



# XML to Graph Mapping Tool

---

Bachelorarbeit

**Safwat Zakkor**  
217205264

Eingereicht von:

**Safwat Zakkor**  
217205264

Gutachter:

**Dr.Holger Meyer**  
Holger Meyer (PhD), U of Rostock, CSEE, Database Research Group

# Inhaltsverzeichnis

<b>1 Einführung</b>	<b>5</b>
1.1 Motivation . . . . .	6
1.2 Problemstellung . . . . .	6
1.3 Verwandte Arbeiten . . . . .	7
<b>2 XML- und Graphdatenmodelle</b>	<b>8</b>
2.1 XML-Allgemein . . . . .	8
2.1.1 Namespace in XML-Dokumenten . . . . .	10
2.1.2 Schema von XML-Dokumenten . . . . .	11
2.1.3 DOM-Dokument Objekten Modell . . . . .	14
2.1.4 XML und XSLT-Transformationstechniken . . . . .	15
2.1.5 SAX . . . . .	16
2.1.6 Andere mögliche Methoden . . . . .	16
2.2 XPath Language . . . . .	16
2.3 Graphen und das Property-Graph-Modell . . . . .	20
2.3.1 Das Property-Graph-Modell . . . . .	22
<b>3 Ein Tool zur Transformation von XML in das Property-Graph-Modell</b>	<b>24</b>
3.1 Das Konzept . . . . .	24
3.2 Anwendungsziele . . . . .	25
3.3 X2G als Transformationswerkzeug . . . . .	25
3.4 Interne Modulstruktur . . . . .	25
3.5 Übersicht der Modulstruktur . . . . .	26
3.6 Die 3 Baugruppen von X2G . . . . .	27
3.6.1 Filter . . . . .	27
3.6.2 Graph-Generator . . . . .	32
3.6.3 Node-Replacement Cases . . . . .	42
3.6.4 Graph-Exporter . . . . .	47
3.6.5 Graph-Szenarien Beispiele . . . . .	49
<b>4 Zusammenfassung und Schlussfolgerung</b>	<b>50</b>
<b>5 Literaturverzeichnis</b>	<b>54</b>

# **Abstract**

Ziel der Arbeit ist die Implementierung eines Tools zur Visualisierung von XML-Dokumenten durch Erstellung von Graphen. Das Tool ist in Java-Sprache codiert. Die Schwerpunkte der Arbeit sind die Untersuchung von XML-Daten, die Erstellung von Graph-Daten, deren Technik und Konzeption, sowie die Implementierung des Tools. Das Tool ermöglicht das Erstellen von Graphen basierend auf den Eingaben des Benutzers, dh der Benutzer kann entscheiden, welche Knoten und welche Kanten aus den XML-Dokumenten in Graphen dargestellt werden. Es beinhaltet auch Unterstützung für weitere XML Schemadarstellungen, da das Tool nicht schemaabhängig ist. Die resultierende Arbeit kann wieder in einer CSV-Datei gespeichert werden. Die Arbeit umfasst die Beschreibung der Werkzeugstruktur und die Beschreibung der wichtigsten Teile der Funktionalität (Filtern von XML-Dokumenten, Generieren der Knoten und Kanten und anschließendes Exportieren als CSV).

The aim of the work is to implement a tool for visualizing XML documents by creating graphs. The tool is coded in Java language. The main idea of the work is: study of XML, graph data and their techniques + conception and implementation of the tool. The tool allows creating graphs based on the user's input, that's mean the user can decide which nodes and which edges from the XML documents are represented in the graph. It also includes support for other XML schema representations as the tool is not schema dependent. The results of this work will be saved as a CSV file. The description of the tool structure and the description of the most important parts of the functionality (filtering XML documents, generating the nodes and edges and then exporting as CSV) are also included in this work .

## **Abkürzungsverzeichnis**

**XML** = Extensible Markup Language

**XPATH** = XML Path Language

**DTD** = Dokumenttyp Definition

**XSD** = XML Schema Definition

**XSL** = Extensible Stylesheet Language

**XSLT** = XSL Transformations

**CSV** = Comma-separated values

**HTML** = HyperText Markup Language

**DOM** = Document Object Model

**SAX** = Simple API for XML

**UML** = Unified Modeling Language

# Abbildungsverzeichnis

Abbildung 1: Beispiel für XML

Abbildung 2: XML fragment 1

Abbildung 3: XML fragment 2

Abbildung 4: Namenskonflikt gelöst

Abbildung 5: Beispiel-DTD

Abbildung 6: Beispiel für ein XSD

Abbildung 7: XML-DOM-Baum

Abbildung 8: Eine Beispiel-XSLT-Transformation, die einen Header im XHTML-Format

Abbildung 9: Beispiel für SAX-Events

Abbildung 10: XPath-Achsen-1 [36]

Abbildung 11: XPath-Achsen-2 [35]

Abbildung 12: Graph mit 2 Knoten

Abbildung 13: Graph mit 3 Knoten

Abbildung 14: Gephi[35]

Abbildung 15: Das Konzept

Abbildung 16: 3-Baugruppen

Abbildung 17: Filter

Abbildung 18-23: Beispiel 1-6

Abbildung 24: Filter - XML AS TREE

Abbildung 25: Filter-Selektion

Abbildung 26: Generierungsschritt 1: Erster Knoten des ersten XML-Dokuments

Abbildung 27: Generierungsschritt 2: Place Node

Abbildung 28: Generierungsschritt 3: Add property

Abbildung 29: Generierungsschritt 4: Type and Value

Abbildung 30: Generierungsschritt 5: Edge

Abbildung 31: Graph-Generierung: 1st Graph from 1st XML

Abbildung 32: Generierungsschritt 6: next Document

Abbildung 33: Graph-Generierung: Compare Properties-1

Abbildung 34: Graph-Generierung: Compare Properties-2

Abbildung 35: Graph-Generierung: Compare Properties-3, Graph is ready

Abbildungen 36-39: Duplikate-Erkennung Varianten

Abbildung 40: Exporter

Abbildung 41: Ausgabe .CSV

# 1 Einführung

Im Rahmen des ISEBEL-Projekts<sup>1</sup>, einer internationalen Suchmaschine, die Daten aus Märchendatenbanken sammelt, gibt es eine bekannte digitale Sammlung, nämlich Wossidia<sup>2</sup> (Von Richard Wossidlo aus Mecklenburg). Ein Teil des Projekts befasst sich mit dem Data- und Graph-Mining für häufige Muster, da die gesammelten Daten in XML-Storydaten gespeichert sind. Das WossiDiA [31] als Teil des ISEBEL-Projektes verwendet Hypergraphen zur Darstellung der Sammlungen von Richard Wossidlo. Um Forscher dabei zu unterstützen, bestimmte Aspekte der gesammelten Daten zu analysieren, zu betrachten und zu visualisieren, die nicht nur auf WossiDiA-Graphdaten beschränkt sind, sollten wir darauf abzielen, alle gesammelten XML-Daten in Property-Graphen umzuwandeln. Für den praktischen Einsatz muss ein gutes und vielseitiges Anwendungs-Modell zur Visualisierung von XML-Dokumenten einige besondere Anforderungen erfüllen:

- Es muss in der Lage sein, Sätze von XML-Dokumenten auszudrücken
- Die Transformation sollte von benutzerdefinierten Regeln geleitet werden. Darüber hinaus muss ein Werkzeug entwickelt werden, das die Property-Graphen, basierend auf den vom Benutzer eingegebenen Regeln, generiert.
- Außerdem muss es die Navigation mit XPath-Ausdrücken zulassen.

Im Prinzip wird die XML-Struktur nicht 1-1 in die Graph-Struktur überführt. Stattdessen werden neue Knoten und Kanten erzeugt, die es zuvor in dem XML-Dokument gar nicht gab. Oder es werden verschiedene XML-Inhalte zusammengeführt.  
Das Tool wird klassische Funktionen enthalten, die jedes Werkzeug haben sollte.

---

<sup>1</sup><https://search.isebel.eu>

<sup>2</sup>[www.wossidia.de](http://www.wossidia.de)

## 1.1 Motivation

Forscher\*innen, die mit großen Mengen von XML-Dateien arbeiten, haben Schwierigkeiten mit dem Durchsuchen sowie mit dem Analysieren der Dateien, wodurch die Effizienz der Arbeit stark eingeschränkt wird. X2G unterstützt die Forscher\*innen diese Schwierigkeit zu überwinden, indem alle gesammelten XML-Dokumente in Property Graph-Daten umgewandelt werden. X2G ist daher ein interessantes Projekt nicht nur aufgrund der Möglichkeiten, die es den Forschenden in der Bibliothek zur Datenanalyse zur Verfügung stellt, sondern auch wegen seiner ganzheitlichen Perspektive, da sein Hauptzweck darin besteht, eine allgemeine Methodik zur Verfügung zu stellen. Es ermöglicht dem Benutzer, Regeln zu definieren, die Elemente, Attribute und Inhalte aus XML-Dateien auswählen und aus dieser Auswahl Knoten, Labels, Kanten und Eigenschaften eines Graphen generieren.

## 1.2 Problemstellung

Wie bereits erwähnt, ist X2G als Konvertierungstool konzipiert. Daher muss es in der Lage sein, die Eingaben des Benutzers zu berücksichtigen, den Graphen so zu erstellen, wie es der Benutzer wünscht.

Gemeint ist hiermit:

- Welche Texte oder Kontexte aus den Dokumenten werden extrahiert?
- Und welche Knoten werden mit welchen verbunden ?
- Wann wird ein neuer Knoten erstellt, wann nicht?
- Und wenn ein neuer Knoten erstellt werden soll, was wird mit den Eigenschaften des neuen bzw. alten Knoten passieren?

Ein möglicher Weg, dieses Ziel zu erreichen, wird in dieser Bachelorarbeit vorgestellt, X2G ermöglicht es dem Benutzer mit Hilfe verschiedener Eingaben eine Graph-Erstellung zu ermöglichen. Graphen-Strukturen werden durch einen bestimmten Prozess erstellt, diesen Prozess werde ich in den nächsten Folien vorstellen.

### 1.3 Verwandte Arbeiten

Es gibt bereits einige Tools zum Umwandeln von XML-Dokumenten. , z. B. den yEd-Graph editor [5] Dieser Editor wandelt die XML-Dokumente in eine Visualdarstellung um. Allerdings dabei muss der Benutzer auch einiges manuell tun, und es wandelt die XML-Struktur genau in die Diagramm-struktur um.

Die XSLT-API<sup>3</sup> (Extensible Stylesheet Language Transformation) kann für eine Vielzahl von Zwecken verwendet werden. Mit einem ausreichend intelligenten Stylesheet kann man beispielsweise PDF-Dokumenten oder PostScript-Ausgaben usw. aus XML-Daten generieren. Typischerweise wird XSLT jedoch verwendet, um eine formatierte HTML-Ausgabe zu generieren oder um alternative XML-Darstellungen von Daten zu erstellen[4]. Es gibt außerdem Saxon, BaseX oder eXist-db XSLT 2/3- oder XQuery 3.1. [6] . Diese Transformationstechniken funktionieren für dieses Projekt nicht, da Daten nicht 1-zu-1 umgewandelt werden sollen, sondern der Benutzer die Umwandlung anhand einer bestimmten Regelsprache festlegt. So werden Knoten und Kanten erzeugt, die nicht direkt im XML enthalten waren oder die aus verschiedenen XML-Inhalten zusammengeführt wurden. Daher wurde im Rahmen dieser Bachelorarbeit das Tool X2G entworfen.

---

<sup>3</sup><https://docs.oracle.com/javase/tutorial/jaxp/xslt/index.html>

## 2 XML- und Graphdatenmodelle

### 2.1 XML-Allgemein

XML (Extensible Markup Language)[2] ist eine generische Auszeichnungssprache zum Erstellen von Auszeichnungssprachen für spezielle Zwecke, die viele verschiedene Datentypen beschreiben können. Mit anderen Worten, XML ist eine Möglichkeit, Daten zu beschreiben. Es ist extensible, da XML es ermöglicht, neue Markup-Elemente zu erstellen.

Abbildung [1] XML-Dokument Beispiel

Der Hauptzweck besteht darin, den Datenaustausch zwischen verschiedenen Systemen zu erleichtern, insbesondere solchen, die über das Internet verbunden sind. XML-basierte Sprachen (z. B. GML), RDF/XML, RSS, Atom, MathML, XHTML, SVG, XUL, ID, Clip und Music XML)[h8] sind formal definiert, sodass Programme Dokumente in diesen Sprachen ohne vorherige Änderung und Authentifizierung ändern und authentifizieren können Kenntnis ihres spezifischen Formats. XML-Dateien sind ganz gewöhnliche Dateien, die der Benutzer problemlos lesen und parsen kann. XML wird auch häufig als Format zum Speichern und Verarbeiten von Dokumenten verwendet, sowohl online als auch offline.

Mehr zu XML findet sich auf der Website des World Wide Web Consortium (W3C)[7].

