle: [CM17]

- Geordnete Liste von Werten wie in der Abbildung 2.8 (Array).
- Die Werte können String, Number, Objekt, Array, True, False oder Null sein wie in der Abbildung 2.9.

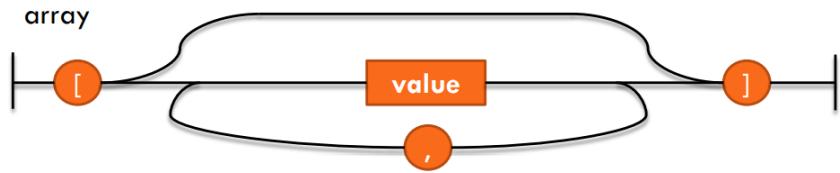


Abbildung 2.8: JSON-List-Value, Quelle: [CM17]

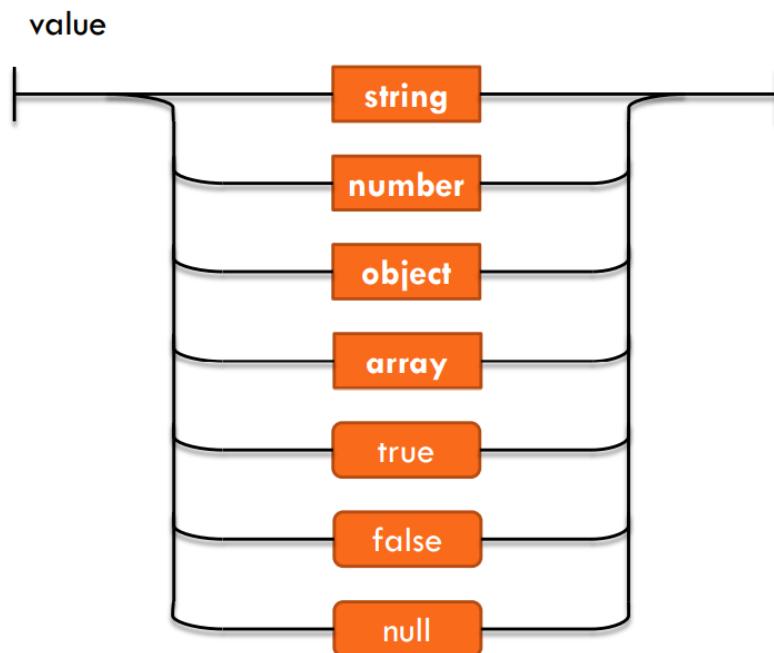


Abbildung 2.9: JSON-Values, Quelle: [CM17]

JSON hat aber variable Datenstrukturen ohne festes Schema [PRS⁺16], [mon], dadurch hat es eine flexiblere Gestaltung, damit ist JSON aber auch mehr durcheinander, zum Beispiel es ist unmöglich zu wissen, wo bestimmte Informationen liegen oder ob ein Wert von der Anwendung benötigt und muss angegeben werden. Eine Definition vom bekannten Version ist in Abbildung 2.10 zusehen.

Die Definition dieser Version hat die folgenden Merkmale:

- Eine Definition beginnt mit einem Verweis auf der Wurzelement.

- Eine Definition hat einen Namen.
 - Eine Definition hat einen Typ und erforderliche Eigenschaften.
 - Eine Definition kann optionale Eigenschaften haben.
 - Die Definition kann auf weitere Definitionen verweisen.
 - Der Typ kann entweder ein Objekt oder ein Array sein.
 - Die erforderlichen Eigenschaften können entweder in der Wurzelement-Definition oder in anderen separaten Definitionen beschrieben.
 - Die optionalen Eigenschaften können entweder in der Wurzelement-Definition oder in anderen separaten Definitionen beschrieben.

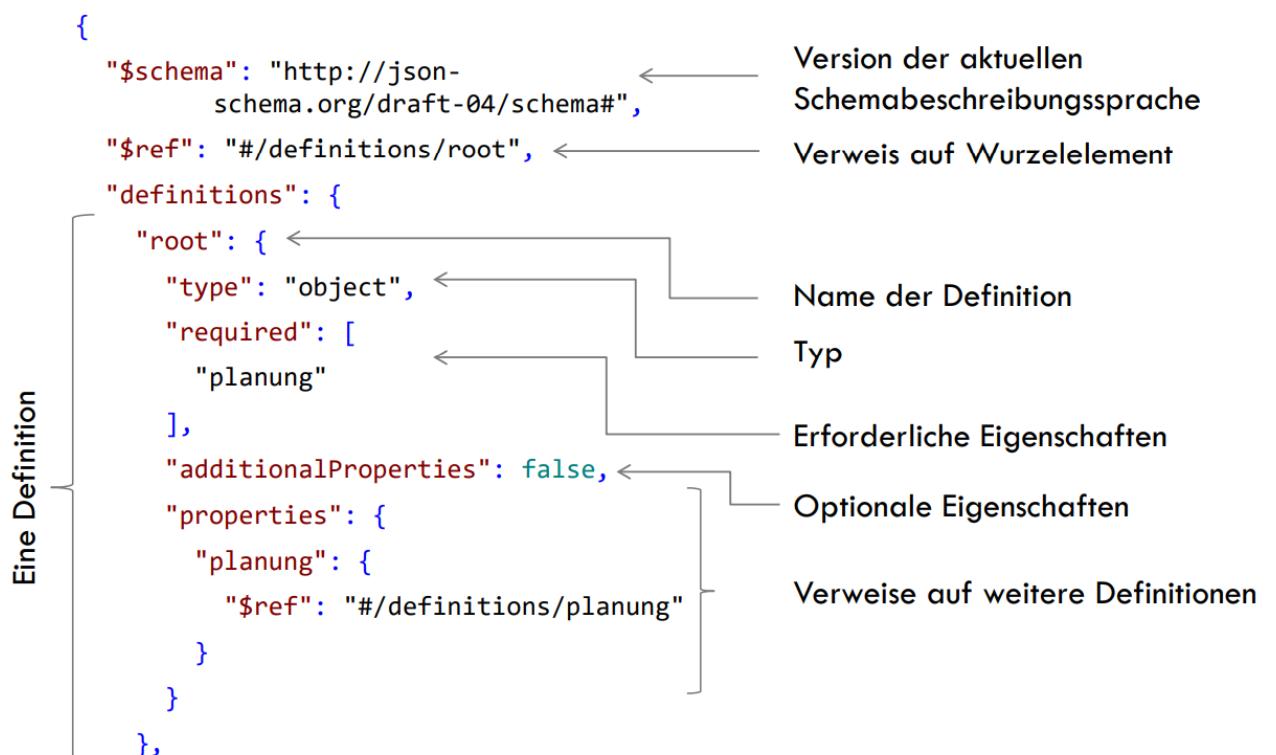


Abbildung 2.10: JSON-Schema

3 Stand der Technik

Das Graphen-Gebiet ist ein sehr umfangreiches Gebiet, mit dem sich Forscher*innen beschäftigen. Die Graphen selbst haben verschiedene Strukturen und Spezialisierungen. Das Eigenschaftsgraph-Modell ist wie bereits erwähnt ein spezieller Fall von Graphen, der Beschriftungen und Eigenschaften für Knoten und Kanten unterstützt.

Auf dem Markt gibt es viele Tools zur Umwandlung von JSON-Daten allgemein in Graph-Daten wie mxgraph, doch keine davon die Umwandlung in Eigenschaftsgraphen unterstützen.

Die Umwandlung in Eigenschaftsgraphen kann auch von anderen Daten-Formaten erfolgen. Zum Beispiel können XML-Daten in Eigenschaftsgraphen umgewandelt werden. Es existieren bereits Tools, basierend auf XPath für die Umwandlung von XML-Dateien in Eigenschaftsgraphen-Daten.

Um JSON-Daten in Eigenschaftsgraph-Daten umzuwandeln, gibt es zwei verschiedene Wege:

- Ein Tool entwickeln basierend auf JSONPath, um die JSON-Daten zu filtern.
- JSON-Daten in andere Daten-Format umwandeln, für die bereits ein Tool zur Umwandlung in Eigenschaftsgraph existiert.

Der JsonPath-basierter Ansatz kann die Anforderungen für unser gegebenes Problem nicht erfüllen, da das JSONPath eingeschränkt ist.

Die Anforderungen können aber durch den zweiten Weg erfüllt werden, da es möglich ist, basierend auf der existierende X2G Tool, die JSON-Daten in Eigenschaftsdaten umzuwandeln, indem die JSON-Daten in XML-Daten allgemein transformiert werden, die dann in Eigenschaftsdaten umgewandelt werden. Dieser Weg hat einen besonderen Vorteil, der darin besteht, JSON und XML Daten gleichzeitig in einem Tool zu Eigenschaftsdaten zu transformieren. Benutzer können durch diesen Weg beliebige JSON und XML Dateien in Eigenschaftsgraph umwandeln.

Das X2G Software hat zwei Versionen, wobei die erste Version spezifisch für WossiDiA-System und die optimierte Version allgemein für die Umwandlung von XML-Dateien in Eigenschaftsgraphen entwickelt wurden.

Für die Transformation von JSON-Dateien zu XML-Dateien gibt es verschiedene Ansätze, die angewendet und optimiert werden können. Die Ansätze konvertieren die JSON-Daten zu XML-Daten in unterschiedlichen Wegen, sodass die Ergebnisse anders sind. Der am besten geeigneten Ergebnis davon ist der dem XML-ToString-Ansatz.

zunächst wird in Abschnitt 3.1 der JsonPath-basierter Ansatz erörtert und erklärt, warum dieser Ansatz unser gegebenes Problem nicht löst. In Abschnitt 3.2 werden die zwei Versionen von X2G vorgestellt und erklärt, wie die optimierte Version die Anforderungen unser gegebenes Problem erfüllt, die von der ersten Version nicht erfüllen konnte. Schließlich werden in Abschnitt 3.3 die Möglichkeiten für die JSON-XML Transformation erörtert und gezeigt wie der XML-ToString-Ansatz am besten geeigneten Ergebnis für dieser Arbeit ist.

3.1 JsonPath-basierter Ansatz

JSONPath	Beschreibung
\$	Das Wurzelement
@	Das aktuelle Element
. or []	Der nächste Kindknoten
n/a	Der Vaterknoten
..	Nachfolgeknoten
*	Wildcard für Elemente
n/a	Zugriff auf Attribute. Diese existieren in JSON nicht
[]	Iteration über Kollektionen.
[,]	Vereinigung von Elementmengen
[start:end:step]	Auswahl einer Teilmenge
?()	Anfrageprädikate
n/a	Gruppierung in XPath

Abbildung 3.1: JSONPath-Funktionen, Quelle: [goe]

Das Filtern von Dokumenten und Daten sowie die selektive Extraktion von Daten aus JSON-Dateien ist eine wesentliche Anforderung dieser Arbeit. Ein häufig hervorgehobener Vorteil von XML ist die Verfügbarkeit zahlreicher Tools zur Analyse, Transformation und selektiven Extraktion von Daten aus XML-Dokumenten. XPath ist eines dieser leistungsstarken Tools. JSON hat hingegen zu XML keine zahlreichen Tools zur Analyse und Transformation, es hat aber sowie XML ein Tool (mit eingeschränkten Möglichkeiten im Vergleich zu XPath) zur selektiven Extraktion von Daten aus JSON-Dokumenten und es heißt JSONPath [tab], [goe]. JSONPath-Ausdrücke verweisen immer auf eine JSON-Struktur [bae], so wie XPath-Ausdrücke in Kombination mit einem XML-Dokument verwendet werden. Da eine JSON-Struktur normalerweise anonym ist und nicht unbedingt ein "Root-Member-Objekt" hat, nimmt JSONPath den abstrakten Namen an, der dem Objekt der äußeren Ebene zugewiesen ist. JSONPath erlaubt das Platzhaltesymbol

bol (*) für Mitgliedsnamen und Array-Indizes. XPath hat jedoch viel mehr zu bieten (Ortspfade in nicht abgekürzter Syntax, Operatoren und Funktionen) als hier aufgeführt. Darüber hinaus gibt es einen bemerkenswerten Unterschied, wie der tiefgestellte Operator in XPath und JSONPath funktioniert. Eine Syntax-Elemente-Beschreibung von JSONPath ist in Abbildung 3.1 zu sehen. Die Abfragesprache JSONPath hat einige Probleme, derentwegen die selektive Extraktion nicht komplett erfolgen kann, Beispiele zu diesen Problemen sind:

- Derzeit sind in JSONPath-Ausdrücken nur einfache Anführungszeichen zulässig. Skript ausdrücke innerhalb von JSONPath-Speicherorten werden derzeit nicht rekursiv von ausgewertet. Nur die globalen und lokalen Symbole werden durch einen einfachen regulären Ausdruck erweitert.
- Eine Alternative für die Rückgabe im Falle einer fehlenden Übereinstimmung kann darin bestehen, in Zukunft ein leeres Array zurückzugeben. JSONPath-False

Das wichtigste Problem ist es, dass JSONPath-Funktionen keinen Zugriff auf Attributwerte erlauben, was ist die Speicherung und Extraktion von Werten unmöglich macht. In XPath ist es mit der Funktion (`text()`) möglich.

Diese Probleme und Einschränkungen von JSONPath verhindern die selektive Extraktion von JSON-Dateien, was die Anforderungen dieser Arbeit verstößt, damit ist dieser Ansatz für diese Arbeit nicht geeignet ist.

3.2 X2G-basierter Ansatz

X2G wurde entwickelt, um XML-Daten zu verarbeiten und sie allgemein in Graph-Daten umzuwandeln, die dann mit speziellen Programmen visualisiert werden können. Die Software kann mit XML-Schema umgehen und eine große Menge von XML-Dokumenten bearbeiten. [Mey22] [Zak22]

Die Software basiert auf XPath und einer definierten Regelsprache. Der XPath steht für die Extraktion von Daten aus XML-Dokumenten zu extrahieren. Mit der Regelsprache sind Benutzer in der Lage Regeln zu definieren, die Regeln können zum Beispiel das Wählen von Knoten und Kanten sein, eine genaue Erklärung von dieser Regelsprache ist in Abschnitt 4.4.3 im Kapitel Konzept.