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <lsebel>
3   <Tools>
4     <Tool name="X2G.Main">
5       <File Source="MainWindow.xaml" />
6       <File Source="MainWindow.xaml.cs" />
7     </Tool>
8     <Tool name="X2G.Model">
9       <File Source="Models/XmlModel.cs" />
10      <File Source="Models/ModelFactory.cs" />
11    </Tool>
12  </Tools>
13  <Quellen>
14    <Image Source="Images/icon.ico" />
15  </Quellen>
16  <RequiredNetVersion version="4.0" />
17  <Notes>
18    <Note>X2G's source code files.</Note>
19  </Notes>
20  <!-- Kommentar -->
21 </lsebel>
```

Abbildung 1: Beispiel für XML

XML Prolog ist die Komponente, die am Anfang eines XML-Dokuments hinzugefügt wird. XML-Dokument beginnt mit einem Prolog. XML Prolog enthält XML-DOCTYPE Declaration DTD[9] und Kommentare sowie Verarbeitungsanweisungen. DOCTYPE Declaration ist kein pflichtiger Teil. Mehr darüber in den folgenden Abschnitten.

Die Tags Ein Tag ist eine Auszeichnung eines Datenbestandes mit zusätzlichen Informationen

- <Isebel> Ein Beispiel: Starttag für den Beginn eines Elementes
- </Isebel> Ein Beispiel: Endtag für das Ende eines Elementes
- <Leer/> ein Leertag für ein Element ohne Inhalt Wenn sich zwischen diesen beiden Tags kein Text oder keine Tags befinden, kann man das End-Tag überspringen und das gesamte Tag als leeres Element ausdrücken.

Element:

Die Buchstaben(Text) zwischen dem ersten und dem letzten Tag sind der Inhalt des Elements.

Beispiel: <Isebel> Story </Isebel>

Die Elemente in XML werden mit ihren eigenen Namen definiert. In der Abbildung 1 gibt es beispielweise das Element: Isebel, Isebel ist der Name des Elements. Es kommt manchmal vor, dass manche Elemente Attribute haben. In XML gibt es keine Regeln darüber, wann Attribute und wann untergeordnete Elemente verwendet werden. Zum Beispiel:

```
<personen gender=„male“> <vname>Mike</vname> <lname>Witt</lname> </personen>
```

Und

```
<personen> <gender>male</gender> <vname>Mike</vname> <lname>Witt</lname> </personen>
```

gender wird als attribut im ersten Beispiel dargestellt, aber im Zweiten ist gender ein untergeordnetes Element. ohnehin wird in beiden Fällen die gleichen Informationen geliefert. In den XML-Dokumenten kann man auch auch Kommentare schreiben, die zwischen <!-- Kommentar --> eingefügt werden müssen. Dieses Beispiel-XML-Dokument ist wohlgeformt.

Textobjekte sind jedoch wohlgeformte XML-Dokumente wenn die die definition von XML-spezifikation entsprechen. Mehr darüber[2].

Es besagt, dass Elemente korrekt ineinander verschachtelt sein müssen[38].  
Es ist sehr wichtig auf die Klein- und Großschreibung Acht zu geben in Tags. Daher müssen Starttag und Endtag eines Elmentes kompelt gleich sein.

Für jedes offene Element muss ein korrektes schließendes Element vorhanden sein  
Verschachtelte Tags müssen in der richtigen Reihenfolge geschlossen werden.  
Groß- und Kleinschreibung beachten [case sensitive]. <Isebel> und <isebel> sind verschiedene Elemente. [38]

### 2.1.1 Namespace in XML-Dokumenten

In der XML gibt es den Begriff Namensraum (Namespace auf englisch) Ein Namespace ist ein URI (Uniform Resource Identifier)[42], die verwendet wird, um eindeutig benannte Elemente und Attribute in einem XML-Dokument bereitzustellen. Die Hauptaufgabe der Namespace ist zu ermöglichen, dass ein und dasselbe Dokument XML-Elemente und -Attribute aus verschiedenen Vokabularen enthält, ohne dass es zu Namenskollisionen kommt[43]. Namespace wurzelt in XML. Beim Erstellen eines Dokuments werden Kombinationen aus Vokabularelementen und Attributen erstellt. XML kann Element- oder Attributnamen aus mehreren XML-Vokabularen enthalten. Indem jedem Vokabular ein Namensraum zugewiesen wird, können Mehrdeutigkeiten zwischen Elementen oder Attributen mit demselben Namen aufgelöst werden[42]. Namespace bietet sich da als Lösung für das Unterscheiden der unterschiedlichen Elementen. Für Namensräume wird das Attribut xmlns (für engl.: XML Namespace) verwendet. [8] Die XML-Fragmente in Abbildung 2 und Abbildung 3 Berücksichtigen.

```
1 <Isebel>
2   <F>Isebel Wossidia</F>
3   <F>Story</F>
4 </Isebel>
```

Abbildung 2: XML fragment 1

```
1 <Isebel>
2   <Name>Isebel Wossidia</Name>
3   <Type>Story</Type>
4 </Isebel>
```

Abbildung 3: XML fragment 2

Die XML-Teile aus den Abbildungen 2 und 3 können nicht zusammen in einem Dokument kommen. Wenn doch dann es wird zu schwierigkeiten kommen. Der Grund dafür ist dass, die Elemente die gleichen Namen haben.

Um dieses Hindernis zu überwinden, müssen wir die XML-Teile in unverbundenen Namenscontainer stellen. Abbildung 4 berücksichtigen.

Erstes Fragment gehört zum Namespace, der durch (URI) [5] <http://www.isebel.eu/> beschrieben wird.

Wie wir sehen können, ist der verwendete Name dafür viel zu lang, es gibt eine Lösung, und zwar ihn mit einem Buchstaben zu identifizieren.

Daher werden wir für den Namespace <http://www.isebel.eu/> das Präfix m verwenden. Das Präfix muss vor dem Namen der Elemente verwendet werden, die zu diesem Namespace gehören.

```

1 <lsebels xmlns : m="http://www.isebel.eu/"
2     xmlns : n="http://www.wossidia.de/">
3     <m:Isebel>
4         <m:F>Isebel Wossidia</m:F>
5         <m:F>Story</m:F>
6     </m:Isebel>
7     <n:Isebel>
8         <n:Name>Isebel Wossidia</n:Name>
9         <n>Type>Story</n>Type>
10    </n:Isebel>
11 </lsebels>
```

Abbildung 4: Namenskonflikt gelöst

### 2.1.2 Schema von XML-Dokumenten

Das XML-Schema beschreibt die Struktur eines XML-Dokuments. Die Sprache XML Schema wird auch als XML Schema Definition (XSD) bezeichnet[10]. XML-Schemas drücken gemeinsame Vokabulare aus und ermöglichen es Maschinen, von Menschen erstellte Regeln auszuführen.

XML-Dokumente haben sehr unterschiedliche Strukturen, wodurch Informationen auf vielfältige Weise dargestellt werden können. Für uns Menschen stellt dies kein Hindernis dar, aber es könnte während der Kommunikation zu Schwierigkeiten zwischen Maschinen führen. Für sowas gibt es XML-Schema. Der Hauptzweck von XML Schema besteht darin, einige Regeln für das Dokument zu bestimmen[45]:

- Welche Elemente und Attribute in einem Dokument zulässig sind
- die Anzahl (und Reihenfolge) der untergeordneten Elemente
- Datentypen für Elemente und Attribute
- Standard- und Festwerte für Elemente und Attribute

Dadurch können beide Seiten(z.B Computern) erfahren, was auf sie zukommen wird und können dafür bereit sein.

```

1 <!DOCTYPE Isebel [
2 <!ELEMENT Isebel (Tools, Quellen, RequiredNetVersion, Notes?)>
3 <!ELEMENT Tools (Tool+)>
4 <!ELEMENT Quellen (Image*)>
5 <!ELEMENT RequiredNetVersion EMPTY>
6 <!ATTLIST RequiredNetVersion version CDATA #REQUIRED>
7 <!ELEMENT Notes (Note*)>
8 <!ELEMENT Note (#PCDATA)>
9 <!ELEMENT Tool (File+)>
10 <!ATTLIST Tool name CDATA #REQUIRED>
11 <!ELEMENT File EMPTY>
12 <!ATTLIST File Source CDATA #REQUIRED>
13 <!ELEMENT Image EMPTY>
14 <!ATTLIST Image Source CDATA #REQUIRED>
15 ]>
```

Abbildung 5: Beispiel-DTD

DTD-Dateien enthalten Regeln, die zum Deklarieren bestimmter Arten von XML-Dokumenten verwendet werden müssen. Sie beinhaltet u.a. den Elementtyp und die Attribute des Elements. Eine DTD definiert die Struktur und die zulässigen Elemente und Attribute eines XML-Dokuments[14]. Die obige DTD[14] wird wie folgt interpretiert:

- !DOCTYPE Isebel definiert, dass das Wurzelement dieses Dokuments Isebel ist
- !ELEMENT Isebel definiert, dass das Isebel -Element vier Elemente enthalten muss.
- !ELEMENT Note definiert das Note -Element als Typ "#PCDATA"
- !ATTLIST Tool definiert Attribute

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:element name="Isebel">
4     <xs:complexType>
5       <xs:choice minOccurs="0" maxOccurs="unbounded">
6         <xs:element name="Tools">
7           <xs:complexType>
8             <xs:sequence>
9               <xs:element name="Tool" minOccurs="0" maxOccurs="unbounded">
10              <xs:complexType>
11                <xs:sequence>
12                  <xs:element name="File" minOccurs="0" maxOccurs="unbounded">
13                    <xs:complexType>
14                      <xs:attribute name="Source" type="xs:string" />
15                    </xs:complexType>
16                  </xs:element>
17                </xs:sequence>
18                  <xs:attribute name="name" type="xs:string" />
19                </xs:complexType>
20              </xs:element>
21            </xs:sequence>
22          </xs:complexType>
23        </xs:element>
24        <xs:element name="Quellen">
25          <xs:complexType>
26            <xs:sequence>
27              <xs:element name="Image" minOccurs="0" maxOccurs="unbounded">
28                <xs:complexType>
29                  <xs:attribute name="Source" type="xs:string" />
30                </xs:complexType>
31              </xs:element>
32            </xs:sequence>
33          </xs:complexType>
34        </xs:element>
35        <xs:element name="RequiredNetVersion">
36          <xs:complexType>
37            <xs:attribute name="version" type="xs:string" />
38          </xs:complexType>
39        </xs:element>
40        <xs:element name="Notes">
41          <xs:complexType>
42            <xs:sequence>
43              <xs:element name="Note" type="xs:string" minOccurs="0" />
44            </xs:sequence>
45          </xs:complexType>
46        </xs:element>
47      </xs:choice>
48    </xs:complexType>
49  </xs:element>
50 </xs:schema>
```

Abbildung 6: Beispiel für ein XSD

### 2.1.3 DOM-Dokument Objekten Modell

Dokumentobjektmodell (Dokument objekt modell)[15] Das Document Object Model ist eine plattformübergreifende Programmierschnittstelle API, die HTML, XML und andere Auszeichnungssprachen in Form einer Baumstruktur verarbeitet. Im Dokumentobjektmodell wird jedes Tag/Element durch einen einzelnen Knoten dargestellt.

- DOM kann Markup-Dokumente (html xml) in Objekte umwandeln
- Kann jede Markierung eines Markup-Dokuments in ein Objekt umwandeln
- Nachdem das Dokument oder Tag in ein Objekt eingekapselt wurde, kann das Objekt mehr Attribute und Möglichkeiten zum Manipulieren des Dokuments haben

Im Dom-Strukturdiagramm sehen wir, dass das Schema aus Elementknoten besteht. Jedes XML-Element ist ein Knoten. Knoten können unterteilt werden in: Textknoten, Elementknoten, Attributknoten. Die Dokument Objekten Modell ist für mehrere Sprachen verfügbar. Abbildung 7 zeigt ein Beispiel der DOM-Baum für das Beispiel in Abbildung 1.

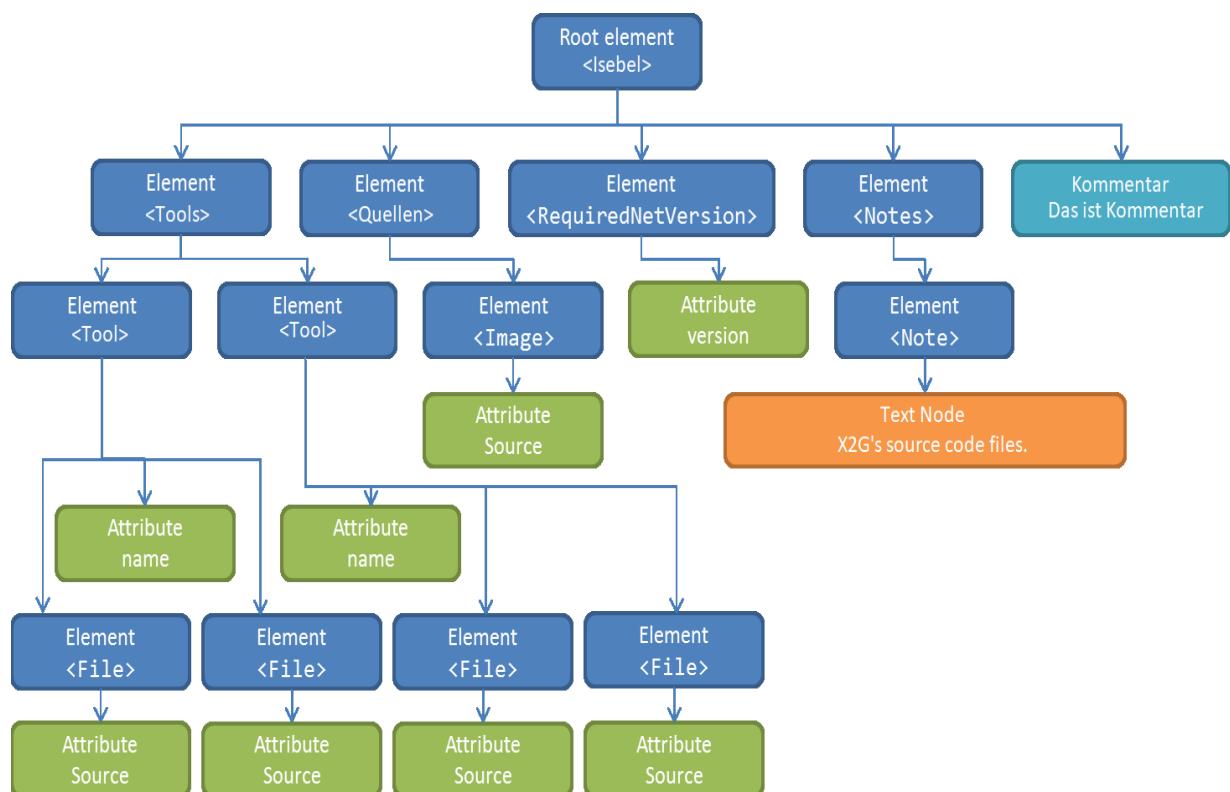


Abbildung 7: XML-DOM-Baum

#### **2.1.4 XML und XSLT-Transformationstechniken**

Bei der Entwicklung der XML- Sprache [2] hatten die W3C – Entwickler klare Absichten.: Sie wollten eine Sprache erstellen, die gleichzeitig formal, prägnant, für Anwendungen und Machine leicht zu verarbeiten und für Menschen les- und schreibbar ist. Aktuell wird XML in praktisch jedem IT-Feld als die natürlichste Form zum Speichern und Verarbeiten von Dokumenten verwendet, sowohl online als auch offline, und bietet mehrere Vorteile:[32]

- Eine solide, logisch beweisbare Formel basierend auf internationalen Standards.
- Die Hierarchie ist für die meisten (aber nicht alle) Arten von Dokumenten geeignet.
- Es erscheint als reine Textdatei, unbelastet von Lizenzen oder Beschränkungen.
- Da es unabhängig ist, ist es daher relativ immun gegen technologische Veränderungen.

XML Sprache ist heutzutage weit verbreitet, daher sind IT-fachleute mit XML-Parsing und -Transformation beschäftigt, bei der Transformation von XML-Dokumenten. Für diesen Zweck hat W3C die XSLT-Sprache zum Transformieren von XML-Dokumenten vorschlagen. XSLT steht für (Abkürzung von Extensible Stylesheet Language Transformations [19]). Das Hauptziel von XSLT ist die Konvertierung von XML-Dokumenten in ein anderes Format wie PDF oder HTML, um es als Webseite anzuzeigen. XSLT verwendet häufig Xpath, um auf bestimmte Teile eines XML-Dokuments zuzugreifen.

### 2.1.5 SAX

Simple API for XML (SAX) [16] ist eine ereignisbasierte API.

SAX, der vollständige Name lautet Simple API für XML, ein Programmierschnittstelle- und Softwarepaket, das auch eine Alternative zum XML-Parsing darstellt. Sax unterscheidet sich vom Dom-Parsing, es scannt das Dokument Zeile für Zeile und parst es während des Scannens. Das Programm überprüft die Daten nur beim Lesen, da die Daten nicht im Arbeitsspeicher abgelegt werden müssen, was bei der Analyse großer Dokumente ein großer Vorteil ist im Gegensatz zu DOM, der nur für kleinere Dokumente geeignet (ca. 0B bis 10MB). Im Allgemeinen ist SAX viel schneller als DOM.

XML-Verarbeitung mit SAX[44]

Ein XML Parser scannt das Dokument Zeile für Zeile und währenddessen Ereignisse werden auftreten wenn wichtige Elemente auftreten (Textknoten, Element start und end Tags, Kommentare etc.)

Abbildung 9 zeigt einige der vom SAX-Parser beim Analysieren der in Abbildung 1 gezeigten Beispieldatei ausgelösten Ereignisse.

```
1 1) startDocument() - beginning of the document
2 2) startElement() Isobel - beginning of the element
3 3) startElement() Tools
4 4) startElement() Tool - also provides attributes
5 5) ...
6 6) endElement() Tool
7 7) startElement() Tool
8 8) ...
9 9) endDocument() - end of parsing the document
```

Abbildung 8: Beispiel für SAX-Events

### 2.1.6 Andere mögliche Methoden

Um auf XML-Dateien zuzugreifen, sind sowohl DOM als auch SAX die bekanntesten APIs.

Die anderen Methoden sind nur von denen erweitert. Der Hauptunterschied zwischen ihnen liegt in der Art und Weise, wie sie implementiert werden, oder in ihrem programmatischen Zweck

## 2.2 XPath Language

XML Path Language (XPath) [13]. Ist eine XML-Erweiterungen. Sie wird verwendet zum Auswählen oder Hervorheben von bestimmten Elementen, Texten, Attributen etc. von XML-Dokumenten. Es kann in vielen Situationen verwendet werden. Das heute wohl

bekannteste ist XPointer, das wiederum von XLink und XInclude verwendet wird, um auf Fragmente anderer Dokumente zu verweisen. XPath ist ein Primitiv, welches es ermöglicht, in ein Dokument zu zeigen, nicht nur in das gesamte Dokument. XPath ist auf die Verarbeitung von Dokumenten beschränkt, die mit dem XML-Namespace kompatibel sind, und funktioniert natürlich auch unter dem XPath-Datenmodell. Das Herzstück eines XPath-Ausdrucks ist der Standortpfad [22], der zum Auswählen von Knoten in XML-Dokumenten verwendet wird.

XPath verwendet Pfadausdrücke, um Knoten in einem XML-Dokument auszuwählen. Es werden Knoten ausgewählt, indem man Pfade oder Schritte folgt.

Die wichtigsten Pfadausdrücke sind unten aufgeführt [32] :

- '/' = alle Elemente ab dem Wurzelement werden selektiert
- '//' = alle Knoten unter dem aktuellen aktuellen Knoten, die mit dem Xpath-Ausdruck übereinstimmen werden selektiert, abgesehen davon wo sie sich befinden
- '.' = Aktueller Knoten, der mit der Auswahl übereinstimmt und selektiert
- '..' = übergeordneten Knoten des aktuellen Knoten wird selektiert
- '@' = Attribute werden mit diesem Ausdruck selektiert

XPath-Platzhalter können verwendet werden, um unbekannte XML-Knoten auszuwählen [32] :

- '\*' = Mit diesem Ausdruck kann jeder beliebige Element-Knoten ausgewählt werden
- '@\*' = Mit diesem Ausdruck kann jedes beliebige Attribut ausgewählt werden
- 'node()' = Mit diesem Ausdruck kann jeder beliebige Element-Knoten(jeglicher Art) ausgewählt werden

Die Achse stellt die Beziehung zum (aktuellen) Kontextknoten dar und wird verwendet, um den Knoten relativ zu diesem Knoten im Baum zu positionieren [33] :

- 'ancestor'(Vorfahren) = Mit diesem Ausdruck werden alle übergeordneten Knoten des Aktuellen Knoten selektiert
- 'attribute' = Mit diesem Ausdruck kann alle Attribute selektiert werden
- 'child'(Nachfolger) = Mit diesem Ausdruck werden alle untergeordneten Knoten des Aktuellen Knoten selektiert
- 'following'(alle folgenden) = Mit diesem Ausdruck wird alles im XML-Dokument nach dem Endtag des aktuellen Knoten selektiert
- 'namespace'(Namensraum) = Mit diesem Ausdruck werden alle Namensraum-knoten des Aktuellen Knoten selektiert

Für einen besseren Überblick über XPath-Achsen siehe Abbildung 9 und 10  
Das Ergebnis eines XPath-Ausdrucks kann Text oder Attribute oder ein boolescher Wert sein

Wenn der Leser Interesse hat Mehr über Xpath zu lesen [17].

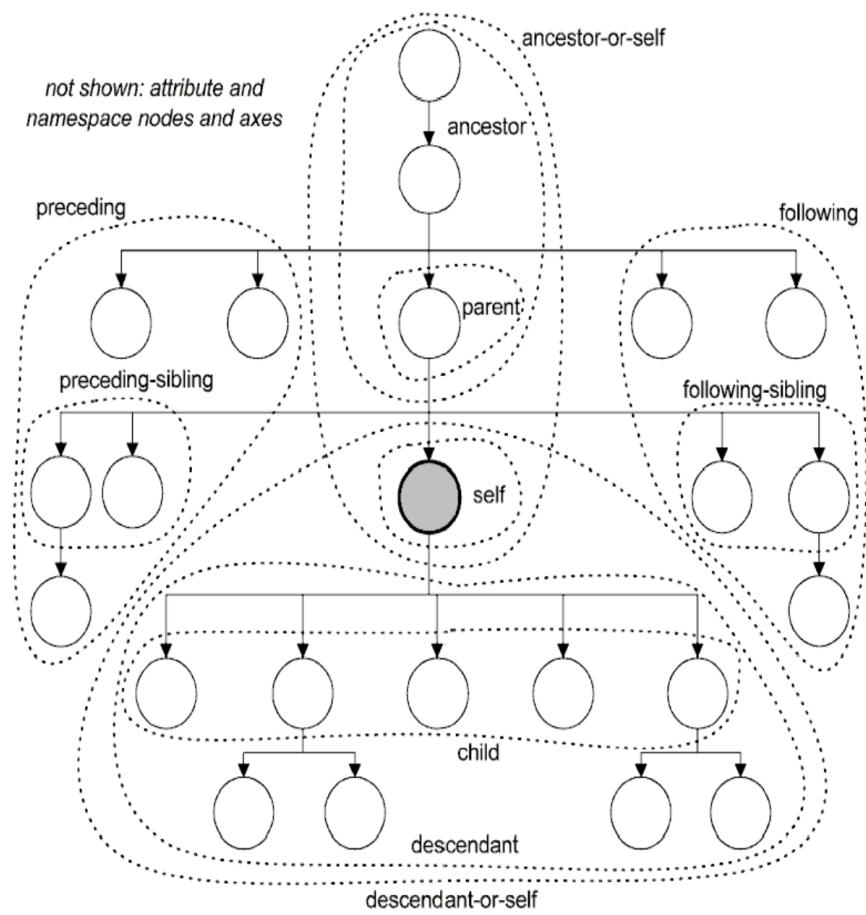


Abbildung 9: XPath-Achsen-1 [36]

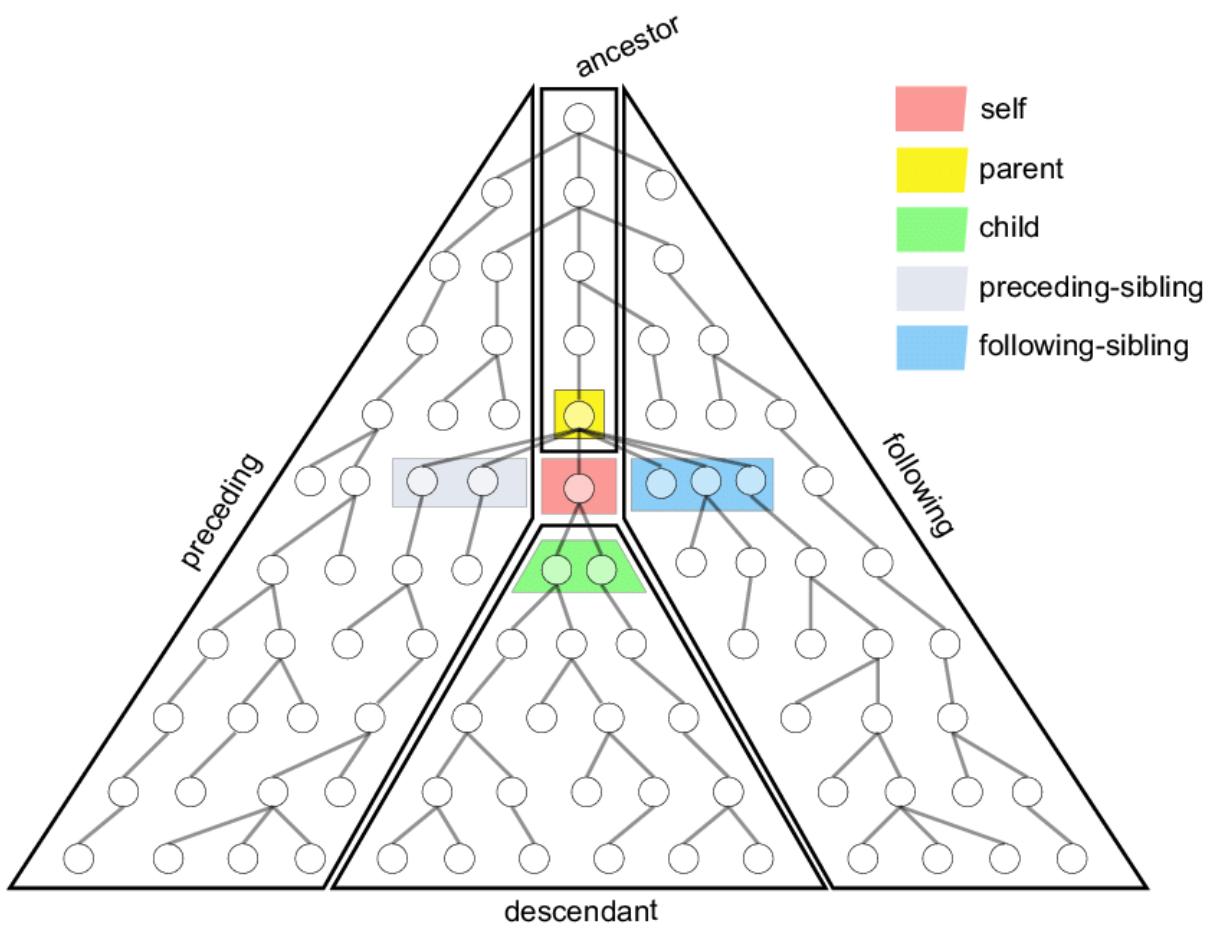


Abbildung 10: XPath-Achsen-2 [35]

## 2.3 Graphen und das Property-Graph-Modell

Formal ist ein Graph nur eine Sammlung von Knoten und Kanten oder eine generische Art, eine beliebige Datenstruktur darzustellen. Tatsächlich können wir als Graph fast jede Situation darstellen, die aus 2 Objekten und einer Beziehung zwischen ihnen besteht. Beispielsweise, aber nicht beschränkt auf, kann die Person und seiner Besitz als Graph dargestellt werden, wie wir in Abbildung 12 sehen können. Abbildung 12 stellt einen Fall dar, wo eine Person Schuhe besitzt. Die Beziehung zwischen der Person und Schuhen stellt einen Graphen dar.



Abbildung 11: Graph mit 2 Knoten

In der Abbildung 12 wird die (Kante)Verbindung BESITZT verwendet, aber es könnte mehrere Verbindungen (Beziehungen) zwischen zwei Objekten (Knoten) geben, wie z. B. trägt, reinigt, bindet usw. In ähnlicher Weise können mehrere verbundene Graphen mehrere Objekte haben, wie in Abbildung 13 gezeigt, die einen Graphen mit 3 Objekten darstellt.