Der grobe Ablauf der Software ist in der Abbildung 3.2 dargestellt. Der Ablauf der Software besteht aus vier Phasen.

- Zunächst können Benutzer mittels XPath die gewünschte XML-Dokumente auswählen, dieser Eingabe wird zum System übermittelt, um die von Benutzer ausgewählte Dokumente zu filtern.
- Die gefilterten Dateien werden in Dateiarrays gespeichert.
- Benutzer können bestimmte Elemente und\oder Attribute aus den gefilterten Dokumenten auswählen.
- Das System fängt an, die ausgewählte Daten zu extrahieren.
- Benutzer bestimmen Konten, Kanten mittels der Regelsprache.
- Das System generiert die gewählte Knoten, Kanten und dazugehörigen Eigenschaften und Beschriftungen aus den extrahierten Daten.

- Der letzte Schritt ist der Exporter, der Exporter speichert die generierte Graph-Struktur (Knoten, Kanten, Eigenschaften, Beschriftungen) als CSV-Dateien.
- Die gespeicherten CSV-Dateien für einen gewünschten Eigenschaftsgraph bestehen aus zwei Dateien (Nods.csv und Edges.csv). Diese Dateien können auch mit Graph-Programme wie Gephi visualisiert werden.

Diese Technik hilft den Forscher*innen beim Analysieren, Durchsuchen, und der Visualisierung bestimmter Aspekte in den gesammelten XML-Daten mithilfe des Graphenparadigmas und der entwickelten Algorithmen zu unterstützen.

Die erste Version der Software ist speziell für das WossiDiA-System entwickelt, d.h., dass die internen Daten der XML-Dateien vom Entwickler bekannt sind, damit wurden eine bestimmte beschränkte Anzahl von Knoten und Kanten definiert, wobei die Software nur diese Knoten bzw. Kanten erzeugen kann [Zak22].

Zum Beispiel beschreiben die Test-Dateien dieser Software die Geschichten von vorherigen Menschen, die von Richard WoosiDiA aufgeschrieben wurden. Mit diesen Kenntnissen hat der Entwickler die Knoten (Person, Story, Keywords, Content) vordefiniert, daher können andere Knoten nicht generiert werden, infogedessen werden keine Knoten und\oder Kanten aus JSON-Dateien erzeugt, wenn ihre internen Daten nicht ähnlich zu den XML-Daten des WossiDiA-Systems sind.

Diese Spezialisierung verstößt die allgemeine Umwandlung von JSON-Daten in Eigenschaftsdaten, was die Anforderungen dieser Arbeit nicht erfüllt.

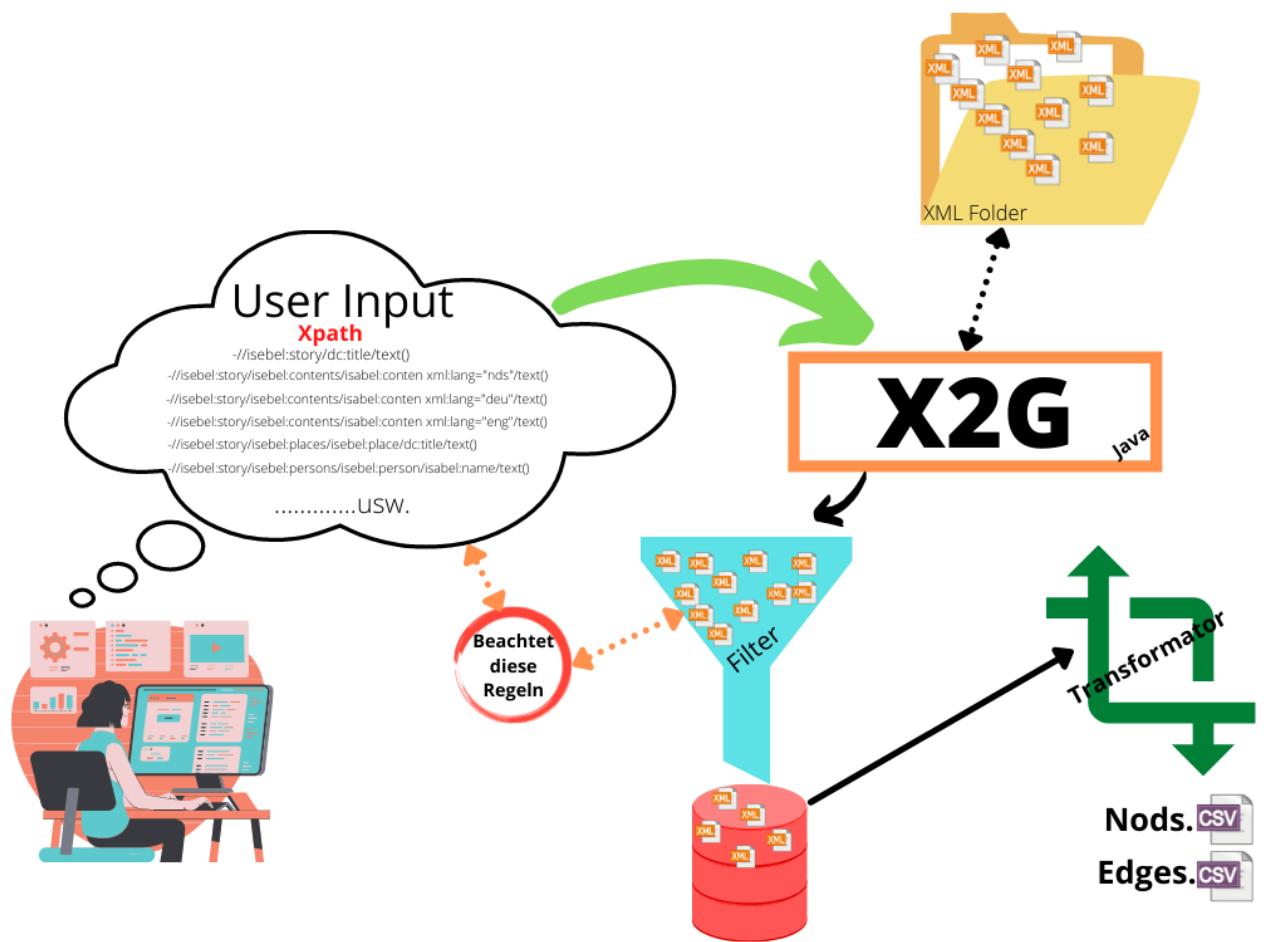


Abbildung 3.2: X2G Ablaufdiagramm erste Version

Die zweite optimierte Version von X2G hat dieses Problem bewältigt, d.h., die Spezialisierung der Umwandlung ist weggefallen, dadurch erwartet die Software nun keine bestimmte Daten mehr, um daraus einen Eigenschaftsgraph erzeugen zu können.

Die optimierte Software von Dr. Meyer kann nun beliebige Knoten und Kanten aus XML-Dateien allgemein oder durch Benutzer definierte Regeln generieren, was die Anforderungen dieser Arbeit erfüllt, damit ist der optimierter X2G-Ansatz für dieser Arbeit geeignet.

Mit dieser Ansatz müssen JSON-Dokumente keine bestimmte interne Daten mehr enthalten und können allgemein oder durch Benutzer definierte Regeln in Eigenschaftsgraph-Daten umgewandelt werden, indem die JSON-Dokumente zu XML-Dokumenten konvertiert werden und danach in Eigenschaftsgraph umgewandelt werden. Die ausführliche Beschreibung dieses Prozesses befindet sich in dem Konzept Kapitel 4.

3.3 JSON-XML-Ansätze

3.3.1 XSLT-basierter-Ansatz

```
<data>{
    "content": [
        {
            "id": 70805774,
            "value": "1001",
            "position": [1004.0,288.0,1050.0,324.0]
        }
    ]
}</data>
```

Abbildung 3.3: in einem Tag gepacktes JSON-Beispiel, Quelle: [ral]

```
<?xml version="1.0"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:math="http://www.w3.org/2005/xpath-functions/math"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="xs math" version="3.0">
    <xsl:output indent="yes" omit-xml-declaration="yes" />

    <xsl:template match="data">
        <xsl:copy-of select="json-to-xml(.)"/>
    </xsl:template>
</xsl:stylesheet>
```

Abbildung 3.4: XSLT-Vorlage für eine JSON-Datei, Quelle: [ral]

```
<map xmlns="http://www.w3.org/2005/xpath-functions">
  <array key="content">
    <map>
      <number key="id">70805774</number>
      <string key="value">1001</string>
      <array key="position">
        <number>1004.0</number>
        <number>288.0</number>
        <number>1050.0</number>
        <number>324.0</number>
      </array>
    </map>
  </array>
</map>
```

Abbildung 3.5: XSLT-Ergebnis, Quelle: [ral]

XSLT ist die zweite Möglichkeit, um XML-Daten für den Browser relativ gut darstellbar zu machen. XSL(T) beschreibt dazu die Funktionen und die Syntax, der es ermöglicht, XML-Elemente in andere Elemente wie z.B. HTML umzuwandeln [acp]. XSL ist von XML abgeleitet, was letztlich der gleichen Syntax hervorruft. XSL(T)-Dokumente können also problemlos mit einem XML-Editor erstellt und bearbeitet werden. Grundsätzlich haben XSL-Dokumente die Dateiendung (.xsl), eine andere Endung sollte allerdings auch keine Probleme bereiten.

Ebenso wie ein XML-Dokument, muss auch ein XSL-Dokument mit der XML-Deklaration beginnen, ein Root-Element (XSL: Stylesheet) enthalten und wohlgeformt sein. Fertige XSL-Dokumente werden ebenso wie CSS-Dateien in XML eingebunden - man verwendet lediglich einen anderen Dateityp.

Die ersten Versionen von XSLT konnten nur XML-Dokumente verarbeiten. Das XSLT 3.0 kann mit JSON-Dateien umgehen, um JSON entweder in XML oder ein neues

Format umzuwandeln [ral]. Bei der Arbeit mit XSLT muss die JSON-Struktur in ein XML-Tag gepackt werden `jData`, wie auf der Abbildung 3.3. Jetzt können die XSLT-Funktion JSON-To-XML verwendet werden, um das JSON in eine XML-Struktur zu konvertieren, dafür muss eine Vorlage in XSTL erstellt werden, um zu bestimmen, wie die JSON-Elemente konvertiert werden sollen wie in der Abbildung 3.4. Als nächstes erstellt der XSLT-Prozessor die Tags "Map und Ärray", um die Daten in einem XML-Schema zu strukturieren. Es konvertiert auch Zahlen, Zeichenfolgen und boolesche Werte in ein entsprechendes XML-Tag wie in der Abbildung 3.5 zu sehen. Nachdem die JSON-Datenstruktur mit der Funktion JSON-To-XML transformiert wurde, können diese Daten dann mit der Template-Technik transformiert werden.

Die Transformation in XSTL erfolgt nicht generisch, dafür muss für jede einzelne JSON-Datei eine Vorlage erstellt werden, was es komplizierter macht. Die erzeugte XML-Datei als Ergebnis ist nicht die erwartete Datei, da der XSLT-Prozessor die Tags "Map und Ärray" erstellt. dieser XML-Struktur macht es schwerer für den Benutzer, bestimmte Elemente aus JSON-Dateien mittels XPath auszuwählen, außerdem muss der XSLT-Prozessor installiert und mit Java verbunden werden. Aus diesen Gründen ist diese Methode für diese Arbeit nicht geeignet.

3.3.2 XML.Serializer-basierter-Ansatz

XML.Serializer Klasse ist die zweite Möglichkeit JSON in XML umzuwandeln. XML.Serializer ist eine vordefinierte Klasse in Java. Am Anfang muss die JSON-Datei zu String umgewandelt und als Parameter in der Methode JSONSerializer.toJSON (String JSON) übergeben werden. Als Nächstes wird ein XMLSerializer Instanz erstellt. Der letzte Schritt ist JSON-String zu XML-String umzuwandeln, mit der Methode xmlSerializer.write(JSON), es wandelt ein Objekt vom Typ JSON in XML-String um.

Diese Methode hat aber einige Probleme, das erste Problem ist das Wurzelement, es erzeugt die Element (o) als Wurzelement, was zu Probleme mit XPath führt. Das zweite Problem ist das Umgehen mit Listen in JSON, diese Methode wandelt Listen als Objekten, indem es die Namen der Listen als Objekt betrachtet und deren Elemente als (e) Elemente erzeugt, ein Beispiel ist in den Abbildungen 3.6 3.7 zu sehen.

Diese beide Probleme führen zu Schwierigkeiten bei der Anwendung von XPath. Beispielsweise bei der Anwendung von XPath mit der Ausdruck (\store\Book[2]) in der erzeugten XML-Datei, um das zweite Element der Liste Book zu extrahieren, wird ein (No match) zurückgeliefert, da Der XPath nur vollständige XML-Namespaces unterstützt. Das zweite Element der Liste Book kann aber mit anderen Ausdruck extrahiert werden, und zwar mit (\store\Book[2]\e[2]). Das macht es komplexer, für den Benutzer beim Wählen ihren gewünschten Daten. Aus diesem Grund ist dieser Ansatz für diese Arbeit nicht geeignet.

```
{
  "store": {
    "book": [
      {
        "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "price": 8.95
      },
      {
        "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      },
      {
        "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      {
        "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
  }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<o>
  <store class="object">
    <bicycle class="object">
      <color type="string">red</color>
      <price type="number">19.95</price>
    </bicycle>
    <book class="array">
      <e class="object">
        <author type="string">Nigel Rees</author>
        <category type="string">reference</category>
        <price type="number">8.95</price>
        <title type="string">Sayings of the Century</title>
      </e>
      <e class="object">
        <author type="string">Evelyn Waugh</author>
        <category type="string">fiction</category>
        <price type="number">12.99</price>
        <title type="string">Sword of Honour</title>
      </e>
      <e class="object">
        <author type="string">Herman Melville</author>
        <category type="string">fiction</category>
        <isbn type="string">0-553-21311-3</isbn>
        <price type="number">8.99</price>
        <title type="string">Moby Dick</title>
      </e>
      <e class="object">
        <author type="string">J. R. R. Tolkien</author>
        <category type="string">fiction</category>
        <isbn type="string">0-395-19395-8</isbn>
        <price type="number">22.99</price>
        <title type="string">The Lord of the Rings</title>
      </e>
    </book>
  </store>
</o>
```

Abbildung 3.6: JSON-Beispiel, Quelle: [staa]

Abbildung 3.7: Das erzeugte XML-Datei,

Quelle: [staa]

3.3.3 XML.toString-basierter-Ansatz

XML.toString Methode steht auch für die Umwandlung von JSON-Daten in XML-Daten. Diese Methode weist auch Probleme wie die vorherigen erklärten Methoden, wie zum Beispiel:

- Das erzeugte XML-String enthält kein Root-Element.
- Bei der Umwandlung achtet sie nicht auf der Reihenfolge der JSON-Datei.

Im Vergleich zu anderen Methoden ist diese Methode am besten geeignete Methode für diese Arbeit da:

- Sie JSON-Elemente ohne Manipulation transformiert.
- Sie einfache lösbarer Probleme im Vergleich zu anderen Methoden (XML.Serialize, XSLT) hat.

im nächsten Kapitel wird diese Methode mehr veranschaulicht und wie ihre Probleme bewältigt wurden.

4 Konzept

Diese integrierte Lösung wurde entwickelt, um JSON-Daten allgemein und durch Benutzer definierte Regeln in Property Graph Daten umzuwandeln. Mit dieser Software sind Forscher*innen in der Lage, Analyse auf der Property Graph daten zu fahren. In diesem Kapital wird die integrierte Lösung J2G erklärt, jede Phase wird beschrieben und wie der Übergang zu der nächsten Phase erfolgt wird gezeigt. Dieser entwickelter Ansatz hat noch neben dem wesentlichen Ziel zwei andere Vorteile, die durch diesen Ansatz gewonnen werden können, und zwar:

- Diese entwickelte kombinierte Lösung steht für die Umwandlung von JSON-Daten in Property Graph-Daten.
- Da dieser Ansatz aus zwei Phasen besteht, kann dann dadurch JSON und XML-Daten in Property Graph-Daten umgewandelt werden.
- Die erste Phase (J2XML) kann getrennt angewendet werden, um JSON-Daten zu XML-Daten zu transformieren.

Dieses Kapitel ist wie folgt aufgebaut: im ersten Abschnitt ?? werden die Anforderungen dieser Arbeit definiert. Im Abschnitt zwei 4.2 wird das Ablaufdiagramm erörtert. Im dritten Abschnitt 4.3 wird die erste Phase erklärt, wie sie aufgebaut ist sowie ihre Aufgabe. Im letzten Abschnitt 4.4 wird die Phase X2G genau beschrieben und wie sie aufgebaut ist.

4.1 Anforderungen

J2G hat wie jede andere Software ein paar Anforderungen. Mit Anforderungen ist was soll\kann die Software machen gemeint. die J2G Anforderungen sind:

- Die Software soll\kann JSON-Daten allgemein in Eigenschaftsgraph umwandeln.
- Die Software kann bestimmte JSON-Dateien durch Benutzer definierte Regeln auswählen bzw. filtern.
- Die Software kann bestimmte JSON-Elemente oder Attribute durch Benutzer definierte Regeln auswählen bzw. filtern.
- Die Software kann aus den resultierenden Dateien bzw. Elementen gewünschte Knoten und\oder Kanten erzeugen.
- Die Software kann von Benutzer definierte Servers JSON-Dateien abrufen.
- Die Software kann Kanten zwischen von Benutzer ausgewählte Elementen (Knoten) aus verschiedenen Dateien erzeugen.
- Die Software soll das erzeugte Eigenschaftsgraph als GEXF-Dateien (Nodes.gexf, Edges.gexf) speichern.

4.2 Ablaufdiagramm

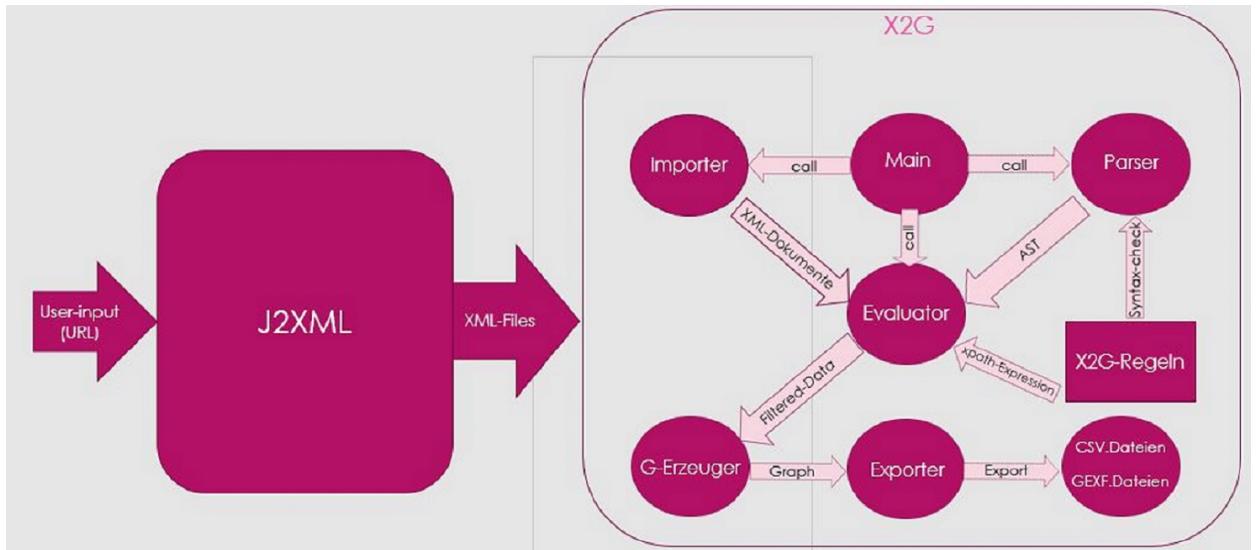


Abbildung 4.1: Ablaufdiagramm des integrierten Ansatzes (J2XML+X2G)

Die Abbildung 4.1 zeigt den Prozess der Software. Diese Software besteht aus zwei Phasen J2XML und X2G. Der Prozess läuft wie folgt:

- Zunächst gibt den Benutzer die URL ein, wo die JSON-Daten eigentlich liegen.
- Wenn die JSON-Daten auf dem PC der Benutzer liegen, hat er dann auch die Möglichkeit, diese Daten problemlos auf der Software zu hochladen.
- Benutzer können mehrere JSON-Dateien gleichzeitig in Property Graph-Daten umwandeln.
- Im nächsten Schritt werden in J2XML-Phase die vom Benutzer gewählte JSON-Daten in XML-Daten umgewandelt.
- Die konvertierte Daten werden zu der X2G-Software (die zweite Phase) übergeben.
- In der zweiten Phase geben Benutzer ihren XPath-Ausdrücken oder JSONPath-Ausdrücke ein, um bestimmte gewünschte Daten zu extrahieren.

- Benutzer definieren Ihre Regeln mittels der Regelspezifikationssprache in X2G-Software, um die gewünschte Knoten bzw. Kanten zu generieren.
- Als Resultat bekommen Benutzer die Property Graph-Daten in CSV und\oder in GEXF Datenformat gespeichert.
- Das erzeugte Property Graph spaltet sich in zwei Dateien (Knoten und Kanten).

4.3 J2XML

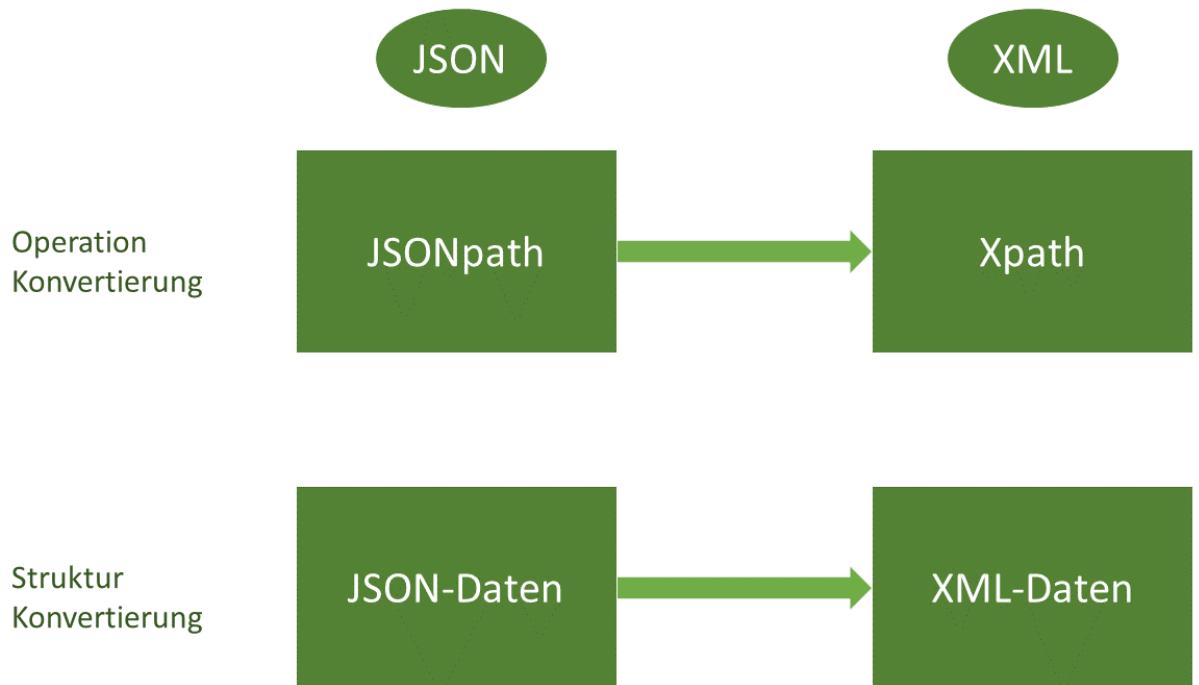


Abbildung 4.2: J2XML

In dieser Phase werden JSON-Daten zu XML-Daten konvertiert. Außerdem werden JSONPath-Ausdrücke zu XPath-Ausdrücke konvertiert. Diese Phase erfüllt zwei Aufgaben, wie auf der Abbildung 4.2 zu sehen:

- JSON-Daten zu XML-Daten transformieren.
- JSONPath-Ausdrücke zu XPath-Ausdrücke konvertieren.

Die Transformation zu XML besteht aus sechs Schritten wie auf der Abbildung 4.3 zu sehen:

- 1-JSON-Daten holen oder empfangen.
- 2-JSON Wohlgeformtheit prüfen.

- 3-JSON zu XML konvertieren.
- 4-XML Wohlgeformtheit prüfen.
- 5-Zu XML-Wohlgeformtheit anpassen.
- 6-XML-Daten als XML-Dokument zu der nächsten Phase (X2G) übergeben.

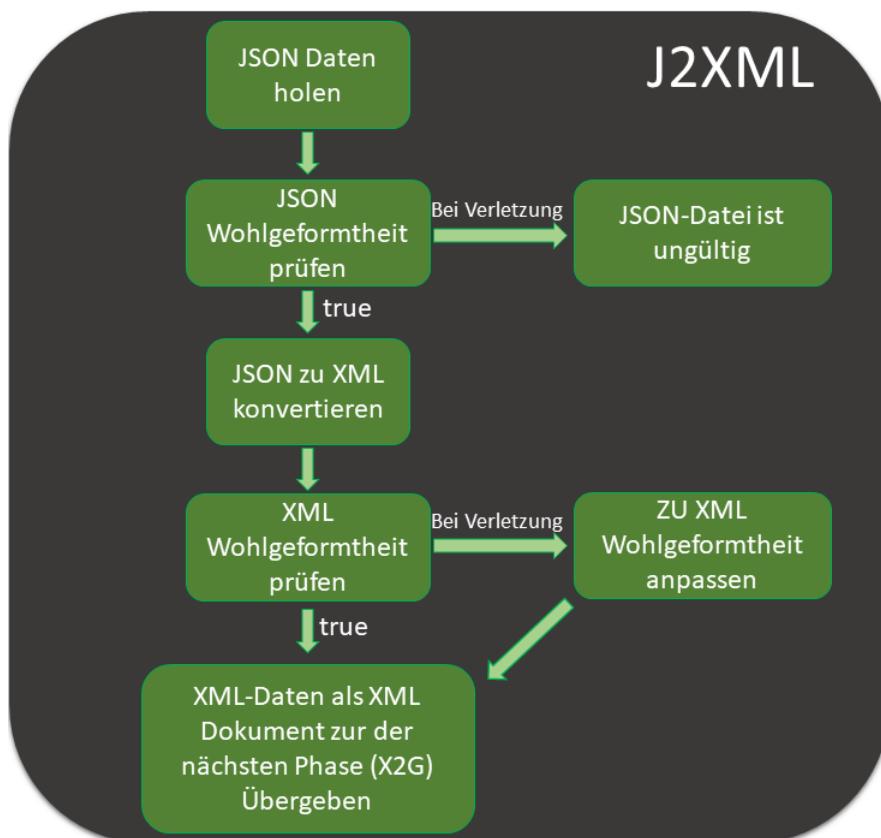


Abbildung 4.3: J2XML, strukturelle Konvertierung

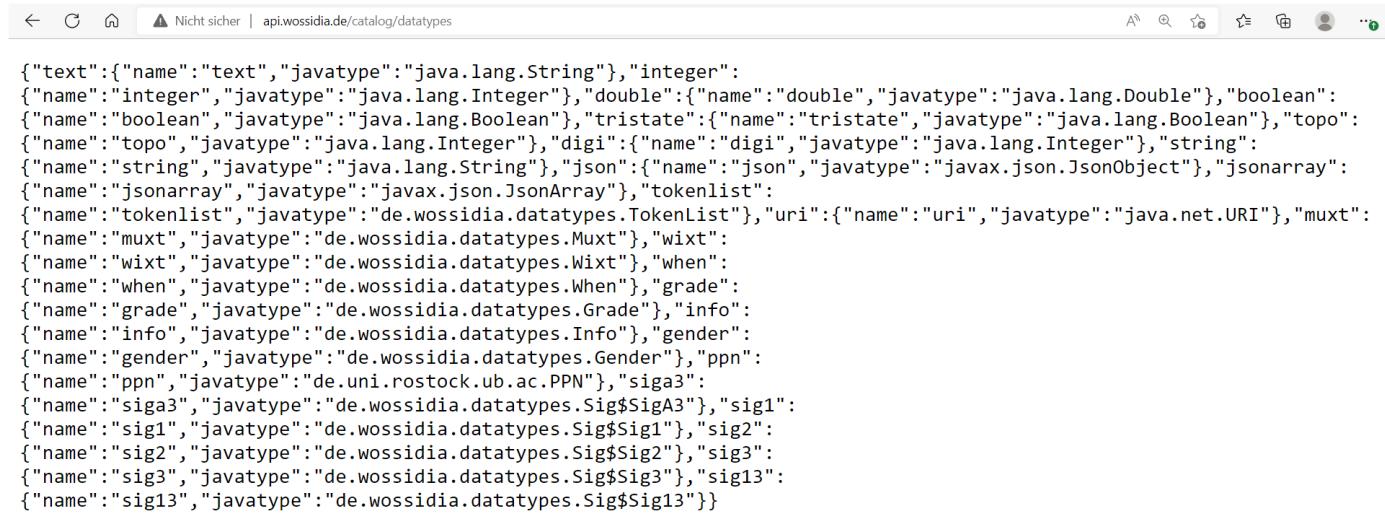
In den nächsten Abschnitten wird jeder Schritt der Transformation beschrieben.