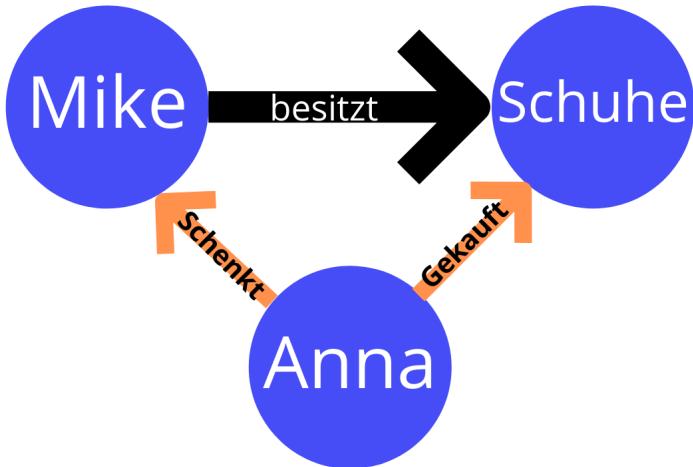


Abbildung 12: Graph mit 3 Knoten

Abbildung 13 stellt als Graph eine Situation dar, in der Anna Schuhe gekauft hat, sie Mike weitergegebenen bzw. geschenkt hat und Mike sie nun besitzt. Dies stellt die Objekte Mike, Anna und Schuhe und die (Kanten)Beziehungen, die ihnen zusammenbringt, als Graph dar. Die Möglichkeiten für Graphen zwischen Objekten sind sehr zahlreich.

Fast jede Situation kann als Graph dargestellt werden. Es kommt nur darauf an, was man für Bedürfnisse hat, und ob das sinnvoll und machbar ist. Die XML-Dokumenten lassen sich sehr gut mit Graphen visualisieren. Die Transformation von XML-Dokumenten in Graphdaten sieht oft wie ein DOM-Baum aus und kann auch um Kanten erweitert werden, die Verweise auf andere Identitäten darstellen. Diese Typinformationen können durch einen Typ-Graphen dargestellt werden, wenn das XML-Dokument einer bestimmten DTD oder einem bestimmten XML-Schema entspricht. Ein XML-Schema oder DTD wird in einen Typ-Graphen umgeformt. Wie bei der objektorientierten Modellierung können Typen über Vererbungsbeziehungen konstruiert werden [29]. Ein Beispiel für einen Typ-Graphen ist ein Strukturdiagramm. Propertygraphen beschreiben formal die Strukturgraphen. Normalerweise deklarieren wir die Property als Variablen, wenn wir uns mit der Programmiersprache befassen: Das heißt, für jede property wird ein Name und ein bestimmter Typ angegeben. Die property kann dann ein beliebiger Wert des angegebenen Typs zugewiesen werden.

### 2.3.1 Das Property-Graph-Modell

Property graph models sind Modelle mit Eigenschaften für Knoten und Kanten[34].

Property graph model besteht aus:

- Knoten und Kanten, bzw. Knoten und Beziehungen genannt
- Knoten bestehen aus (mindestens einer) Eigenschaft, die Attribute speichert
- Gelabelte und gerichtete Kanten, die Source- und TargetNodes umfasst
- Knoten ohne Kanten sind möglich, aber kommt nicht so oft vor
- Kanten können Eigenschaften haben, aber dies ist nicht wie bei Knoten erforderlich.

Abbildung 36 stellt Property-graph dar.

Um die Ergebnisse unseres Tools visualisieren zu können, benötigen wir ein Visualisierungs- oder Graphkprogramm Ein Beispiel für eine Graphkprogramm wäre Gephi, da diese Software gut zu unserem Projekt passt. Abbildung 14

Gephi übernimmt die exportierten .CSV-Dateien und stellt sie in Graph dar. Mehr über .CSV-Dateien kommt in den nächsten Folien.



Abbildung 13: Gephi[35]

### 3 Ein Tool zur Transformation von XML in das Property-Graph-Modell

#### 3.1 Das Konzept

In Abbildung 15 ist der grobe Ablauf des Tools als Ablaufdiagramm dargestellt.

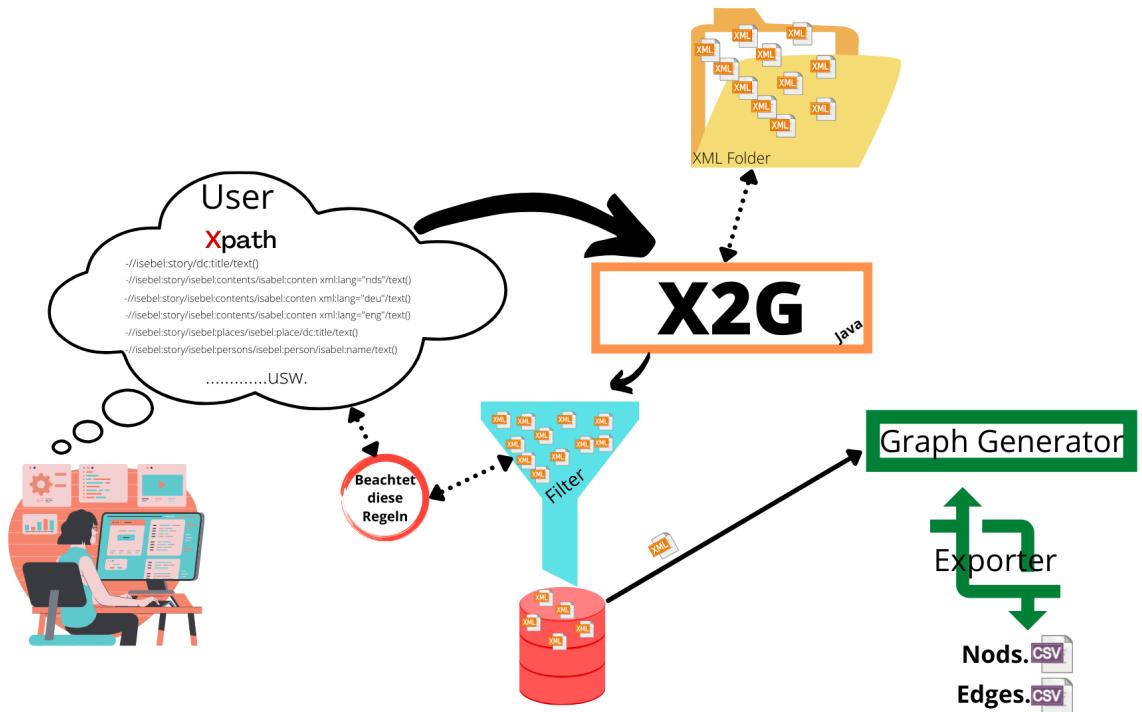


Abbildung 14: Das Konzept

Mit X2G werden XML-Daten verarbeitet und in Graph-Daten umgewandelt, um eine spätere Visualisierung zu ermöglichen!

Der Benutzer bereitet die Eingabe als Xpath vor. Diese Eingabe bestimmt, welche XML-Dokumente verwendet werden, um die Knoten und Kanten den Graphen zu generieren. Dann übermittelt es diese Eingabe und den Pfad der XML-Dateien an das Tool. Das Tool beginnt dann mit dem Filtern der XML-Dateien und speichert die gefilterten XML-Dateien in Dateiarrays. Nun fährt das Tool mit dem nächsten Schritt fort, der darin besteht, Knoten und Kanten aus den gefilterten XML-Dateien zu generieren. Der letzte Schritt ist der Exporter. Die Aufgabe des Graph-Exporters besteht darin, die erstellten Elemente (Knoten, Kanten, Labels und Eigenschaften) von Graphstrukturen in Dateien in einem geeigneten Format zu speichern.

Das Konzept der einzelnen Module

- Filterung der XML-Dateien
- Erzeugen von Knoten und Kanten
- Exportieren der Graph-Struktur in einem geeigneten Format

wird in den folgenden Folien detailliert aufgezeigt und erklärt.

## **3.2 Anwendungsziele**

Das als Ziel dieser Arbeit erstellte Tool heißt X2G abgekürzt von XML to Graph. Von nun an verwenden wir bei Bezugnahme auf dieses Tool den Namen X2G. Hauptziele der X2G-Tool sind die folgenden Punkte:

- Filtern der XML-Dokumente
- Auswahl bestimmter Elemente.
- Umgang mit XML-Schema
- Arbeiten mit großen Dokumentenmengen.
- Arbeiten mit großen Dokumenten
- Visualisierung von XML-Dokumenten:
  - Die Knoten des Graphen werden in der Node.csv-Datei gespeichert
  - Die Kanten des Graphen werden in der Node.csv-Datei gespeichert

und dann werden sie(.CSV Datein) an ein Graph-programm weitergegeben, z.B. Gephi

Als letztes Ergebniss haben wir einen Graph, der aus mehreren XML-Dokumenten erstellt wurde.

## **3.3 X2G als Transformationswerkzeug**

X2G dient der Aufbereitung von XML-Daten und deren Umformung in Graphdaten, um eine anschliessende Visualisierung zu ermöglichen. Visualisierung zu ermöglichen bedeutet, dass der Benutzer nicht gezwungen ist, den Quellcode des Dokuments zu bearbeiten, sondern eine visuelle Darstellung der XML-Dokumenten basierend auf seiner Eingabe erhält. Jedes Element eines Dokuments, sei es ein Element oder ein Text, kann mit einer eigenen grafischen Entität(Node) dargestellt werden und diese ist nicht mit der Struktur des Dokuments verbunden/abhängig. d.h. der nutzer wird in der Lage sein, die Graph-struktur beliebt bestimmen. Also wird im Prinzip die XML -Struktur nicht 1-zu1 in die Graphenstruktur überführt. Es können stattdessen neue Knoten und Kanten erzeugt werden, die es zuvor im XML- Dokument gar nicht gab oder die aus verschiedenen XML Inhalten zusammengeführt werden.

## **3.4 Interne Modulstruktur**

Die Struktur von X2G basiert auf einigen Schritten. Dies bedeutet, dass das gesamte Tool in eine Reihe von Schritte unterteilt ist, wobei jedes Schritt eine gekapselte Reihe von Funktionen bietet. Dieser Ansatz trägt dazu bei, dass das gesamte Tool während seiner Lebensdauer besserbar und editierbar bleibt und für andere Programmierer leichter verständlich ist. Schreibt man kleine Anwendungen mit dieser Technik, so verlängert sich gefühlt die Entwicklungszeit durch die Komplexität der Anwendung. Auf längere Sicht gesehen erfordern Änderungen allerdings weniger Zeit und Mühe, da die Anwendungen erweiterbar sind.

Bei der Entscheidung, wie X2G gestaltet werden soll, haben wir diese Fragen berücksichtigt und versucht, das Beste daraus zu machen.

### 3.5 Übersicht der Modulstruktur

X2G ist in Java geschrieben Das Tool besteht aus drei Hauptteilen (Baugruppen) wie in Abbildung (3- Baugruppen) gezeigt wird. Jeder Teil hat seine Funktionalität, um das Ziel dieser Arbeit zu erreichen.

Die Funktionalität der einzelnen Teile wird auf den nächsten Folien näher erläutert.