4.3.1 JSON-Daten holen

Zunächst geben Benutzer ihren URLs ein, wo JSON-Daten liegen. Diese URL wird von der Software übernommen. Nun ruft die Software den Server ab und holt von ihm die Daten ab. Im Fall zwei, dass JSON-Daten nicht in einem Server gespeichert sind, können Benutzer ihren JSON-Daten in der Software hochladen. Der Server Abruf läuft wie folgt:

- Am Anfang baut die Software eine Verbindung mit dem Server auf.
- Die Verbindung beginnt mit einem Request von der Software an den Server.
- Die Verbindung erfolgt, nachdem der Server eine Response („ok“) zu der Software zurückschickt.
- Die Software überschreibt nun die JSON-Daten und speichert sie in einem String.
- Bei erfolgreicher Überschreibung beendet die Software die Verbindung mit dem Server.
- Bei erfolgloser Verbindung oder Überschreibung zeigt die Software einen Fehler.

Die Abbildung 4.4 zeigt ein Beispiel von JSON-Daten, die in einem von Benutzer eingegebenen URL liegen. Diese URL ist von dem WossiDiA-System und enthält JSON-Daten, die Datentyps beschreiben. Die Abbildung 4.5 zeigt der überschriebenen JSON-Daten in einem String. Die Sortierung der JSON-Elemente im String ist anders als die Sortierung in der ursprünglichen JSON-Datei. Der Unterschied ist aber kein Fehler, da die Reihenfolge in JSON keine Rolle spielt.



```

{
  "text": {"name": "text", "javatype": "java.lang.String"}, "integer": {"name": "integer", "javatype": "java.lang.Integer"}, "double": {"name": "double", "javatype": "java.lang.Double"}, "boolean": {"name": "boolean", "javatype": "java.lang.Boolean"}, "tristate": {"name": "tristate", "javatype": "java.lang.Boolean"}, "topo": {"name": "topo", "javatype": "java.lang.Integer"}, "digi": {"name": "digi", "javatype": "java.lang.Integer"}, "string": {"name": "string", "javatype": "java.lang.String"}, "json": {"name": "json", "javatype": "javax.json.JsonObject"}, "jsonarray": {"name": "jsonarray", "javatype": "javax.json.JSONArray"}, "tokenlist": {"name": "tokenlist", "javatype": "de.wossidia.datatypes.TokenList"}, "uri": {"name": "uri", "javatype": "java.net.URI"}, "muxt": {"name": "muxt", "javatype": "de.wossidia.datatypes.Muxt"}, "wixt": {"name": "wixt", "javatype": "de.wossidia.datatypes.Wixt"}, "when": {"name": "when", "javatype": "de.wossidia.datatypes.When"}, "grade": {"name": "grade", "javatype": "de.wossidia.datatypes.Grade"}, "info": {"name": "info", "javatype": "de.wossidia.datatypes.Info"}, "gender": {"name": "gender", "javatype": "de.wossidia.datatypes.Gender"}, "ppn": {"name": "ppn", "javatype": "de.uni.rostock.ub.ac.PPN"}, "siga3": {"name": "siga3", "javatype": "de.wossidia.datatypes.Sig$SigA3"}, "sig1": {"name": "sig1", "javatype": "de.wossidia.datatypes.Sig$Sig1"}, "sig2": {"name": "sig2", "javatype": "de.wossidia.datatypes.Sig$Sig2"}, "sig3": {"name": "sig3", "javatype": "de.wossidia.datatypes.Sig$Sig3"}, "sig13": {"name": "sig13", "javatype": "de.wossidia.datatypes.Sig$Sig13"} }

```

Abbildung 4.4: URL: WossidiA-Datatypes


```

{
  "text": {"name": "text", "javatype": "java.lang.String"}, "integer": {"name": "integer", "javatype": "java.lang.Integer"}, "double": {"name": "double", "javatype": "java.lang.Double"}, "boolean": {"name": "boolean", "javatype": "java.lang.Boolean"}, "tristate": {"name": "tristate", "javatype": "java.lang.Boolean"}, "topo": {"name": "topo", "javatype": "java.lang.Integer"}, "digi": {"name": "digi", "javatype": "java.lang.Integer"}, "string": {"name": "string", "javatype": "java.lang.String"}, "json": {"name": "json", "javatype": "javax.json.JsonObject"}, "jsonarray": {"name": "jsonarray", "javatype": "javax.json.JSONArray"}, "tokenlist": {"name": "tokenlist", "javatype": "de.wossidia.datatypes.TokenList"}, "uri": {"name": "uri", "javatype": "java.net.URI"}, "muxt": {"name": "muxt", "javatype": "de.wossidia.datatypes.Muxt"}, "wixt": {"name": "wixt", "javatype": "de.wossidia.datatypes.Wixt"}, "when": {"name": "when", "javatype": "de.wossidia.datatypes.When"}, "grade": {"name": "grade", "javatype": "de.wossidia.datatypes.Grade"}, "info": {"name": "info", "javatype": "de.wossidia.datatypes.Info"}, "gender": {"name": "gender", "javatype": "de.wossidia.datatypes.Gender"}, "ppn": {"name": "ppn", "javatype": "de.uni.rostock.ub.ac.PPN"}, "siga3": {"name": "siga3", "javatype": "de.wossidia.datatypes.Sig$SigA3"}, "sig1": {"name": "sig1", "javatype": "de.wossidia.datatypes.Sig$Sig1"}, "sig2": {"name": "sig2", "javatype": "de.wossidia.datatypes.Sig$Sig2"}, "sig3": {"name": "sig3", "javatype": "de.wossidia.datatypes.Sig$Sig3"}, "sig13": {"name": "sig13", "javatype": "de.wossidia.datatypes.Sig$Sig13"} }

```

Abbildung 4.5: Das von der Software gespeicherte String

4.3.2 JSON Wohlgeformtheit prüfen

Nachdem die Software die JSON-Daten geholt hat, beginnt sie mit dem nächsten Schritt. sie testet in diesem Schritt die Wohlgeformtheit der geholten JSON-Daten. eine JSON-Datei ist wohlgeformt, wenn Sie eine JSON Objekt-Form oder eine JSON Liste-Form hat 2.2. Dieser Schritt hilft dabei, wenn die von Benutzer gegebenen URL keine JSON-Daten enthält. Der Testprozess läuft wie folgt:

- Die Software testet, ob die JSON-Datei eine JSON Objekt-Form hat.
- Die Software testet, ob die JSON-Datei eine JSON Liste-Form hat, wenn die JSON-Datei kein JSON Objekt-Form hat.
- Bei erfolgreichem Test speichert die Software die getesteten Daten in einem File.
- Die Software schickt einen Fehler zurück, falls die JSON-Datei weder JSON Objekt noch JSON Liste-Form hat.

Bei der Testing, ob die JSON-Daten ein JSON Objekt oder eine JSON Liste-Form hat, überprüft die Software zugleich, ob die stehenden Values gültig sind. Ein Value ist gültig, wenn es String, number, Objekt, Array, true, false oder null ist.

In der Abbildung 4.6 steht eine wohlgeformte JSON-Datei. In der Abbildung 4.7 steht ein nicht wohlgeformtes JSON-Datei, da nach dem Root-Element (Store) zwei geschweifte Klammern steht und das verstößt die Wohlgeformtheit der JSON, da ein JSON-Objekt nur innerhalb von zwei geschweiften Klammern definiert werden kann.

```
{
  "people": [
    {
      "firstName": "Joe",
      "lastName": "Jackson",
      "gender": "male",
      "age": 28,
      "number": "7349282382"
    },
    {
      "firstName": "James",
      "lastName": "Smith",
      "gender": "male",
      "age": 32,
      "number": "5678568567"
    },
    {
      "firstName": "Emily",
      "lastName": "Jones",
      "gender": "female",
      "age": 24,
      "number": "456754675"
    }
  ]
}
```

Abbildung 4.6: Wohlgeformte Datei

```
{"store": [{"book": [
    {
      "category": "reference",
      "author": "Nigel Rees",
      "title": "Sayings of the Century",
      "price": 8.95
    },
    {
      "category": "fiction",
      "author": "Evelyn Waugh",
      "title": "Sword of Honour",
      "price": 12.99
    },
    {
      "category": "fiction",
      "author": "Herman Melville",
      "title": "Moby Dick",
      "isbn": "0-553-21311-3",
      "price": 8.99
    },
    {
      "category": "fiction",
      "author": "J. R. R. Tolkien",
      "title": "The Lord of the Rings",
      "isbn": "0-395-19395-8",
      "price": 22.99
    }
],
  "bicycle": {
    "color": "red",
    "price": 19.95
  }
}]}
```

Abbildung 4.7: Nicht wohlgeformte Datei

4.3.3 JSON zu XML konvertieren

Konvertierung	
JSON Konzepte	XML Konzepte
Attribut	Element
Objekt	Element
Array	Elemente
Array ohne Name	Elemente mit dem Namen Array
Array-Element	Element mit dem Array-Namen

In diesem Schritt wird die JSON-Datei zu einem XML-String konvertiert. Die Konvertierung erfolgt mittels der `XML.ToString` Methode 3.3.3. Diese Methode erzeugt die XML-String ohne Root-Element und ohne Informationen über die Version der XML und deren Encoding. Außerdem sortiert sie die Elemente anders als in JSON-Datei, das führt zu keinen Problemen, da XPath bei der Suche auf der Reihenfolge gar nicht achtet. Die Konvertierung-Regeln sind in der oben gezeigten Tabelle.

Das wichtigste ist, dass diese Methode die Elemente ohne Manipulationen konvertiert, d.h., dass sie das JSON-Objekt bzw. JSON-Liste dem entsprechenden XML-Element transformiert. Bei anderen Methoden wie `XML.Serializer` 3.3.2 werden JSON-Listen als JSON-Objekte transformiert. Allerdings braucht diese Methode keine vordefinierte Regeln oder Vorlagen, wie bei der XSLT-Ansatz 3.3.1.

Die Abbildung 4.8 zeigt die konvertierte XML-String vor den Anpassungen zu dem Beispiel in der Abbildung 4.5. Sie zeigt wie das Root-Element fehlt, ohne diese Root-Element können mittels XPath keine Daten daraus extrahiert werden, da die XPath-Struktur auf dem Root-Element bei der Suche nach den gewollten Daten basiert. Das Problem kann gelöst werden, indem das Root-Element sowie die Informationen über XML hinzugefügt werden. Diese Anpassungen erfolgen generisch. Die Abbildung 4.9 zeigt die konvertierte XML-String nach den Anpassungen.

```
|<tristate><name>tristate</name><javatype>java.lang.Boolean</javatype></tristate>
<jsonarray><name>jsonarray</name><javatype>javax.json.JsonArray</javatype></jsonarray>
<string><name>string</name><javatype>java.lang.String</javatype></string>
<gender><name>gender</name><javatype>de.wossidia.datatypes.Gender</javatype></gender>
<double><name>double</name><javatype>java.lang.Double</javatype></double>
<sigA3><name>sigA3</name><javatype>de.wossidia.datatypes.Sig$SigA3</javatype></sigA3>
<digi><name>digi</name><javatype>java.lang.Integer</javatype></digi>
<integer><name>integer</name><javatype>java.lang.Integer</javatype></integer>
<topo><name>topo</name><javatype>java.lang.Integer</javatype></topo>
<uri><name>uri</name><javatype>java.net.URI</javatype></uri>
<when><name>when</name><javatype>de.wossidia.datatypes.When</javatype></when>
<tokenlist><name>tokenlist</name><javatype>de.wossidia.datatypes.TokenList</javatype></tokenlist>
<ppn><name>ppn</name><javatype>de.uni.rostock.ub.ac.PPN</javatype></ppn>
<boolean><name>boolean</name><javatype>java.lang.Boolean</javatype></boolean>
<sig2><name>sig2</name><javatype>de.wossidia.datatypes.Sig$Sig2</javatype></sig2>
<sig1><name>sig1</name><javatype>de.wossidia.datatypes.Sig$Sig1</javatype></sig1>
<sig13><name>sig13</name><javatype>de.wossidia.datatypes.Sig$Sig13</javatype></sig13>
<muxt><name>muxt</name><javatype>de.wossidia.datatypes.Muxt</javatype></muxt>
<grade><name>grade</name><javatype>de.wossidia.datatypes.Grade</javatype></grade>
<sig3><name>sig3</name><javatype>de.wossidia.datatypes.Sig$Sig3</javatype></sig3>
<json><name>json</name><javatype>javax.json.JsonObject</javatype></json>
<wixt><name>wixt</name><javatype>de.wossidia.datatypes.Wixt</javatype></wixt>
<text><name>text</name><javatype>java.lang.String</javatype></text>
<info><name>info</name><javatype>de.wossidia.datatypes.Info</javatype></info>
```