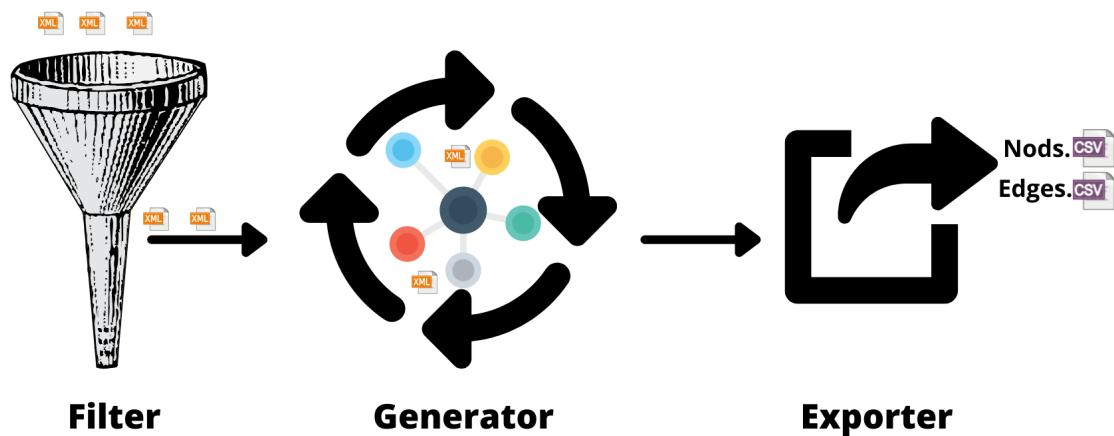


Abbildung 15: 3-Baugruppen

## 3.6 Die 3 Baugruppen von X2G

### 3.6.1 Filter

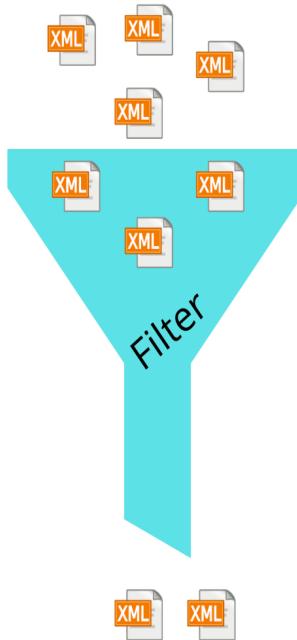


Abbildung 16: Filter

Die Hauptaufgabe des Filters liegt in seiner Funktionalität, eine Reihe von XML-Dokumenten auszuwählen, die den vom Benutzer festgelegten Regeln entsprechen. Hier stellt sich oft die Frage, warum ist es notwendig, einen Filter zu erstellen? Um diese Frage zu beantworten, müssen wir zunächst wissen, dass die XML-Dateien, auf denen wir die X2G-Tool ausführen, hier meinen wir die (Isebel-XML-Dateien), nicht alle vollständig sind. Bei einigen Dokumenten fehlen wichtige Teile der Story-Struktur. Diese Teile kann der Benutzer möglicherweise benötigen, um den gewünschten Graphen zu erstellen. Solche XML-Dokumente werden beim Generieren von Graphen nicht verwendet. Und dafür ist der Filter da, um ein bestimmtes Ziel-XML-Dokument auszuwählen. Dies wird auf der folgenden Abbildung noch genauer verdeutlicht.

```

▼<isebel:story xmlns:isebel="http://www.isebel.eu/ns/isebel" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:html="http://www.w3.org/1999/xhtml" xml:id="xmd-s001-000-000-030" xml:lang="deu">
  <dc:identifier>de.wossidia.xmd-s001-000-000-030</dc:identifier>
  <isebel:purl>https://apps.wossidia.de/webapp/run/node/XMD-S001-000-000-030</isebel:purl>
  <dc:title>[30] Der gefundene Schatz macht standesgemäß</dc:title>
  <dc:title xml:lang="nds"/>
  <dc:title xml:lang="deu"/>
  <dc:title xml:lang="eng"/>
  </isebel:contents>
  <isebel:content xml:lang="nds">Kleekamp willen poor Daglöhner Holt stietzen. 'Dat' is noch tiegenden Morgen. 'Se kamen an 'ne! Kuul mit Steen 'vörbil. Een mööt ut de Hosen. 'He' will 'solang' Dorvon het de Herr en Stuv bugen laten in't Herrenhaus. Erzähler: H. Müller, Hohen Viechein, Kreis Wismar; Aufzeichner: Wossidlo, 21. 5. 1932.</isebel:content>
  <isebel:content xml:lang="deu">In Kleekamp wollen ein paar Tagelöhner Holz stehlen. Es ist noch sehr zeitig am Morgen. Sie kommen an einer Steingrube vorbei. Einer von ihnen muß mal in die almodisch gewesen. Er hat es aber genommen und umgewechselt. Davon hat ihm der Herr im Herrenhaus eine Stube bauen lassen.</isebel:content>
  <isebel:content xml:lang="eng">A few daytaller wanted to steal some wood. It was very early in the morning. They passed a stone pit. One of them had to go in the bushes. Whilst he wanted to From that profit, the squire had a room in the manor built for him.</isebel:content>
</isebel:contents>
  <isebel:places>
    <isebel:place id="100001174">
      <dc:title>Hohen Viechein</dc:title>
      <isebel:point>
        <datacite:pointLatitude xmlns:dcatac="http://datacite.org/schema/kernel-4">53.78404</datacite:pointLatitude>
        <datacite:pointLongitude xmlns:dcatac="http://datacite.org/schema/kernel-4">11.51169</datacite:pointLongitude>
      </isebel:point>
      <isebel:role>narration</isebel:role>
    </isebel:place>
  </isebel:places>
  <isebel:events>
    <isebel:event id="990013673">
      <isebel:date>1932-05-21</isebel:date>
      <isebel:role>narration</isebel:role>
    </isebel:event>
  </isebel:events>
  <isebel:keywords>
    <isebel:keyword id="XMD-M030-000-000-640" type="controlled vocabulary" provenance="Wossidla-Index">Schätze</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-004-718" type="controlled vocabulary" provenance="Wossidla-Index">finden</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-008-000" type="controlled vocabulary" provenance="Wossidla-Index">nehmen (untersch. Bed.)</isebel:keyword>
    <isebel:keyword xml:lang="deu" id="ZAM-B802" type="controlled vocabulary" provenance="RW-Index">Schätze</isebel:keyword>
    <isebel:keyword xml:lang="eng" id="ZAM-B802" type="controlled vocabulary" provenance="RW-Index">treasure</isebel:keyword>
    <isebel:keyword xml:lang="deu" id="ZAM-B802-017" type="controlled vocabulary" provenance="RW-Index">Den gefundenen Schatz nimmt ein anderer</isebel:keyword>
    <isebel:keyword xml:lang="eng" id="ZAM-B802-017" type="controlled vocabulary" provenance="RW-Index">The found treasure takes another</isebel:keyword>
  </isebel:keywords>
  <isebel:imageResources>
    <isebel:imageResource id="ZAM-B802-017-009-001">
      <isebel:purl>https://apps.wossidia.de/digipool/3bab7eb/master</isebel:purl>
    </isebel:imageResource>
    <isebel:imageResource id="ZAM-B802-017-009-002">
      <isebel:purl>https://apps.wossidia.de/digipool/3bab7ec/master</isebel:purl>
    </isebel:imageResource>
  </isebel:imageResources>
</isebel:story>

```

Abbildung 17: Beispiel 1

Ausschnitt aus einem ISEBEL-Story-Dokument mit keyword-Elementen.

```

▼<isebel:story xmlns:isebel="http://www.isebel.eu/ns/isebel" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:html="http://www.w3.org/1999/xhtml" xml:id="xmd-s001-000-000-218" xml:lang="deu">
  <dc:identifier>de.wossidia.xmd-s001-000-000-218</dc:identifier>
  <isebel:purl>https://apps.wossidia.de/webapp/run/node/XMD-S001-000-000-218</isebel:purl>
  <dc:title>[218] Bestrafte Habgier</dc:title>
  <isebel:contents/>
</isebel:story>

```

Abbildung 18: Beispiel 2

Story-Dokument ohne Inhalt und nur Titelangabe.

```

▼<isebel:story xmlns:isebel="http://www.isebel.eu/ns/isebel" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:html="http://www.w3.org/1999/xhtml" xml:id="xmd-s001-000-000-081" xml:lang="deu">
  <dc:identifier>de.wossidia.xmd-s001-000-000-081</dc:identifier>
  <isebel:purl>https://apps.wossidia.de/webapp/run/node/XMD-S001-000-000-081</isebel:purl>
  <dc:title>[81] Mord</dc:title>
  <dc:title xml:lang="deu"/>
  ▼<isebel:contents>
    <isebel:content xml:lang="deu">Wo der Weg Moltenow-Bernitt von einem Feldweg gekreuzt wird, dicht hinter Moltenow, steht noch jetzt eine alte Eiche. – Johannistag zwischen 9 und 10 sitzt der Vater wollte aber nichts davon wissen. Wenn nun der Vater am Sonntag nachmittag auf die Jagd ging, erhielt sie Besuch von ihrem Gelebten. Doch dieser mußte immer am Moltenower Schäfer Schäfer der Tat. Der Gutsherr traf ihn, wie er gerade unter der Eiche saß und schlug ihn nieder. Aber er hatte keine Ruhe im Grabe, und am Johannistage, an dem dies geschehen war, kehrte er zurück von der Arbeit. Sch. ist ängstlich, und als sie bei der Eiche vorbeikommen, nimmt er den Hut ab. „Worüm deist du dat?“ fragt der mutige F. „Hest du denn Griesen bi de Eik nich beledigten Geistes. Erzähler: Untertertianer Scharf, Penzin, Kreis Bützow; Aufzeichner: Studienrat Barnewitz, Bützow, Kreis Bützow, 1920.</isebel:content>
  </isebel:contents>
</isebel:story>

```

Abbildung 19: Beispiel 3

```

▼<isebel:story xmlns:isebel="http://www.isebel.eu/ns/isebel" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:html="http://www.w3.org/1999/xhtml" xml:id="xmd-s001-000-000-026" xml:lang="deu">
  <dc:identifier>de.wossidia.xmd-s001-000-000-026</dc:identifier>
  <isebel:purl>https://apps.wossidia.de/webapp/run/node/XMD-S001-000-000-026</isebel:purl>
  <dc:title>[26] Den Schatz nimmt der Fürst</dc:title>
  <dc:title xml:lang="deu"/>
  <dc:title xml:lang="eng"/>
  ▼<isebel:contents>
    <isebel:content xml:lang="deu">In Brunow kommt ein Jäger spät abends nach Hause. Es ist sehr dunkel, da glimmt nach seiner Meinung ein alter Stamm dicht am Weg, wie er annimmt, von Schäferj Da findet er an der Stelle eine große Menge Geld. Dadurch, daß er davon erzählt, soll's ihm später unter den Händen verschwunden sein oder der Fürst soll es zur Hälfte eingezogen haben. Er damage. Then, he found an awful lot of money. Consequently that he told about it, the money should disappeared or the ruler confiscated the half.</isebel:content>
  </isebel:contents>
</isebel:story>

```

Abbildung 20: Beispiel 4

```

▼<isebel:story xmlns:isebel="http://www.isebel.eu/ns/isebel" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:html="http://www.w3.org/1999/xhtml" xml:id="xmd-s001-000-000-016" xml:lang="deu">
  <dc:identifier>de.wossidia.xmd-s001-000-000-016</dc:identifier>
  <isebel:purl>https://apps.wossidia.de/webapp/run/node/XMD-S001-000-000-016</isebel:purl>
  <dc:title>[16] Den gefundenen Schatz nimmt der Gutsherr</dc:title>
  <dc:title xml:lang="nds"/>
  <dc:title xml:lang="deu"/>
  <dc:title xml:lang="eng"/>
  ▼<isebel:contents>
    <isebel:content xml:lang="nds">In Walkendorp uppe Braak het en Jung Swien hött. 'Dann het en! Soog 'n ganzen Schäpel Geld rutbraken. 'Dann seggt' de Herr, 'dat Geld gehürt em un' het sik dat In Walkendorf auf der Brache hat ein Junge Schweine gehütet. Da hat eine Sau einen ganzen Scheffel Geld herausgewühlt. Da sagt der Herr, das Geld gehöre ihm A boy had herded pigs on the fallow in Walkendorf. Once, a sow has dug out of the earth a whole Scheffel* money. Then, the squire said the money would be his
    </isebel:contents>
</isebel:story>

```

Abbildung 21: Beispiel 5

Beispiele 3,4,5: Storys mit umfangreichen textuellen Inhalt (content-Element)

```

<isebel:story xmlns:isebel="http://www.isebel.eu/ns/isebel" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:html="http://www.w3.org/1999/xhtml" xml:id="xmd-s001-000-000-005" xml:lang="deu">
  <dc:identifier>de.ossidia.xmd-s001-000-000-005</dc:identifier>
  <dc:purl>https://apps.ossidia.de/webapp/run/node/XMD-S001-000-000-005</isebel:purl>
  <dc:title>[5] Eddelmann wird totgeschlagen, nachdem er Matresse misshandelt hat</dc:title>
  <dc:title xml:lang="nds"/>
  <dc:title xml:lang="deu"/>
  <isebel:content>
    <isebel:content xml:lang="nds">De Eddelmann in Stellshagen het en Dieren so mißhannelt, de het toletzt 'n Stoeker nahmen un is up em ingahn. Un de Lüüd hebben em von'n Pierd krägen doot. De hebben slik flücht, de annern sind faatt. – Gerichtschlag heet dat hüt noch. As fier all richtt sind, un de sösst ran soll, kömmt Ondre von Swerin, se sullen bloß enen richten. Dat heit Wasserlämpel, da sind viele hindurhgeschwommen, die haben sich geflüchtet. Die anderen sind gefaßt worden. – Gerichtschlag heißt es ja noch heute dort. – Als fünf schon hingerichtet sind, kann der Knecht nicht mehr aufstehen, er ist tot. –</isebel:content>
  <isebel:contents>
  <isebel:places>
    <isebel:place id="100001174">
      <dc:title>Hohen Viecheln</dc:title>
      <isebel:point>
        <datacite:pointLatitude xmlns:datacite="http://datacite.org/schema/kernel-4">53.78404</datacite:pointLatitude>
        <datacite:pointLongitude xmlns:datacite="http://datacite.org/schema/kernel-4">11.51169</datacite:pointLongitude>
      </isebel:point>
      <isebel:role>narration</isebel:role>
    </isebel:place>
    <isebel:place id="100002701">
      <dc:title>Stellshagen</dc:title>
      <isebel:point>
        <datacite:pointLatitude xmlns:datacite="http://datacite.org/schema/kernel-4">53.94487</datacite:pointLatitude>
        <datacite:pointLongitude xmlns:datacite="http://datacite.org/schema/kernel-4">11.13973</datacite:pointLongitude>
      </isebel:point>
      <isebel:role>action</isebel:role>
    </isebel:place>
  </isebel:places>
  <isebel:persons>
    <isebel:person id="110002742">
      <isebel:name>Hinrichs</isebel:name>
      <isebel:gender>male</isebel:gender>
      <isebel:role>narrator</isebel:role>
      <isebel:profession>Pantoffelmacher</isebel:profession>
    </isebel:person>
  </isebel:persons>
  <isebel:keywords>
    <isebel:keyword id="XMD-M030-000-001-053" type="controlled vocabulary" provenance="MossidIA-Index">Edelleute</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-005-606" type="controlled vocabulary" provenance="MossidIA-Index">Haberlandt</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-030-625" type="controlled vocabulary" provenance="MossidIA-Index">Frevelsage</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-000-023" type="controlled vocabulary" provenance="MossidIA-Index">Wasser</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-000-595" type="controlled vocabulary" provenance="MossidIA-Index">Wasserlauf</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-000-198" type="controlled vocabulary" provenance="MossidIA-Index">Edelmann</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-001-536" type="controlled vocabulary" provenance="MossidIA-Index">totschlagen</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-001-977" type="controlled vocabulary" provenance="MossidIA-Index">Leute</isebel:keyword>
    <isebel:keyword xml:lang="nds" id="XMD-M030-000-002-413" type="controlled vocabulary" provenance="MossidIA-Index">Dirn</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-025-668" type="controlled vocabulary" provenance="MossidIA-Index">Scheid</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-038-232" type="controlled vocabulary" provenance="MossidIA-Index">misshandeln</isebel:keyword>
    <isebel:keyword id="XMD-M030-000-040-931" type="controlled vocabulary" provenance="MossidIA-Index">besprechen</isebel:keyword>
    <isebel:keyword xml:lang="deu" id="ZAW-B609-001" type="controlled vocabulary" provenance="RW-Index">Frevel</isebel:keyword>
    <isebel:keyword xml:lang="eng" id="ZAW-B609-001" type="controlled vocabulary" provenance="RW-Index">arrogant</isebel:keyword>
    <isebel:keyword xml:lang="deu" id="ZAW-B609-001" type="controlled vocabulary" provenance="RW-Index">Wahnsinn</isebel:keyword>
    <isebel:keyword xml:lang="eng" id="ZAW-B609-001" type="controlled vocabulary" provenance="RW-Index">Haberlandt</isebel:keyword>
  </isebel:keywords>
  <isebel:imageResources>
    <isebel:imageResource id="ZAW-B609-001-012-002">
      <isebel:purl>https://apps.ossidia.de/digipool/3bab5921/master</isebel:purl>
    </isebel:imageResource>
    <isebel:imageResource id="ZAW-B609-001-012-001">
      <isebel:purl>https://apps.ossidia.de/digipool/3bab5920/master</isebel:purl>
    </isebel:imageResource>
  </isebel:imageResources>
</isebel:story>

```

Abbildung 22: Beispiel 6

Aus den vorherigen Beispielen können wir erkennen, dass nicht alle XML-Dokumenten, mit denen wir es zu tun haben, die gleiche Menge an Informationen enthalten.