Abbildung 4.8: XML-String vor Anpassungen

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<tristate><name>tristate</name><javatype>java.lang.Boolean</javatype></tristate>
<jsonarray><name>jsonarray</name><javatype>javax.json.JsonArray</javatype></jsonarray>
<string><name>string</name><javatype>java.lang.String</javatype></string>
<gender><name>gender</name><javatype>de.wossidia.datatypes.Gender</javatype></gender>
<double><name>double</name><javatype>java.lang.Double</javatype></double>
<sigA3><name>sigA3</name><javatype>de.wossidia.datatypes.Sig$SigA3</javatype></sigA3>
<digi><name>digi</name><javatype>java.lang.Integer</javatype></digi>
<integer><name>integer</name><javatype>java.lang.Integer</javatype></integer>
<topo><name>topo</name><javatype>java.lang.Integer</javatype></topo>
<uri><name>uri</name><javatype>java.net.URI</javatype></uri>
<when><name>when</name><javatype>de.wossidia.datatypes.When</javatype></when>
<tokenlist><name>tokenlist</name><javatype>de.wossidia.datatypes.TokenList</javatype></tokenlist>
<ppn><name>ppn</name><javatype>de.uni.rostock.ub.ac.PPN</javatype></ppn>
<boolean><name>boolean</name><javatype>java.lang.Boolean</javatype></boolean>
<sig2><name>sig2</name><javatype>de.wossidia.datatypes.Sig$Sig2</javatype></sig2>
<sig1><name>sig1</name><javatype>de.wossidia.datatypes.Sig$Sig1</javatype></sig1>
<sig13><name>sig13</name><javatype>de.wossidia.datatypes.Sig$Sig13</javatype></sig13>
<muxt><name>muxt</name><javatype>de.wossidia.datatypes.Muxt</javatype></muxt>
<grade><name>grade</name><javatype>de.wossidia.datatypes.Grade</javatype></grade>
<sig3><name>sig3</name><javatype>de.wossidia.datatypes.Sig$Sig3</javatype></sig3>
<json><name>json</name><javatype>javax.json.JsonObject</javatype></json>
<wixt><name>wixt</name><javatype>de.wossidia.datatypes.Wixt</javatype></wixt>
<text><name>text</name><javatype>java.lang.String</javatype></text>
<info><name>info</name><javatype>de.wossidia.datatypes.Info</javatype></info>
</root>
```

Abbildung 4.9: XML-String nach Anpassungen

4.3.4 XML Wohlgeformtheit prüfen

Wohlgeformtheit	
JSON	XML
<p>Das Root-Element wird durch eine geschweifte Klammer oder eine Ecke Klammer gekennzeichnet</p> <p>Das Root-Element kann keinen Elementnamen enthalten</p> <p>Eine Datei beginnt mit einer geschweiften Klammer oder mit einer Ecke Klammer und endet mit deren Schließung</p>	<p>Das XML-Dokument darf nur ein Root-Element enthalten.</p> <p>Das Root-Element muss einen Elementnamen enthalten</p> <p>Eine Datei beginnt mit dem öffnenden Root-Element Tag und endet mit dessen schließendem Tag</p>
<p>Ein Objekt beginnt mit einer geschweiften Klammer und endet mit deren Schließung</p> <p>Eine Liste beginnt mit einer Ecke Klammer und endet mit deren Schließung</p>	<p>Ein Element beginnt mit öffnendes Tag</p> <p>Ein Element endet mit schließendes Tag</p>
<p>Ein Objekt oder eine Liste darf beliebige Zeichen enthalten</p> <p>Der Bezeichner (Name) darf mit einer Zahl oder mit Sonderzeichen beginnen</p>	<p>Ein Element darf keine Sonderzeichen wie: (;) und (@) u.s.w enthalten</p> <p>Der Bezeichner (Name) darf nicht mit einer Zahl oder mit Sonderzeichen beginnen</p>
<p>Ein leeres Objekt oder eine leere Liste werden durch zwei leerende geschweiften oder Ecke klammer gekennzeichnet</p>	<p>Ein leeres Element wird durch leerendes öffnendes und schließendes Tags gekennzeichnet</p>

Nachdem JSON-Daten zu XML-Daten konvertiert wurden, müssen die Wohlgeformtheit der konvertierten Daten überprüft werden. Die Überprüfung ist erforderlich, da JSON dinge erlaubt, die die Wohlgeformtheit von XML verstößt. Die oben dargestellte

Tabelle zeigt den Wohlgeformtheit-Unterschied zwischen JSON und XML

Mit der XML-Wohlgeformtheit ist konkret gemeint, dass:

- XML-Dokument ein oder mehrere Elemente enthalten kann.
- Öffnendes (Tag) und schließendes Tag (/Tag) und leeres Element (Tag/) sind möglich.
- XML-Dokument nur ein Root-Element enthalten darf.
- Bei Öffnendes und schließendes Tag auf die Groß- und Kleinschreibung beachtet werden muss.
- Es auf die korrekte Schachtelung beachtet werden muss, keine verzahnten Elemente sind erlaubt (Wer A sagt und dann B sagt, muss erst wieder B und dann erst A sagen).
- Bezeichner (Name) eines Elements Buchstaben, Ziffern, Binde- oder Unterstriche, Doppelpunkte oder Punkte enthalten darf (Doppelpunkt nur in Zusammenhang mit Namespace).
- Bezeichner (Name) eines Elements keine XML enthalten, auch Sonderzeichen wie (;) und (@) u.s.w enthalten darf.
- Kommentare überall außer im Markup erlaubt sind.

Zum Beispiel darf der Name eines JSON-Objektes mit einer Zahl beginnen, was in XML nicht zulässig ist. Bei anderen Fällen ist es sehr unwahrscheinlich, dass sie vorkommen. da die Konvertierung unter Beachtung die Wohlgeformtheit von XML erfolgt. Bei dem Fall, dass der Name mit einer Zahl beginnt, gibt es keiner äquivalenten Form, sodass die Methode die Wohlgeformtheit von XML nicht verstößt, daher wird das Element mit

dessen Fehler erzeugt. In dieser Arbeit wurde das Problem gelöst, indem das erzeugte Element zu der Wohlgeformtheit von XML angepasst wird. Die detaillierte Erklärung ist im nächsten Abschnitt 4.3.5.

Die Abbildung 4.10 zeigt ein Beispiel von einem XML-Element, dessen Name mit einer Zahl beginnt. Das Element (870Jars) beginnt mit einer Zahl, dieses Element würde die Wohlgeformtheit nicht verstören, wenn die Elementnamen mit dem Wort (Jars) angefangen wären, sodass die Element-Name (Jars870) ist. Die Unter-Elemente verstören die Wohlgeformtheit von XML nicht.

```
<870Jars>
  <filename>v10.3.0-870Jars.zip</filename>
  <targetfilename/>
  <version>v10.3.0</version>
</870Jars>
```

Namen dürfen nicht mit einer Zahl oder einem Satzzeichen beginnen

Abbildung 4.10: XML-Element, dessen Name mit einer Zahl beginnt

4.3.5 Zu XML-Wohlgeformtheit anpassen

```
{  
  "1planung": {  
    "planer": {  
      "name": "Uni"  
    },  
    "stundenplan": [  
      {  
        "zeitraum": "SS 2016",  
        "tag": {  
          "name": "Dienstag",  
          "termin": {  
            "zeit": "13-15",  
            "veranstaltung": {  
              "typ": "Seminar",  
              "titel": "Forschungsseminar",  
              "dozent": "",  
              "raum": "312"}  
            }  
          }  
        }  
      ]  
    }  
  }  
}
```

Abbildung 4.11: JSON-Element, dessen Name mit einer Zahl beginnt

```
<?xml version="1.0" encoding="UTF-8"?>  
<root>  
  <1planung>  
    <planer>  
      <name>Uni</name>  
    </planer>  
    <stundenplan>  
      <zeitraum>SS 2016</zeitraum>  
      <tag>  
        <termin>  
          <veranstaltung>  
            <titel>Forschungsseminar</titel>  
            <raum>312</raum>  
            <dozent/>  
            <typ>Seminar</typ>  
          </veranstaltung>  
          <zeit>13-15</zeit>  
        </termin>  
        <name>Dienstag</name>  
      </tag>  
    </stundenplan>  
  </1planung>  
</root>
```

Abbildung 4.12: XML-String mit Verletzung der Wohlgeformtheit

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<E-1planung>
<planer>
<name>Unic</name>
</planer>
<stundenplan>
<zeitraum>SS 2016</zeitraum>
<tag>
<termin>
<veranstaltung>
<titel>Forschungsseminar</titel>
<raum>312</raum>
<dozent/>
<typ>Seminar</typ>
</veranstaltung>
<zeit>13-15</zeit>
</termin>
<name>Dienstag</name>
</tag>
</stundenplan>
</E-1planung>
</root>
```

Abbildung 4.13: XML-String nach Anpassung zu der Wohlgeformtheit

In diesem Schritt werden die konvertierte Daten zu der XML-Wohlgeformtheit angepasst. Dieser Schritt wird ausgeführt, nur wenn ein JSON-Objekt bzw. eine JSON-Liste vorkommt, deren Namen mit einer Zahl anfängt. Das Problem kann gelöst werden, indem ein Wort oder einen Buchstaben zu dem Elementnamen hinzugefügt wird. Damit beginnen die Elementnamen mit einem Buchstabe (was in XML zulässig ist) und nicht mit einer Zahl. Dieser Fall kann in einer JSON-Datei mehrfach vorkommen, deswegen ist es nötig jede Elementnamen zu prüfen.

Eine generische Überprüfung für jedes Element kann durch die Regex erreicht werden, mit denen Zugriff auf die XML-Elemente möglich ist. Regex (eine Abkürzung für regulärer Expression, auf Deutsch regulärer Ausdruck) sind eine Art Filterkriterium, mit deren Hilfe man Zeichenketten analysieren und manipulieren kann. Manchmal werden Sie auch Pattern (Muster) genannt [web].

Die Schritte der Anpassung sind folgendes:

- Die Software wendet die Regex ((<[0-9])) an, um die öffnende Tags, die mit einer Zahl beginnen, zu finden.
- Die Software teilt die XML-String in zwei Teilen auf, wobei der erste Teil vom Anfang des Strings zu der Match-Stelle des öffnenden Tags ist und die zweite Teil von dieser Match-Stelle bis Ende des Strings ist.
- Die Software fügt die (E-) Zeichen zwischen den beiden Teile hinzu.
- Die Software wendet die Regex ((</[0-9])) an, um die schließende Tags, die mit einer Zahl beginnen, zu finden.
- Die Software teilt die XML-String in zwei Teilen auf, wobei der erste Teil vom Anfang des Strings zu der Match-Stelle des schließenden Tags ist und die zweite Teil von dieser Match-Stelle bis Ende des Strings ist.
- Die Software fügt die (E-) Zeichen zwischen den beiden Teile hinzu.

Die hinzugefügte Zeichen (E-) bezeichnet die Abkürzung des Worts Element. Infolgedessen wird zum Beispiel die Element «E-3book» (Element-3book) gelesen. Durch die Iteration erfolgt die Anpassung generisch, d.h alle verletzte Stellen werden gesammelt und eine nach der anderer werden angepasst. Benutzer sollen beim Eingeben vom XPath-Ausdrücke darauf achten, die Zeichen (E-) vor den Namen, die mit einer Zahl anfangen, zu schreiben. Zum Beispiel wird die Element (1Planung) mit der XPath-Ausdruck: (/root/E-1plannung) abgerufen.

In der Abbildung 4.11 steht ein Beispiel von einer JSON-Datei, deren ersten Element mit einer Zahl beginnt. Die Abbildung 4.12 zeigt die erzeugte XML-String vor den Anpassungen. Die Abbildung 4.13 zeigt dieselben XML-String nach den Anpassungen.

4.3.6 XML-Dokument zu X2G übergeben

Der letzte Schritt dieser Phase ist die Umwandlung von XML-String zu XML-Dokument und dieses Dokument zu der nächsten Phase (X2G) übergeben. Die Umwandlung in XML-Dokument erfolgt mithilfe der XML-DOM-Klasse, DOM ist die Abkürzung für (Dokumentobjektmodell). Die XML-DOM- Klasse ist eine Darstellung eines XML-Dokuments im Speicher. Mit dem DOM kann ein XML-Dokument programmgesteuert gelesen und geändert werden. Der DOM erfüllt in erster Linie Bearbeitungsfunktionen. Es ist die herkömmliche strukturierte Methode, XML-Daten im Speicher darzustellen [mic]. Wegen seiner Dokument-Struktur ist der Zugriff auf die XML-Elemente, Attribute und Fragmente mit dem XML-DOM einfacher. Innerhalb der Struktur eines XML-Dokuments stellt jeder Kreis einen Knoten dar, der als XML-Node-Objekt bezeichnet wird. Das XML-Node-Objekt ist das Basisobjekt in der DOM-Struktur. Die XmlDocument-Klasse, die das XML-Node-Objekt erweitert, unterstützt Methoden zum Ausführen von Vorgängen für das gesamte Dokument (z.B. Laden des Dokuments in den Speicher oder Speichern von XML-Daten in eine Datei). Außerdem bietet die XmlDocument-Klasse eine Möglichkeit zum Anzeigen und Ändern der Knoten im gesamten XML-Dokument. Sowohl das XML-Node-Objekt als auch die XmlDocument-Klasse sind hinsichtlich der Leistung und Benutzerfreundlichkeit verbessert worden. Sie bieten zudem Methoden und Eigenschaften für folgende Zwecke:

- Zugreifen auf DOM-spezifische Knoten, z.B. Elementknoten, Entitätsverweisknoten usw.
- Abfragen ganzer Knoten zusätzlich zu den im Knoten enthaltenen Informationen, z. B. dem Text in einem Elementknoten.