In Beispiel-1: Wir haben ein XML-Dokument und der Erzähler ist im Content DEU namend „H. Müller“

In Beispiel-2: Ein XML-Dokument, das nur einen Titel enthält.

In Beispiel-3: Ein XML-Dokument hat Titel und Inhalt in Deutsch.

.

.

In Beispiel 6: XML-Dokument aus dem ISEBEL-Projekt: Geschichten (Sagen, Legenden, Märchen) haben Inhalt, Ort, Person, Stichwort ...

Durch diesen Sachverhalt sind wir zu der Schlussfolgerung gekommen, einen Filter auf die XML-Dokumente anzuwenden, um nur bestimmte Dokumente zu behandeln.

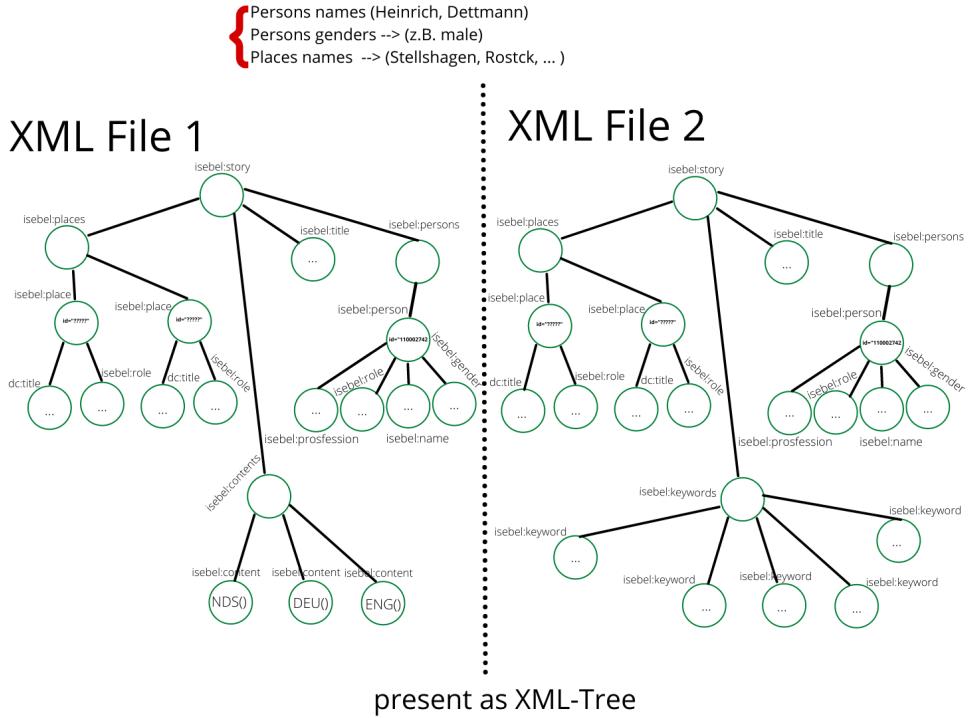


Abbildung 23: Filter - XML AS TREE

#### Filter-Funktionsweise:

Der Filter in unserem Tool funktioniert auch mit der Xpath-Sprache. Damit wählt der Benutzer die gewünschten Inhalte der XML-Dokumente über die von ihm angegebenen XPaths aus. Daraufhin scannt das Tool alle Dokumente und speichert dann die Dokumente, die den Pfaden entsprechen, in einem Array von Dokumenten. Aus diesem Satz von Dateien werden wir später die Knoten und Kanten extrahieren und erstellen. In den Beispielen in den Abbildungen 24 und 25 können wir auf praktisch-visuelle Weise die Arbeit der XPath-Sprache sehen und wie sie zu den erforderlichen Informationen gelangt. Wir haben die XML-Dokumenten in Form von Bäumen dargestellt, um das Verständnis der Arbeit von xpath zu erleichtern.

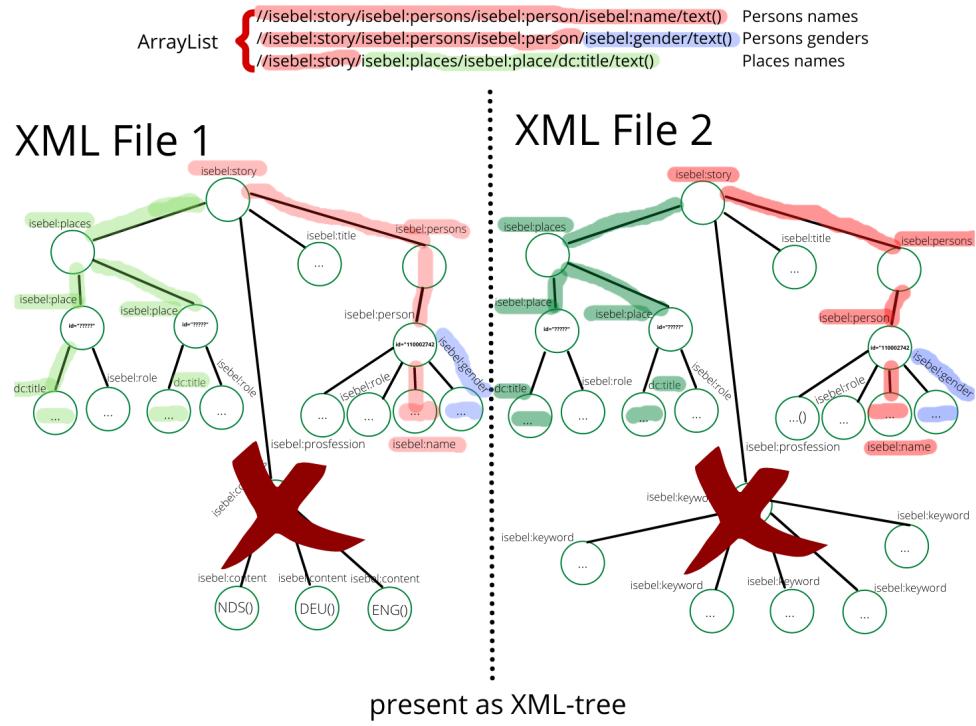


Abbildung 24: Filter-Selektion

### 3.6.2 Graph-Generator

Die Hauptfunktion von Generatoren ist es, Knoten und Kanten zu erstellen.

Nachdem wir die gewünschten XML-Dokumente herausgefiltert haben, können wir nun mit dem nächsten Schritt fortfahren und zwar mit dem Generieren von Graphen. Hierbei werden nodes und edges nach Wunsch des Nutzers erstellt, d.h. das Tool wird die regelspezifische Sprache, die den Graphen bestimmt, beachten. Diese Sprache entscheidet viel über die Gestaltung des Graphen. Das Tool parst die gefilterten XML-Dokumente Stück für Stück. Der genaue Ablauf wird auf den folgenden Abbildungen verdeutlicht.

Die XML-Dokumente werden geladen und das Tool beginnt mit dem Parsen des ersten Dokumentes. Der erste erstellte Knoten des ersten Dokuments wird als Hauptnode (Story) bezeichnet.

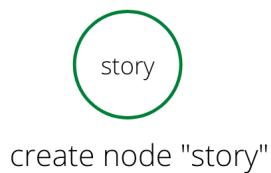


Abbildung 25: Generierungsschritt 1: Erster Knoten des ersten XML-Dokuments

Der Root-Knoten wurde erstellt und jetzt fährt das Tool damit fort, das aktuelle XML zu parsen. Das Tool findet jetzt (basierend auf der regelspezifischen Sprache) einen anderen Knoten, der im Graphen gezeichnet werden muss. In unserem Beispiel wie in Abbildung 27 als "Place" bezeichnet

Dieser Knoten könnte eventuell noch Propertys (Eigenschaften) enthalten. Abbildung 28

```
create node "place" Place
```

```
story
```

Abbildung 26: Generierungsschritt 2: Place Node

Die Property eines Graphen besteht aus: Typ der Property und deren Wert.

Da dieser Knoten eine Property enthält, wird sie dieser hinzugefügt. In der Abbildung ist der Typ der Property: name und der Wert: Stellshagen



Abbildung 27: Generierungsschritt 3: Add property

Abbildung 29 zeigt, woher der Wert der Property eigentlich kommt. Es ist ein Ergebnis des Xpath:

`isebel:places/isebel:place/dc:title/text()`

Denn der Aufruf dieses xpath in unserem Tool ergibt den Wert: Stellshagen.

Und für diesen Type wird der String = Name zugewiesen. In der XML-Dokument heißt das: "dc:title".

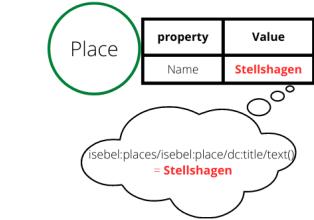


Abbildung 28: Generierungsschritt 4: Type and Value

Dieser Vorgang wird für jedes Unterelement im aktuellen XML-Dokument wiederholt.  
Iterator

Jedes Mal, wenn das Tool ein Element findet, das den Regeln des Benutzers entspricht, wird ein Knoten erstellt und die zugehörigen Properties hinzugefügt.

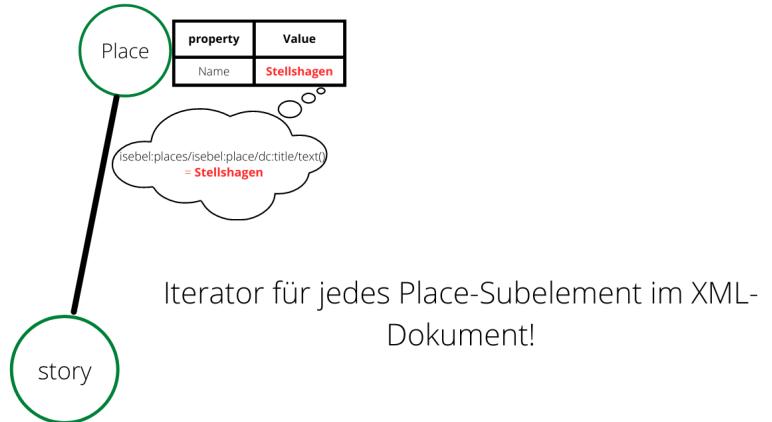


Abbildung 29: Generierungsschritt 5: Edge

Nachdem der zweite Knoten erstellt wurde, wird nun eine Kante dazwischen erstellt. Und so wiederholt sich den Prozess bis wir einen vollständigen Graphen aus dem aktuellen Dokument erstellt haben, der aus Knoten und Kanten besteht.

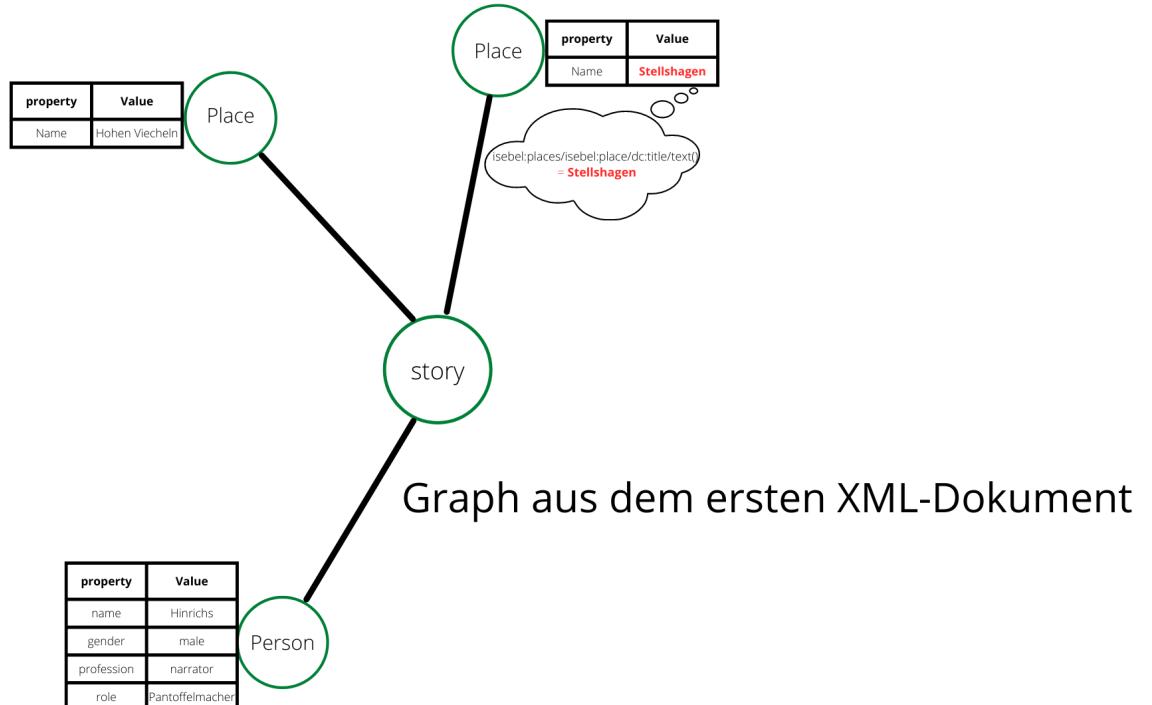


Abbildung 30: Graph-Generierung: 1st Graph from 1st XML

Das Vollbild des Graphen aus dem ersten XML-Dokument.

Nach mehreren Schritten wird der Graph aus dem ersten XML-Dokument erstellt.  
Abbildung 31

Wir sehen hier einen vollständigen Graphen, der aus 4 Knoten besteht und zwar: Story, 2 mal Place (mit properties) und Person (mit properties).

Und damit fährt das Tool fort, das nächste XML-Dokument zu parsen.

zur Bemerkung:

Ein standardmäßiger Startknoten für jedes XML-Dokument ist der Story-Knoten, und dann werden die anderen Knoten generiert, die zu demselben XML-Dokument gehören. z.B. Place, Person, Keywords ... usw.

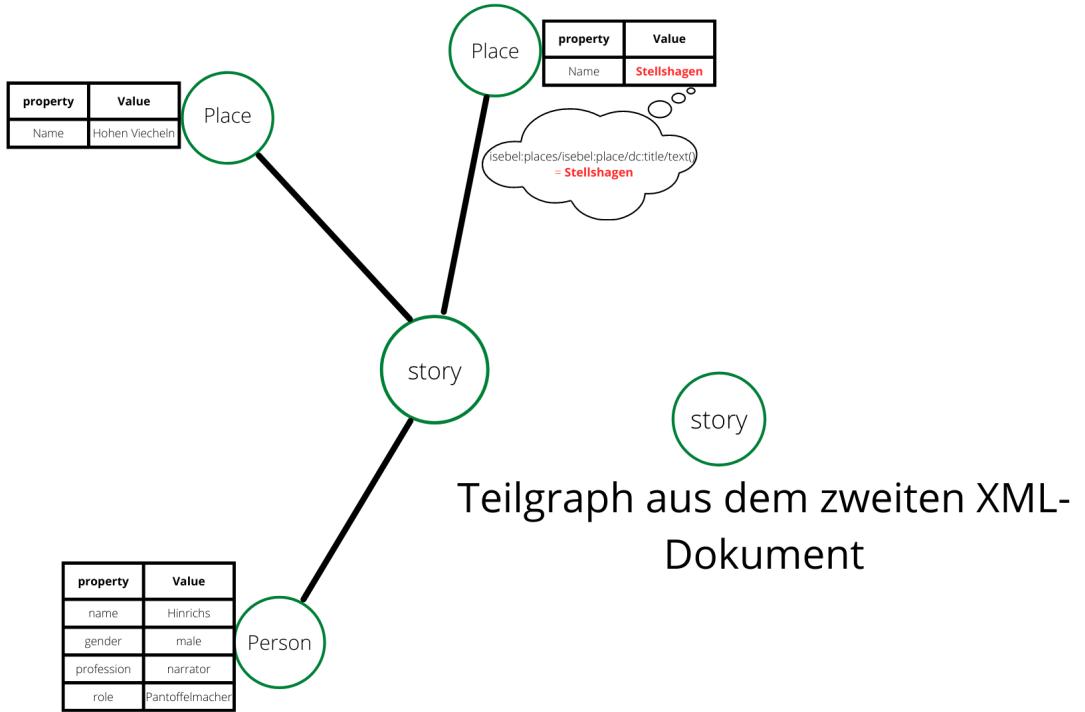


Abbildung 31: Generierungsschritt 6: next Document

Das Tool beginnt nun mit dem Parsen des nächsten XML-Dokuments. Erneut wird der erste Knoten des neuen Dokumentes erstellt und wiederum als Haupt-Node (Story) bezeichnet.

Der Root-Knoten wurde erstellt und das Tool fährt fort (basierend auf der regelspezifischen Sprache) die weiteren Knoten zu erstellen.

Jedem Knoten werden die zugehörigen properties zugeordnet.

Jeder neue Knoten ist durch eine Kante mit dem Root-knoten verbunden.

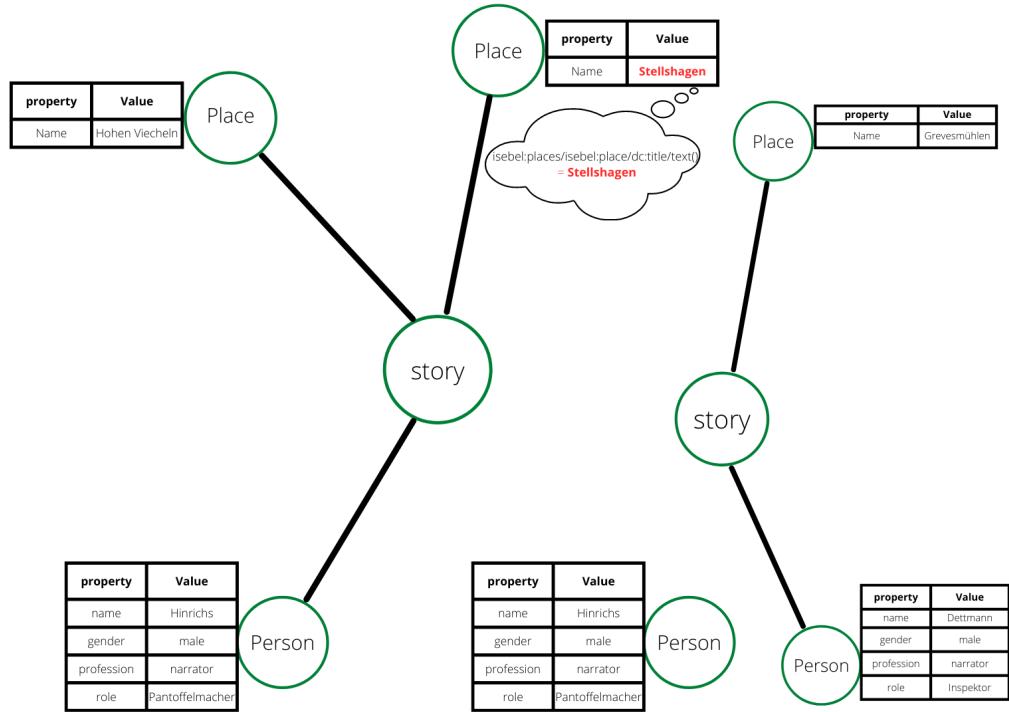


Abbildung 32: Graph-Generierung: Compare Prperties-1

Bei dem Generieren des nächsten Graphen sind einige Dinge zu beachten.

Beim Erstellen der neuen Knoten muss basierend auf dem Knotentyp der Wert jeder Eigenschaft(property) mit den Werten der Knoten der vorherigen Graphen verglichen werden, damit keine Knoten mit denselben Werten und Eigenschaften erstellt werden.

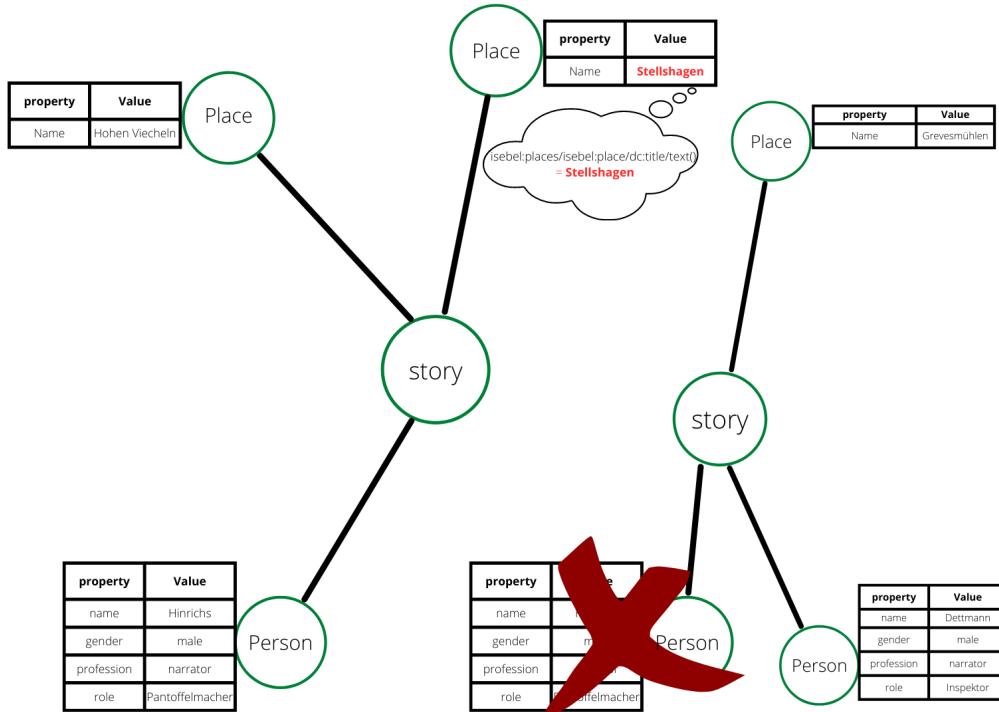


Abbildung 33: Graph-Generierung: Compare Properties-2

In unserem in den Abbildungen gezeigten Beispiel werden wir den Fall behandeln, wie Knoten mit identischen properties + identischen properties-anzahl verarbeitet werden. Die anderen Fälle werden in dem nächsten Kapitel ausführlich behandelt.

Wie bereits erwähnt jede property eines Knoten besteht aus: Type und dazugehöriger Wert.

Das Tool hat nun einen neuen Knoten gelesen, dieser Knoten hat den Typ Person. Nun werden die Eigenschaften dieses Knotentyps mit den Eigenschaften desselben Knotentyps aus dem vorherigen XML-Dokument verglichen.

In unserem Fall, der in dem Beispiel auf Abbildungen gezeigt wird, haben wir folgende Properties von dem ersten Knoten aus dem ersten Graph(G1) :

| name = Hinrichs | gender = male | profession = narrator | role = Pantoffelmacher |

Und folgende Properties von dem zweiten Knoten aus dem zweiten Graph(G2):

| name = Hinrichs | gender = male | profession = narrator | role = Pantoffelmacher |

Nun wird jede property aus dem ersten Knoten(G1) mit den Properties aus dem zweiten Knoten(G2) verglichen.

Da die properties identisch sind, wird in diesem Fall kein neuer Knoten erzeugt, sondern eine Kante zwischen dem Root-knoten (Story) aus G2 und dem Knoten Person aus G1 erstellt.

Auf Abbildung 35 ist nun ein Vollständiger Graph.

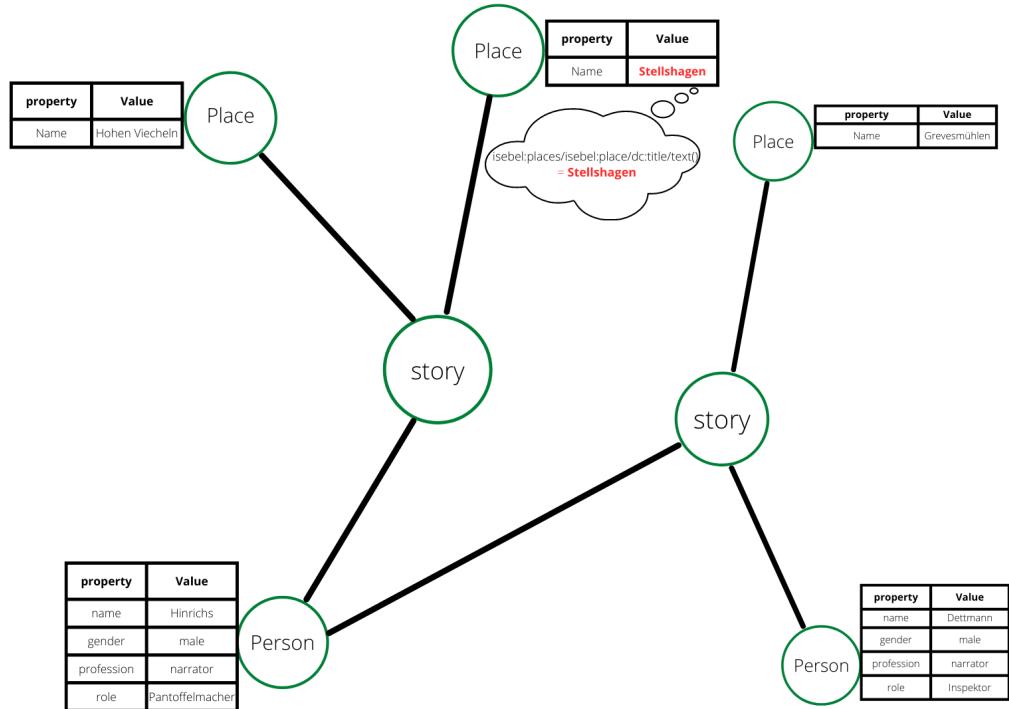


Abbildung 34: Graph-Generierung: Compare Prperties-3, Graph is ready

### 3.6.3 Node-Replacement Cases

Duplikaterkennung Varianten bestehen aus zwei Teilen:

- 1) Es wird kein neuer Knoten erstellt. Abbildungen [37,38]
- 2) Es wird ein neuer Knoten erstellt. Abbildungen [39,40]

Die Varianten wurden anhand von Beispielen aus der ISEBEL-Xml-Storys behandelt. Mehrere Knotentypen werden nach der Generierung aus den ISEBEL-XML-Storys hervorgehen, Beispiele sind Person, Place, Keyword ...