Das XML-Dokument wird zu der X2G-Software übergeben, um es in Property Graph-Modell umzuwandeln.

4.3.7 JSONPath zu XPath konvertieren

Konvertierung		
JSONPath	XPath	Beschreibung
§	/root	das Stammobjekt/-element
@	.	das aktuelle Objekt/Element
.	/	das nächste Objekt/Element
==	=	ist gleich Symbol
-1:	last()	das letzte Objekt/Element
-n:	$position() < last() - n$	die letzten n-Objekte/Elemente
:n	$position() < n + 1$	die ersten n-Objekte/Elemente
&&	and	Logisches und Symbol
	or	Logisches oder Symbol
..	//	rekursiver Abstieg
[]	[]	Subscript-Operator
n	n+1	das n-ten Objekt/Element
? ()	[]	wendet einen Filterausdruck an.

Benutzer können die JSON-Daten in X2G-Software nicht nur mittels XPath filtern, sondern auch mittels JSONPath. Bei der Extraktion von gewünschten JSON-Daten können Benutzer JSONPath-Ausdrücke eingeben. Diese eingegebene JSONPath-Ausdrücke werden von X2G-Phase zu J2XML-Phase gegeben, um JSONPath zu XPath zu konvertieren. Die Transformation erfolgt unter Beachtung der obigen dargestellten Regeln, wobei fehlende JSONPath-Funktionen mit den XPath-Funktionen ersetzt werden können. Zum Beispiel fehlt in JSONPath das `text()` Methode, das die Attribut-werte extrahiert. solche Methoden können Benutzer in ihren JSONPath-Ausdruck eingeben, da der gesamte JSONPath-Ausdruck sowieso zu XPath-Ausdruck konvertiert wird. Bei der Konvertierung wird auf der Semantik und Syntax von XPath geachtet. Die oben dargestellten Regeln beschreiben den Mechanismus der Konvertierung. Um die Einhaltung von XPath Syntax und Semantik werden folgende Regeln berücksichtigt:

- Das in J2XML erzeugte XML-Dokument enthält, die Root-Element, daher wird das Symbol (§.) zu (`root/`) konvertiert.
- Der Filtern-Ausdruck in XPath enthält keine Fragezeichen, daher wird der Symbol (?) bei der Konvertierung entfernt.
- Der Filtern-Ausdruck in XPath enthält keine Klammern, daher werden Klammern bei der Konvertierung entfernt.
- Die Indexierung in XPath beginnt mit der Element-Nummer (1), daher werden Zahlen bei der Konvertierung um eins erhöht.

Als Beispiel wird der folgende JSONPath-Ausdruck:

```
$.store.book[?(@.display-price>12)].display-price.text()
```

zu dem folgendem Xpath-Ausdruck konvertiert:

```
/root/store/book[display-price>12]/display-price/text()
```

4.4 X2G

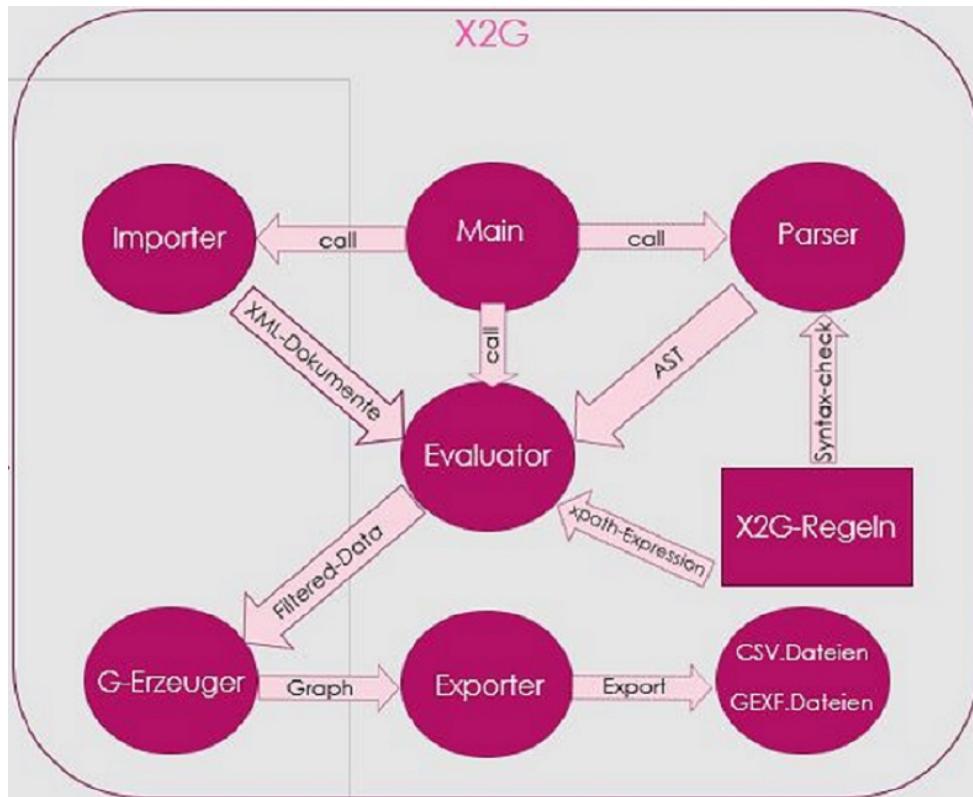


Abbildung 4.14: X2G-Architektur zweite Version

X2G steht wie bereits erwähnt für die Umwandlung von XML-Daten in property Graph-Daten. Die zweite Version von X2G besteht aus vier Phasen [Mey22]:

- Importer.
- Evaluator.
- Graph Erzeuger.
- Exporter.

Außerdem basiert X2G auf eine Regelspezifikation-Sprache, mit der Benutzer Ihre Regeln zur Selektion bestimmter XML-Daten und Erstellung von Knoten und Kanten definieren können.

4.4.1 Importer

Die erste Phase dieses Tools ist der Importer. Er importiert die XML-Daten und wandelt sie in XML-Dokumente um. Für eine XML-Datei gibt es zwei Formen:

- XML-Datei mit Namespaces (Namenräume).
- XML-Datei ohne Namespaces (Namenräume).

Der Importer prüft zunächst, ob die XML-Datei Namespaces (Namenräume) enthält. Bei der enthaltung von Namespaces wird die XML-Datei in XML-Dokument mittels SAX-Parser umgewandelt und deren Namespaces werden extrahiert.

Ein Namespace ist eine Reihe von einzigartigen Namen. Namespace ist eine Mechanismen, durch welche Element und Attribut-Namen zu einer Gruppe zugeordnet werden können. Der Namespace wird durch URI (Uniform Resource Identifier) identifiziert. Ein Namespace wird mit reservierten Attribute erklärt. so ein Attribut Name muss entweder mit (xmlns) oder mit (xmlns:) beginnen. Das Wort nach dem xmlns: ist der Namespace-Präfix. Die URI ist der Namespace-Kennung. [tutb].

Der SAX-Parser ist ein ereignisbasierter Parser für XML-Dokumente. Im Gegensatz zu einem DOM-Parser erstellt ein SAX-Parser keinen Parsebaum. SAX ist eine Streaming-Schnittstelle für XML, was bedeutet, dass Anwendungen, die SAX verwenden, Ereignisbenachrichtigungen über das XML-Dokument erhalten, das ein Element und ein Attribut verarbeitet, und zwar zu einem Zeitpunkt in sequenzieller Reihenfolge, beginnend am Anfang des Dokuments und endend mit dem Schließen des Root-Elements [tuta].

Wenn die XML-Datei keine Namespaces enthält, wird sie dann in XML-Dokument mittels DOM-Parser umgewandelt. Zuletzt werden die XML-Dokumente zu der Evaluator geschickt.

4.4.2 Evaluator

Die zweite Phase der X2G-Software ist die Evalualtion. Diese Phase erfüllt zwei Aufgaben:

- X2G-Regeln semantisch und syntaktisch überprüfen.
- Matching die ausgewählten XML-Dateien mit den eingegebenen Xpath-Ausdrücken überprüfen.

Der X2G-Regelsprache Syntax wird in dieser Phase überprüft. Die von Benutzer eingegebene Regeln werden semantisch und Syntaktisch evalualiert. Ablauf der Evalualtion für die X2G-Regeln ist wie folgt:

- Die Evalualtion beginnt mit dem Parser Aufruf.
- Es werden Syntax und einige semantische Überprüfungen für Bindungsvariablen durchgeführt.
- Als nächstes wird eine Symboltabelle erstellt und wenn keine Fehler vorliegen, wird die AST (Abstrakte Syntax Baum) generiert.
- Basierend auf der Variablenbindung werden die Ausdrücke des AST überprüft, wenn fehler vorliegen, wird die Software die Benutzer darauf hinweisen.

Die zweite Aufgabe des Evaluators ist die Matching-überprüfung. Es wird geprüft, ob die die ausgewählten XML-Dateien mit den eingegebenen Xpath-Ausdrücken übereinstimmen.

Der Ablauf sieht folgendes aus:

- Benutzer wählen die XML-Dateien aus und geben Ihre Xpath-Ausdrücke ein.
- Die Software prüft, ob für die eingegebenen Xpath-Ausdrücke Daten in den ausgewählten Dateien vorliegen, bei keinen Match zeigt die Software einen Fehler.

Zuletzt werden die extrahierten Daten zu dem Graph Erzeuger geschickt.

4.4.3 Regelsprache

die X2G-Regelsprache wurde entwickelt zum Zuordnen von XML-Dokumenten zu Property Graph Modelle. Die Regelsprache weist die folgende Hauptkonzepte:

- Extrahieren von XML-Fragmenten nach XPath-Ausdruck, JSON-Fragmenten nach JSON-Pfad und relationalen Daten nach SQL-Abfragen.
- extrahierte Daten werden an Variablen gebunden.
- Generierung von Knoten und Kanten aus Literalen und Auswertung von Variablenbindungen.
- Verschachtelte Auswertung von Anweisungen im Kontext anderer Anweisungen und Variablenbindungen. Regeln haben einen Kopf, der einzigen XPath-Ausdrücken oder Knoten- oder Kantenbeschriftungen entspricht. Der Text kann lokale Übereinstimmungs-Regeln oder Knoten- und Kantengenerierungs-Anweisungen enthalten.
- Regeln auf der obersten Ebene werden in der Reihenfolge des Dokuments ausgewertet.
- Verschachtelte Regeln werden im Kontext der Regel ausgewertet, die sie enthält.
- Es gibt keine Ausnahmebehandlung durch Design. Wenn beispielsweise ein XPath-Ausdruck mit nichts in den XML-Dokumenten übereinstimmt, passiert nichts d.h. Vom Regeltext werden keine Knoten und Kanten generiert.

4.4.3.1 XML-Extraktion mit match und XPath-Ausdrücken

Die allgemeine Syntax lautet:

```
match <optional-var-ref> xpath (<>xpath-expr>) using <var-binding> { <body> }
```

Der Hauptteil der Anweisung wird für jedes XML-Fragment ausgewertet, das mit dem XPath-Ausdruck übereinstimmt. Die Klausel ermöglicht das Binden einer Variablen an jede der Übereinstimmungen. Innerhalb des Textkörpers kann alternativ über die gebundene Variable oder durch relative XPath-Ausdrücke auf die aktuelle Bindung zugegriffen werden. Ein relativer XPath beginnt normalerweise mit wie in der zweiten Anweisung im folgenden Beispiel:

```
match <xpath-expr> using <var-binding> '.' match <body>
```

können mehrere weitere sein, und generieren oder bedingte Anweisungen

```
match node edge
```

Es gibt andere Varianten der Anweisung, die keine xpath-Ausdrücke verwenden, sondern z.B. alle Knoten mit einer bestimmten Bezeichnung abgleichen:match

```
match node (<string-expr>) using <var-binding> { <body> }
```

Durch diese Aussage ist es möglich, einen bestimmten Satz bestehender Knoten durchzulaufen, z.B. um weitere Kanten zu erzeugen. Die folgende Ausdrücke sind die Varianten der Übereinstimmungsausdrücke:

- `xpath(<xpath-expr>)`
- `jpath(<jpath-expr>)`
- `sql(<sql-expr>)`
- `node(<string-expr>)`
- `edge(<string-expr>)`

Außerdem kann man mehrere Übereinstimmungsausdrücke in einer Anweisung auf die folgende weise kombinieren:

```
match <match-expression>, <match-expression>, ... { <body> }
```

Die Auswertung einer solchen Aussage ist wie folgt: Jede wird berechnet und der Körper wird für alle möglichen Kombinationen der resultierenden Variablenbindungen ausgewertet.

```
match <match-expression>
```

Die Abbildung 4.15 zeigt Ein laufendes Beispiel der X2G-Regelsprache

```
// nested match example, the second match is evaluated within the context of the first
match xpath("//story") using $s {
    create node $sn label "story" {
        // properties
        content = xpath("./content/text()"),
        title = xpath("./title/text()"),
        unique (title)
    }
    match $s.xpath("././person[@role='narrator'])" using $p {
        create node $pn label "person" {
            // properties
            name = $p.xpath("name/text()"),
            if ($p.xpath("appellation/text())" == "Mr") {
                gender = "male"
            }
            else {
                gender = "female"
            }
        }
        create edge $e from $sn to $pn label "narrator" { /* no properties given */ }
    }
}
```

Abbildung 4.15: Ein laufendes Beispiel der X2G-Regelsprache, Quelle: [Mey22]

4.4.3.2 Knoten-Generierung Anweisung

Die node-Anweisung hat die allgemeine Form von:

```
create node <var-binding> label <string-expr>
{ <optional-property-list> <optional-unique-clause> }
```

Für jeden eindeutigen kombinierten Wert wird ein neuer Knoten generiert. Die generierte eindeutige Knoten-ID wird dann an die Variable gebunden. Wenn bereits ein Knoten mit derselben Bezeichnung und demselben Satz von Eigenschaftswerten vorhanden ist, wird seine Knoten-ID gebunden, und es wird kein neuer Knoten generiert. Wenn ein gegeben ist, steuert nur der Satz der angegebenen Eigenschaften die Generierung neuer Knoten, d.h. es wird kein neuer Knoten generiert, wenn bereits ein Knoten mit den gleichen Werten für die eindeutigen Eigenschaften vorhanden ist.

```
<string-expr> <optional-property-list> <var-binding><optional-unique-clause>
```

4.4.3.3 Kanten-Generierung Anweisung

Eine gerichtete Kante von einem Knoten zu einem anderen wird durch folgende Anweisung erzeugt:

```
create edge <var-binding> from <node-reference>
to <node-reference> label <string-expr> { <optional-property-list> }
```

Sowohl die als auch die Klausel müssen auf vorhandene Knoten durch die gegebenen verweisen, d. h. auf bereits generierte Knoten. A ist in der Regel eine gebundene Knotenvariable aus der Knotenanweisung, die zuvor ausgewertet wurde.

```
from to <node-reference> <node-reference>
```

4.4.3.4 Variable Bindungen und Ausdrücke

Es gibt drei Arten von variablen Bindungen:

- Pfadvariablen, die durch XPath-Ausdrücke an XML-Fragmente gebunden sind.
- `create node node (<label>)`

Knotenvariablen, die durch oder an Knoten gebunden sind.

- `create edge`

Kantenvariablen, die durch an Kanten gebunden sind.

Eine Pfadvariable kann verwendet werden, um über einen XPath-Unterausdruck weiter auf alle Elemente oder Attribute zuzugreifen, z. B.:

```
$<var>/<xpath-expression>
```

Ein solcher Ausdruck stammt aus dem obigen Beispiel 4.14, der die Zeichenfolge des Unterelements aus einem Fragment extrahiert:

```
$p/name/text() name person
```

Knoten- und Kantenvariablen können auf die zugewiesenen Eigenschaften zugreifen. Es gibt einige reservierte Attribute / Eigenschaften, die sind:

- `id`: So greifen Sie auf die ID des Knotens zu.
- `label`: Zurückgeben der Knoten- oder Kantenbezeichnung.
- `from-to`: und stellt die Quell- und Zielknoten-IDs einer Kante dar.

Um XPath von Knoten- und Kantenvariablen zu unterscheiden, wird eine Punktnotation für den Zugriff auf die certain-Eigenschaften verwendet:

```
$n.id$e.from
```

4.4.3.5 Property Listen

Sowohl Knoten als auch Kanten können eine Reihe von Eigenschaften aufweisen. Jede Eigenschaft hat einen Namen und einen zugewiesenen Wert. Eigenschaften werden Knoten und Kanten innerhalb von Anweisungen zugewiesen. Die Werte sind Literale oder Ausdrücke aus der Auswertung von Variablenbindungen. Hier ist eine Property-Liste aus dem laufenden Beispiel:

```
{ // property list  
    content = $s.xpath("content/text()") ,  
    title = $s.xpath("title/text()")  
}
```

4.4.3.6 Kommentare

Kommentar kann wie in C ++ oder Java verwendet werden wie:

- ein Kommentar, der mit // beginnt und bis zum Ende der Zeile.
- mehrzeilige Kommentare, die eingeschlossen sind. /**/

4.4.4 Graph Erzeuger

In dieser Phase werden die gefilterten Daten empfangen und dann Knoten und Kanten generiert. Die Generierung von Knoten bzw. Kanten erfolgt unter Beachtung von bestimmten Regeln. Die Erstellung eines Knoten bedingt nach Labels und Eigenschaften, in der X2G ersten Version wurde die Grundlage für die Entscheidung, wann einen neuen Knoten erstellt werden muss oder nicht festgelegt. Im Allgemein sind Zwei Knoten (ein bereits erstellter Knoten und die Label und Eigenschaften eines potentiellen neuen Knotens) gleich, wenn die Labels und alle Eigenschaften die gleichen Werte für denselben Eigenschaftsnamen haben. Außerdem, wenn es eine spezielle (unique) Eigenschaft gibt, dessen Wert eine Liste von Eigenschaftsnamen ist, die zur Überprüfung anstelle von allen Eigenschaften. Dies führt zu folgenden Fällen:

- Sei (A) die Menge der Eigenschaften des bereits erstellten Knotens und (B) die Menge der Eigenschaften des Knotens, der erstellt werden soll. Wenn A == B und alle Immobilien haben paarweise gleiche Werte, wird der bereits erstellten Knotens zurückgegeben.
- Wenn es eine Eigenschaft (unique) gibt, muss sie sich sowohl in (A) als auch in (B) befinden mit den gleichen Eigenschaftsnamen in beiden Knoten. Alle eindeutigen Eigenschaften müssen die gleichen Werte haben. Die anderen Eigenschaften werden zu einer Obermenge verschmolzen.
- Wenn A » B und alle Eigenschaften (B) haben die gleichen Werte wie in (A), wird der bereits erstellten Knotens zurückgegeben.
- Wenn A « B und alle Eigenschaften (A) haben die gleichen Werte wie in (B), wird A durch B ersetzt und wird der bereits erstellten Knotens zurückgegeben.
- Ansonsten wird einen neuen Knoten mit einer gegebenen Label und Eigenschaften erstellt.

Die Erstellung von Kanten erfolgt unter Beachtung bestimmte Regeln:

- Eine Kante vom Quell-Knoten zum Ziel-Knoten wird generiert, wenn bereits keine Kante zwischen denen existiert .
- Eine Kante vom Quell-Knoten zum Ziel-Knoten wird generiert, wenn bereits eine Kante vom selben Quell-Knoten zum selben Ziel-Knoten existiert aber mit einem anderen Label.
- Eine Kante vom Quell-Knoten zum Ziel-Knoten wird nicht generiert, wenn bereits eine Kante vom selben Quell-Knoten zum selben Ziel-Knoten existiert aber mit dem gleichen Label.
- Es existiert kein Multigraph (Graph mit mehreren Kanten zwischen zwei Knoten), wenn im Graph keine mehrere Kanten zwischen einem Quell-Knoten und einem Ziel-Knoten existieren.
- Es existiert kein Multigraph, wenn eine Kante mit einem Label vom Quell-Knoten zum Ziel-Knoten existiert und eine andere Kante mit dem gleichen Label generiert werden muss.
- Es existiert ein Multigraph, wenn eine Kante mit einem Label vom Quell-Knoten zum Ziel-Knoten existiert und eine andere Kante mit einem anderen Label generiert werden muss.

4.4.5 Exporter

Node.csv	Edge.csv
<pre>"id","label","type","properties" "1","xmd_s001_000_000_005","story","" "2","place","place","Name: Stellshagen" "3","place","place","Name: Hohen Viecheln" "4","person","person","Name: Hinrichs, gender: male, profession: narrator, role: Pantoffelmacher" "5","xmd_s001_000_000_006","story","" "6","place","place","Name: Grevesmühlen" "7","person","person","Name: Dettmann, gender: male, profession: narrator, role: Inspektor"</pre>	<pre>"id","source","target","label" "1","1","2","place" "2","1","3","person" "3","1","4","person" "4","5","6","place" "5","5","7","person" "6","5","4","person"</pre>

Abbildung 4.16: Beispiel für eine CSV-Ausgabe, Quelle: [Zak22]

Der letzte Schritt der X2G-Software ist das Graph-Exportierung. Nachdem gewünschte Knoten und Kanten generiert wurden, werden sie nun in CSV-Dateien gespeichert. Das erzeugte Graph wird in zwei verschiedenen CSV-Dateien gespeichert (Nodes.csv und Edges.csv). CSV (Comma-separated values) hat verschiedene Aufbau-Strukturen, innerhalb der Textdatei haben einige Zeichen eine Sonderfunktion zur Strukturierung der Daten. Ein Zeichen wird zur Trennung von Datensätzen benutzt. Dies ist in der Regel der Zeilenumbruch. Für die Trennung von Datenfeldern (Spalten) gibt es verschiedene Möglichkeiten:

- Semikolon.
- Doppelpunkt.
- Tabulatorzeichen.
- Leerzeichen.
- Komma.

Um Sonderzeichen innerhalb der Daten nutzen zu können (z.B. Komma in Dezimalzahlwerten), wird ein Feldbegrenzerzeichen benutzt. Normalerweise ist dieser Feldbegrenzer das Anführungszeichen. Die gespeicherte CSV-Dateien in X2G benutzen das Komma-Zeichen, um Datenfeldern zu trennen.

Ein Nodes.csv-Datei enthält die folgende Daten:

- Id des Knotens.
- Label des Knotens.
- Knoten-Typ.
- Properties des Knotens.

In der ersten Zeile der Datei stehen die Knoten-Elemente und in den restlichen Zeilen stehen die Elemente-Werte.

Ein Nodes.csv-Datei enthält die folgende Daten:

- Id der Kante.
- Quell-Knoten.
- Ziel-Knoten.
- Label der Kante.

Die Daten werden genauso wie in der Nodes.csv-Datei gespeichert. Benutzer können schließlich diese Dateien herunterladen.

4.5 Beispiel Szenario

Das Beispiel Szenario der integrierte Lösung besteht aus vier Schritten:

- URL Eingabe.
- JSON-Daten zu XML-Daten mittels J2XML konvertieren.
- Eigene Regeln definieren in X2G-Software, um gewünschte Knoten und Kanten erzeugen zu lassen.
- Das erzeugte Property Graph in CSV-Datei herunterladen.

Das Beispiel Szenario beginnt mit der URL Benutzer-Eingabe. Damit wird die J2XML-Phase aufgerufen , die dann die JSON-Daten vom Server holt wie auf der Abbildung 4.17 zu sehen. Der zweite Schritt ist die Konvertierung, und das erfolgt in der J2XML-Phase. in dieser Phase werden JSON-Daten zu XML-Daten konvertiert und als XML-Dokument zu X2G-Software weitergegeben wie in der Abbildung 4.18 zu sehen.

Der nächste Schritt ist das Regeln Definieren mittels der spezifikation-Regelsprache in der X2G-Software. Zunächst können Benutzer entscheiden, ob die Extraktion durch JSON oder Xpath erfolgt. Als nächstes definieren Benutzer Ihren Regeln (Knoten und Kanten erzeugen, ... usw). Die Abbildung 4.18 zeigt ein Beispiel für die Anwendung von der X2G-Software, wobei die Daten mittels Xpath gefiltert werden. in dem Beispiel werden Knoten mit der Bezeichnung (cd) erzeugt und andere Knoten mit der Bezeichnung (artist). Benutzer können die Bezeichnungen aber anders definieren. Das zweite Match ist von dem ersten Abhängig, d.h die zweite Extraktion der gewünschten Daten aus der ertsnen Extraktion erfolgt. Nachdem Matching können Benutzer Knoten erstellen und sie an Variablen binden:

```
match xpath("//cd") using $c
```

```
match $c.xpath("artist") using $a
```

Für diese Knoten können Benutzer nun Properties definieren:

```
// properties  
title = $c.jpath("$.title.text()"),  
src = $c.jpath("$.src.text()"),  
avail = true,  
unique (title)
```

Die letzte Definition ist die Kanten. Aus der definierten Knoten können Benutzer Kanten erzeugen:

```
create edge $e from $an to $cn label "interpret"
```

Bei der Definition von dem Quelle und Ziel-Knoten in der Kante-Definition reicht es aus, deren Variablen-Bindungen einzugeben. Der letzte Schritt ist die Speicherung von dem erzeugten Graph. Das erzeugte Graph wird in zwei CSV-Dateien gespeichert (Nodes.csv und Edges.csv) 4.21, 4.22.

Die Abbildung 4.20 ist Analog zu der Abbildung 4.19 aber die Daten werden mit JSON-path extrahiert.

```
{  
  "catalog": {  
    "cd": [  
      {  
        "id": "1",  
        "artist": "Sufjan Stevens",  
        "title": "Illinois",  
        "src": "http://www.sufjan.com/"  
      },  
      ...  
    ]  
  }  
}
```

Abbildung 4.17: Schritt 1: Die JSON-Daten aus der gegebenen URL

```
<?xml version="1.0" encoding="UTF-8"?>  
<root>  
<catalog>  
<cd>  
<artist>Sufjan Stevens</artist>  
<src>http://www.sufjan.com/</src>  
<id>1</id>  
<title>Illinois</title>  
</cd>  
...  
</catalog>  
</root>
```

Abbildung 4.18: Schritt 2: Die XML-Datei

```
// nested match example, the second match is evaluated within the context of the first
match xpath("//cd") using $c {
    create node $cn label "cd" {
        // properties
        title = $c.xpath("title/text()"),
        src = $c.xpath("src/text()"),
        avail = true,
        unique (title)
    },
    match $c.xpath("artist") using $a {
        create node $an label "artist" {
            name = $a.xpath("text()"),
            artist = $cn.src,
            unique (name)
        },
        create edge $e from $an to $cn label "interpret" {
            alt = "disc-artist"
        }
    }
}
```

Abbildung 4.19: Schritt 3: Daten Extraktion mittels Xpath

```
// nested match example, the second match is evaluated within the context of the first
match jpath("$.cd") using $c {
    create node $cn label "cd" {
        // properties
        title = $c.jpath("$.title.text()"),
        src = $c.jpath("$.src.text()"),
        avail = true,
        unique (title)
    },
    match $c.jpath("artist") using $a {
        create node $an label "artist" {
            name = $a.jpath("text()"),
            artist = $cn.src,
            unique (name)
        },
        create edge $e from $an to $cn label "interpret" {
            alt = "disc-artist"
        }
    }
}
```

Abbildung 4.20: Schritt 3: Daten Extraktion mittels JSONpath

```
id,label,properties
5,cd,avail,true,src,http://www.stoatmusic.com/,title,Future come and get me
9,cd,avail,true,src,http://www.led-zeppelin.com/,title,"IV, Four Symbols"
1,cd,avail,true,src,http://www.led-zeppelin.com/,title,The Song Remains the Same
3,cd,avail,true,src,http://www.sufjan.com/,title,Illinois
7,cd,avail,true,src,http://www.whitestripes.com/,title,Get behind me satan
4,artist,artist,http://www.sufjan.com/,name,Sufjan Stevens
8,artist,artist,http://www.whitestripes.com/,name,The White Stripes
2,artist,artist,http://www.led-zeppelin.com/,name,Led Zeppelin
6,artist,artist,http://www.stoatmusic.com/,name,Stoat
```

Abbildung 4.21: Schritt 4: Nodes.csv

```
src-id,dst-id,label,properties
2,9,interpret,alt,disc-artist
6,5,interpret,alt,disc-artist
8,7,interpret,alt,disc-artist
4,3,interpret,alt,disc-artist
2,1,interpret,alt,disc-artist
```

Abbildung 4.22: Schritt 3: Edges.csv

5 Zusammenfassung

Die Verarbeitung von Daten in JSON-Dateien oder bestimmter Aspekte davon ist kompliziert, da der Zugriff auf ihre internen Daten nicht einfach ist. Außerdem ist die Verknüpfung von mehreren Dokumenten schwierig, sodass Forscher*innen Schwierigkeiten mit Analysieren, Untersuchen, sowie das Durchsuchen von einer massiven gesammelten Menge von JSON-Dateien begegnen, daher fehlt die Genauigkeit der Analyse von diesen Daten und die Arbeit an denen stark wird eingeschränkt.

Die Arbeit zielte darauf ab, die Erstellung von Graphen aus JSON-Dokumenten basierend auf den Eingaben der Benutzer zu ermöglichen. Für dieses Ziel wurde ein Tool mit dem Namen (J2G) konzipiert, das aus zwei Phasen besteht (J2XML und X2G). Die entwickelte Lösung lässt sich als Pipeline definieren, der mit JSON beginnt und mit Property Graph endet und dazwischen liegt die XML-Phase (JSON-XML-Property Graph). Für die Extraktion von JSON-Daten können entweder JSONPath oder XPath benutzt werden. Trotz der Schwierigkeiten beim Umgehen mit JSONPath, können bestimmte gewünschte (Attribute, Elemente) dank der JSONPath zu XPath Konverter extrahiert werden.

Die erste Phase der Lösung (J2XML) kann separat als JSON zu XML Konverter eingesetzt werden. Dank der Regelsprache in X2G-Software können aus den gewählten Daten beliebige Regeln für die Erstellung von Knoten und Kanten definiert werden. Das Property Graph wird in CSV-Dateien gespeichert werden.

6 Ausblick

Das J2G Tool kann JSON-Dokumente in Property Graph umgewandelt werden. Das J2G Tool kann JSON-Daten aus verschiedenen Quellen (URL) empfangen. Als Eingabe akzeptiert das J2G Tool auch neben der XPath-Abfrage-Sprache die JSONPath-Abfrage-Sprache, um bestimmte JSON-Daten zu extrahieren. Ein Teil des Tools kann separat verwendet werden, um JSON-Dokumente zu XML-Dokumente umzuwandeln. Als Ausgabe-Format wird das erzeugte Graph in J2G Tool in CSV-Dateien gespeichert, Ziel war es aber das Graph nicht nur in CSV-Dateien zu speichern, sondern auch in GEXF-Dateien. Wegen der Zeit-Einschränkung konnte das nicht realisiert werden.

Das J2G Tool kann doch mit den bereits beschriebenen Eigenschaften eingesetzt werden, jedoch kann das Tool noch verbessert werden indem:

- Das Graph mit anderen Datei-Formate exportieren werden könnte.
- Das Einfügen von Elementen durch Eingabefeld erlaubt würde.
- GUI hinzugefügt wird, um Anwendung zu vereinfachen.
- Die verschiedene Daten aus verschiedenen Daten-Formaten als Eingabe-Quelle erlaubt wird.
- JSON-Dokumente in Hyper-Graphen umgewandelt werden können.
- Ein Assistent hinzugefügt wird, um Benutzer beim Erstellen von Knoten und Kanten zu helfen.

Abbildungsverzeichnis

Abbildungsverzeichnis

1.1	Beispiel von WossiDiA-Webanwendung, Quelle: [MSS14]	6
2.1	Beispiel (1) von Property Graph, Quelle: [neob]	7
2.2	Beispiel (2) von Property Graph [neoa]	9
2.3	JSON-Konzepte	12
2.4	JSON-Wurzel, Quelle: [gee]	13
2.5	JSON-Objekt, Quelle: [cob]	14
2.6	JSON-Liste, Quelle: [stab]	15
2.7	JSON-Objekt-Value, Quelle: [CM17]	15
2.8	JSON-List-Value, Quelle: [CM17]	16
2.9	JSON-Values, Quelle: [CM17]	16
2.10	JSON-Schema	17
3.1	JSONPath-Funktionen, Quelle: [goe]	20
3.2	X2G Ablaufdiagramm erste Version	24
3.3	in einem Tag gepacktes JSON-Beispiel, Quelle: [ral]	26
3.4	XSLT-Vorlage für eine JSON-Datei, Quelle: [ral]	26
3.5	XSLT-Ergebnis, Quelle: [ral]	27
3.6	JSON-Beispiel, Quelle: [staa]	30
3.7	Das erzeugte XML-Datei, Quelle: [staa]	30
4.1	Ablaufdiagramm des integrierten Ansatzes (J2XML+X2G)	34
4.2	J2XML	36
4.3	J2XML, strukturelle Konvertierung	37
4.4	URL: WossidiA-Datatypes	39
4.5	Das von der Software gespeicherte String	39
4.6	Wohlgeformte Datei	41
4.7	Nicht wohlgeformte Datei	41
4.8	XML-String vor Anpassungen	43
4.9	XML-String nach Anpassungen	44
4.10	XML-Element, dessen Name mit einer Zahl beginnt	47
4.11	JSON-Element, dessen Name mit einer Zahl beginnt	48
4.12	XML-String mit Verletzung der Wohlgeformtheit	48

4.13 XML-String nach Anpassung zu der Wohlgeformtheit	49
4.14 X2G-Architektur zweite Version	54
4.15 Ein laufendes Beispiel der X2G-Regelsprache, Quelle: [Mey22]	59
4.16 Beispiel für eine CSV-Ausgabe, Quelle: [Zak22]	65
4.17 Schritt 1: Die JSON-Daten aus der gegebenen URL	69
4.18 Schritt 2: Die XML-Datei	69
4.19 Schritt 3: Daten Extraktion mittels Xpath	70
4.20 Schritt 3: Daten Extraktion mittels JSONpath	70
4.21 Schritt 4: Nodes.csv	71
4.22 Schritt 3: Edges.csv	71

Literaturverzeichnis

- [AAB⁺17] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, 50(5), sep 2017.
- [acp] a-coding project. Xslt-tutorial, tipps tricks. <https://www.a-coding-project.de/ratgeber/xslt>. Last accessed: Juli 29, 2022.
- [AW10] Charu Aggarwal and Haixun Wang. *Managing and Mining Graph Data*, volume 40. 01 2010.
- [bae] baeldung. Introduction to jsonpath. <https://www.baeldung.com/guide-to-jayway-jsonpath>. Last accessed: Juli 29, 2022.
- [BFVY18] Angela Bonifati, G.H.L. Fletcher, Hannes Voigt, and N. Yakovets. *Querying graphs*. Morgan Claypool Publishers, 2018.
- [Bra14] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, March 2014.
- [CH06] Diane Cook and Lawrence Holder. Mining graph data. 04 2006.
- [CM17] Douglas Crockford and Chip Morningstar. Standard ecma-404 the json data interchange syntax, 12 2017.
- [cob] cobol. Redvers cobol json interface. https://www.cobol.de/cobol_json_interface.php. Last accessed: Juli 29, 2022.
- [CYM20] Hirokazu Chiba, Ryota Yamanaka, and Shota Matsumoto. G2gml: Graph to graph mapping language for bridging rdf and property graphs. In Jeff Z. Pan, Valentina Tamma, Claudia d’Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web – ISWC 2020*, pages 160–175, Cham, 2020. Springer International Publishing.
- [di] dev insider. Was ist json? <https://www.dev-insider.de/was-ist-json-a-702243/>. Last accessed: Juli 29, 2022.
- [gee] geekflare. 20 online-tools für json-editor, -parser und -formatierer. <https://geekflare.com/de/json-online-tools/>. Last accessed: Juli 29, 2022.

- [goe] goessner. Jsonpath - xpath for json. <https://goessner.net/articles/JsonPath/>. Last accessed: Juli 29, 2022.
- [gra] graphdatamodeling. What is a graph anyway? <http://graphdatamodeling.com/Graph%20Data%20Modeling/GraphDataModeling/page/PropertyGraphs.html>. Last accessed: Juli 29, 2022.
- [Har14] Olaf Hartig. Reconciliation of rdf* and property graphs. *CoRR*, abs/1409.3288, 2014.
- [Mey22] Holger J Meyer. X2G — A Tool for Mapping NoSQL Data into Property Graph Model. Technical Report CS-2-22, University of Rostock, Computer Science Department, 2022.
- [mic] microsoft. Xml-dokumentobjektmodell (dom). <https://docs.microsoft.com/de-de/dotnet/standard/data/xml/xml-document-object-model-dom>. Last accessed: Juli 29, 2022.
- [mon] mongodb. Json schema examples tutorial. <https://www.mongodb.com/basics/json-schema-examples>. Last accessed: Juli 29, 2022.
- [MSH17] Holger Meyer, Alf-Christian Schering, and Andreas Heuer. The hydra.powergraph system. *Datenbank-Spektrum*, 17(2):113–129, 2017.
- [MSS14] Holger Meyer, Alf-Christian Schering, and Christoph Schmitt. Wossidia — the digital wossidlo archive. 2014.
- [neoa] neo4j. Graph database concepts. <https://www.neo4j.com/docs/getting-started/current/graphdb-concepts/>. Last accessed: Juli 29, 2022.
- [neob] neo4j. What is a graph database? <https://neo4j.com/developer/graph-database/>. Last accessed: Juli 29, 2022.
- [PRS⁺16] Felipe Pezoa, Juan Reutter, Fernando Suarez, Martin Ugarte, and Domašo Vrgoč. Foundations of json schema. pages 263–273, 04 2016.
- [ral] ralph.blog.imixs. How to convert json to xml with xslt 3.0. <https://ralph.blog.imixs.com/2019/08/05/how-to-convert-json-to-xml/>. Last accessed: Juli 29, 2022.
- [RN10] Marko Rodriguez and Peter Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, 08 2010.

- [staa] stackoverflow. Convert xml to/from json in java (without extra `je_` and `jo_` elements). <https://stackoverflow.com/questions/7472282/convert-xml-to-from-json-in-java-without-extra-e-and-o-elements>. Last accessed: Juli 29, 2022.
- [stab] stackoverflow. json-how to properly create json array. <https://stackoverflow.com/questions/14575433/json-how-to-properly-create-json-array>. Last accessed: Juli 29, 2022.
- [sub] submitfile. Informationen zu gephi. <https://submitfile.com/de/download/gephi>. Last accessed: Juli 29, 2022.
- [tab] tabnine. How to use jsonpath in com.jayway.jsonpath. <https://www.tabnine.com/code/java/classes/com.jayway.jsonpath.JsonPath>. Last accessed: Juli 29, 2022.
- [TAS⁺19] Dominik Tomaszuk, Renzo Angles, Łukasz Szeremeta, Karol Litman, and Diego Cisterna. Serialization for property graphs. In Stanisław Kozielski, Dariusz Mrozek, Paweł Kasprowski, Bożena Małysiak-Mrozek, and Daniel Kostrzewa, editors, *Beyond Databases, Architectures and Structures. Paving the Road to Smart Data Processing and Analysis*, pages 57–69, Cham, 2019. Springer International Publishing.
- [tlc] tlcpv. Bestes / mehr standard-dateiformat für die grafikdarstellung? (graphson, gexf, graphml?). <https://tlcpv.org/876923-best-more-standard-graph-representation-QRDKQV>. Last accessed: Juli 29, 2022.
- [TP18] Dominik Tomaszuk and Karol Pak. Reducing vertices in property graphs. *Plos One*, 13(2):1–25, 2018.
- [tuta] tutorialspoint. Java sax parser - Übersicht. https://www.tutorialspoint.com/java_xml/java_sax_parser.htm. Last accessed: August 22, 2022.
- [tutb] tutorialspoint. Xml - namespaces. https://www.tutorialspoint.com/de/xml/xml_namespaces.htm. Last accessed: August 22, 2022.
- [UDNS16] J. L. Usó-Doménech and J. Nescolarde-Selva. What are belief systems? *Foundations of Science*, 21(1):147–152, Mar 2016.
- [vsZMEO20] Genoveva vargas solar, José Luis Zechinelli Martini, and Javier Espinosa-Oviedo. *Enacting Data Science Pipelines for Exploring Graphs: From Libraries to Studios*, pages 271–280. 08 2020.

- [web] webmasterpro. Regex: Einführung in reguläre ausdrücke, syntax und deren verwendung. <https://www.webmasterpro.de/coding/einfuehrung-in-regular-expressions/>. Last accessed: Juli 29, 2022.
- [Wik22] Wikipedia. Javascript object notation — wikipedia, die freie enzyklopädie. https://de.wikipedia.org/w/index.php?title=JavaScript_Object_Notation&oldid=224659887, 2022. [Online; Stand 30. Juli 2022].
- [Zak22] Safwat Zakkor. Xml to graph mapping tool. *University of Rostock*, pages 1–57, 2022.

Eidesstattliche Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Rostock, 15. September 2022

Kazzaz Abdulrahman