Angenommen, wir möchten über das Tool ein Gaphen aus 2 XML-Dokumenten generieren, sodass wir die folgenden Parameter haben:

- 1) G — steht als Abkürzung für Graph
- 2) N — steht als Abkürzung für Knoten
- 3) P — steht als Abkürzung für Property
- 4) K — steht als Abkürzung für Key
- 5) V — steht als Abkürzung für Value

Jeder Graph besteht aus Knoten. Die Knoten beeinhalten Properties. Jede Property besteht aus einem Schlüssel und dazugehörigen Wert.

daraus folgt:

G1(N1S, N1P, N1O, N1O)

N1S: Knoten aus Graph 1 Type Story

N1P: Knoten aus Graph 1 Type Person

N1O: Knoten aus Graph 1 Type Ort

N1O: Knoten aus Graph 1 Type Ort

G2(N2S, N2P, N2P, N2O) ;

N2S: Knoten aus Graph 2 Type Story

N2P: Knoten aus Graph 2 Type Person

N2P: Knoten aus Graph 2 Type Person

N2O: Knoten aus Graph 2 Type Ort

1S, 2S .... identifier ; sie werden so gelesen: N1S = Knoten aus Graph 1 Type Story.

N2P = Knoten aus Graph 2 Type Person.

Knoten enthalten properties, daher können sie folgendermaßen geschrieben werden:

N1S(P1, P2, P3 ....)

N2S(P1, P2, P3 ....)

Properties bestehen aus Schlüssel und Werte:

N1S: P1(Key, Value) — zB. Key = name, Value = Hinrichs

N2S: P1(Key, Value) — zB. Key = name, Value = Dettmann

Diese Werte G1: N1S: P1 (Key, Wert), G2: N2S: P1 (Key, Wert) werden tatsächlich verglichen und daraus wird entschieden, ob ein neuer Knoten erstellt wird oder nicht, basierend auf dem Ergebnis des Vergleichs und der Anzahl der Properties von jedem Knoten.

Nun werden die 2 Varianten (Es wird k\*ein neuer Knoten erstellt) anhand der folgenden Abbildungen näher erläutert. Dabei verwenden wir den Knotentyp Person aus den ISEBEL-XML-Storys.

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
name	Hinrichs	name	Hinrichs
gender	male	gender	male
profession	narrator	profession	narrator
role	Pantoffelmacher	role	Pantoffelmacher

Es wird kein neuer Knoten erstellt

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
name	Hinrichs	name	Hinrichs
gender	male	gender	male
profession	narrator	profession	narrator
role	Pantoffelmacher		

Es wird kein neuer Knoten erstellt

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
name	Hinrichs	name	Hinrichs
gender	male	gender	male
profession	narrator	profession	narrator
role	Pantoffelmacher		

Es wird kein neuer Knoten erstellt

N = Konten  
G = Graph

Abbildung 35: Variante 1

Kein neuer Knoten wird erstellt wenn:

- Die Properties von beiden Knoten vollständig identisch sind.
- Die properties der Knoten sind weitgehend identisch Der property-name muss identisch sein) und die Anzahl der Eigenschaften des zweiten Knotens müssen um 1 oder 2 properties geringer sein.
- Der zweite Knoten ist nur über die Name-property und eine weitere property verfügbar, und der erste Knoten ist über die gleiche Name-property verfügbar, hat aber mehr properties, dann wird ein neuer Knoten erstellt und die property wird vom zweiten Knoten an den ersten übergeben.

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
name	Hinrichs		
gender	male	gender	male
profession	narrator	profession	narrator
role	Pantoffelmacher	role	Pantoffelmacher

Es wird kein neuer Knoten erstellt

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
name	Hinrichs		
gender	male	gender	male
profession	narrator	profession	narrator
role	Pantoffelmacher		

Es wird kein neuer Knoten erstellt

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
name	Hinrichs		
gender	male		
profession	narrator		

Es wird kein neuer Knoten erstellt

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
name	Hinrichs		
gender	male		
profession	narrator		

Es wird kein neuer Knoten erstellt

N = Konten  
G = Graph

N = Konten  
G = Graph

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
name	Hinrichs		
gender	male		
profession	narrator		

Es wird kein neuer Knoten erstellt

Abbildung 36: Variante 2

Kein neuer Knoten wird erstellt, wenn:

- Die Properties von jedem Knoten sehr identisch sind und es fehlt die Property name von dem zweiten Knoten. (Warnung wird generiert, dass der Schlüsselwert fehlt!)
- Die Properties von jedem Knoten zum Teil identisch sind und es fehlt die Property name und noch andere property von dem zweiten Knoten.(Warnung wird generiert, dass der Schlüsselwert fehlt!)
- Im ersten Knoten ein paar Properties sind aber im zweiten Knoten nur eine Name-property verfügbar und identisch ist. (Der vorhandene Knoten wird übernommen!)

N1(G1)		K2(G2)		N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value	property (Key)	Value	property (Key)	Value
name	Hinrichs	name	Dettmann	name	Dettmann	name	Hinrichs
gender	male	gender	male	gender	male	gender	male
profession	narrator	profession	narrator	profession	narrator	profession	narrator
role	Pantoffelmacher	role	Inspektor	role	Pantoffelmacher	role	Pantoffelmacher

Ein neuer Knoten wird erstellt

Ein neuer Knoten wird erstellt

**N = Konten**  
**G = Graph**

N1(G1)		N2(G2)		N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value	property (Key)	Value	property (Key)	Value
name	Hinrichs	name	Dettmann	name	Hinrichs	name	Hinrichs
		gender	male	gender	male	gender	male
		profession	narrator	profession	narrator	profession	narrator
		role	Pantoffelmacher	role	Pantoffelmacher	role	Pantoffelmacher

Ein neuer Knoten wird erstellt

Kein neuer Knoten wird erstellt

Abbildung 37: Variante 3

- Wenn properties zum großen Teil identisch sind, aber die Name-property im zweiten Knoten nicht identisch ist. Ein neuer Knoten wird erstellt.
- Wenn properties zum großen Teil identisch sind, aber die Name-property im zweiten Knoten nicht identisch ist. Ein neuer Knoten wird erstellt.
- Die im ersten Knoten verfügbaren properties mit den properties im zweiten Knoten identisch sind, und der zweite Knoten hat mehr properties als der erste. Der alte Knoten wird ergänzt um die zusätzliche Eigenschaften, es wird kein neuer Knoten erzeugt!

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
name	Hinrichs	name	Hinrichs
gender	male	gender	male
profession	narrator	profession	narrator
		role	Inspektor

Kein neuer Knoten, Eigenschaften ergänzen beim alten Knoten!

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
name	Hinrichs	name	Hinrichs
gender	male	gender	male
profession	narrator	profession	narrator
role	Pantoffelmacher		

Alten Knoten lassen und Eigenschaften vereinigen.

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
name	Hinrichs	name	Hinrichs
profession	narrator	profession	narrator
		role	Pantoffelmacher

Kein neuer Knoten, Eigenschaften ergänzen beim alten Knoten!

N = Konten  
G = Graph

N1(G1)		N2(G2)	
property (Key)	Value	property (Key)	Value
		name	Hinrichs
		gender	male
		profession	narrator
		role	Pantoffelmacher

Neuer Knoten erzeugt, aber für die fehlende Eigenschaft(en) beim alten Knoten eine Warnung ausgeben!

Abbildung 38: Variante 4

Kein neuer Knoten, Eigenschaften ergänzen beim alten Knoten. Und so können wir uns das Erstellen neuer Kanten sparen.

Alten Knoten lassen und Eigenschaften vereinigen. Kein neuer Knoten.

Neuer Knoten erzeugt, aber für die fehlende Eigenschaft(en) beim alten Knoten eine Warnung ausgeben. Und eine Kante 'is related' wird erzeugt, die beide Knoten verbindet.

### 3.6.4 Graph-Exporter

Aufgabe des Graph-Exporters ist es, die erzeugten, programm-internen Graphstrukturen in einem geeigneten Format in Dateien zu speichern. Mögliche Formate sind:

-CSV:

Eine CSV-Datei [39] ist eine einfache Textdatei, die eine Liste von Daten enthält.

Diese Dateien werden häufig verwendet, um Daten zwischen verschiedenen Anwendungen auszutauschen. Beispielsweise unterstützen Datenbanken und Kontaktmanager häufig CSV-Dateien.

Diese Dateien können manchmal als durch Zeichen getrennte Werte oder durch Kommas getrennte Dateien bezeichnet werden. Sie verwenden meistens das Kommazeichen (,), um Daten zu trennen (oder zu definieren), aber manchmal verwenden sie andere Zeichen, wie z. B. Semikolons (;).

Die Idee der CSV ist, dass komplexe Daten aus einer Anwendung in eine CSV-Datei exportieren lassen und dann die Daten aus dieser CSV-Datei in eine andere Anwendung importiert werden können.[39]

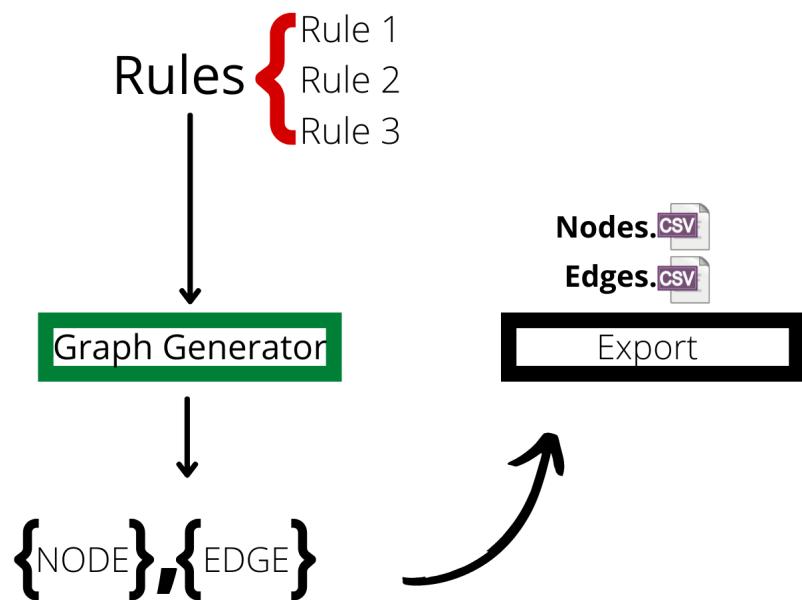


Abbildung 39: Exporter

-Graphml:

GraphML ist ein XML-basiertes Dateiformat zur Darstellung von Graphen. GraphML besteht aus einem Sprachkern zur Beschreibung von Graphstrukturen und einem Erweiterungsmechanismus für anwendungsspezifische Daten.[40]

-Gexf:

GEXF (Graph Exchange XML Format) ist eine Sprache zur Beschreibung komplexer Netzwerkstrukturen, ihrer zugehörigen Daten und Dynamik. Die gexf-Spezifikationen, sind erweiterbar, offen und für echte spezifische Anwendungen geeignet.[41]

Das Modul hat folgende Aufgaben:

- Erzeugen der Dateien
- Ausgabe der Knoten mit Label und Properties
- Ausgabe der Kanten mit Label und Properties.

Implementiert ist aktuell die Ausgabe als .CSV, in zwei separate Dateien, die die Knoten bzw. die Kanten enthalten.

Ein Beispiel für die Ausgabe(Abbildung [41]), die den Graphen in Abbildung [35] darstellt.

Node. 

```
"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"
```

Edge. 

```
"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"
```

Abbildung 40: Ausgabe .CSV

### 3.6.5 Graph-Szenarien Beispiele

Mögliche Graph-Szenarien Beispiele für X2G-Tool beim Umgang mit Isebel-XML-Dokumenten:

- Alle Storys von einer bestimmten Person XY
- Alle Storys, die an einem bestimmten Ort erzählt wurden (Stellshagen, Wismar...)
- Einfluss des Geschlechts des Erzählers auf die Hexen- oder Werwolf-Sagen
- Alle Storys, die in 3 Sprachen (DEU, NDS, ENG) verfügbar sind

## 4 Zusammenfassung und Schlussfolgerung

Das Ziel dieser Arbeit war es, die Erstellung von Graphen aus XML-Dokumenten basierend auf den Eingaben des Benutzers zu ermöglichen. Für dieses Ziel wurde ein Tool X2G konzipiert, dass Xpath als eingabe verwendet.

Mit der Xpath-Sprache werden nicht nur die Textinhalte zwischen den Tags in Graphen dargestellt, sondern auch Texte und Kontexte, die im XML gar nicht vorhanden waren, werden auch in den Graphen dargestellt.

Realisiert wurden folgende wesentliche Bestandteile:

- Filter: XML-Daten werden gefiltert
- Generator: Knoten und Kanten werden generiert
- Exporter: Ausgabe der generierten Knoten und Kanten in bestimmten Formaten

Ein Problem bei der Konzipierung waren die Regeln zum Erzeugen von Knoten. Es wurden dazu verschiedene Varianten analysiert und Lösungsmöglichkeiten aufgezeigt.

Die Ausgabe der generierten Graphen kann prinzipiell in verschiedenen Formaten erfolgen. Umgesetzt ist aktuell die Ausgabe einer Knoten und Kantentabelle jeweils als CSV-Datei.

X2G kann und will jedoch nicht mit professionellen Lösungen konkurrieren, die heutzutage auf dem Markt entwickelt werden. Diese Lösungen bzw. Programs werden von ganzen Teams aus professionellen Entwicklern, Architekten und anderen Personen entwickelt, die an der Anwendungsentwicklung beteiligt sind. X2G als neue Idee versucht zu zeigen, wie XML-Visualisierung auf eine andere Art und Weise durchgeführt werden kann, und kann beispielsweise zum Erstellen von Graphen aus XML-Dokumente verwendet werden, ohne die gesamte Dokumentstruktur 1-zu-1 in Graphstrukturen zu überführen.

Natürlich gibt es einige Möglichkeiten, wie X2G verbessert werden könnte. Die folgende Liste zeigt einige Ideen, die X2G zu einer besseren Anwendung machen könnten:

- Erlauben das Einfügen von Elementen durch Eingabefeld
- Verbesserte Kontexthilfe für XML-Schemas.
- Benutzern ermöglichen, die Auswahl der Elemente besser anzupassen.
- GUI hinzufügen, um es für den Benutzer einfacher zu machen \*Abbildungen[41-43] sehen. Eine Vorstellung für eine GUI
- Auswahl der Konten und Kanten aus den Zusammenhängen oder Texten verbessern
- Exporter als andere Formaten bereitstellen

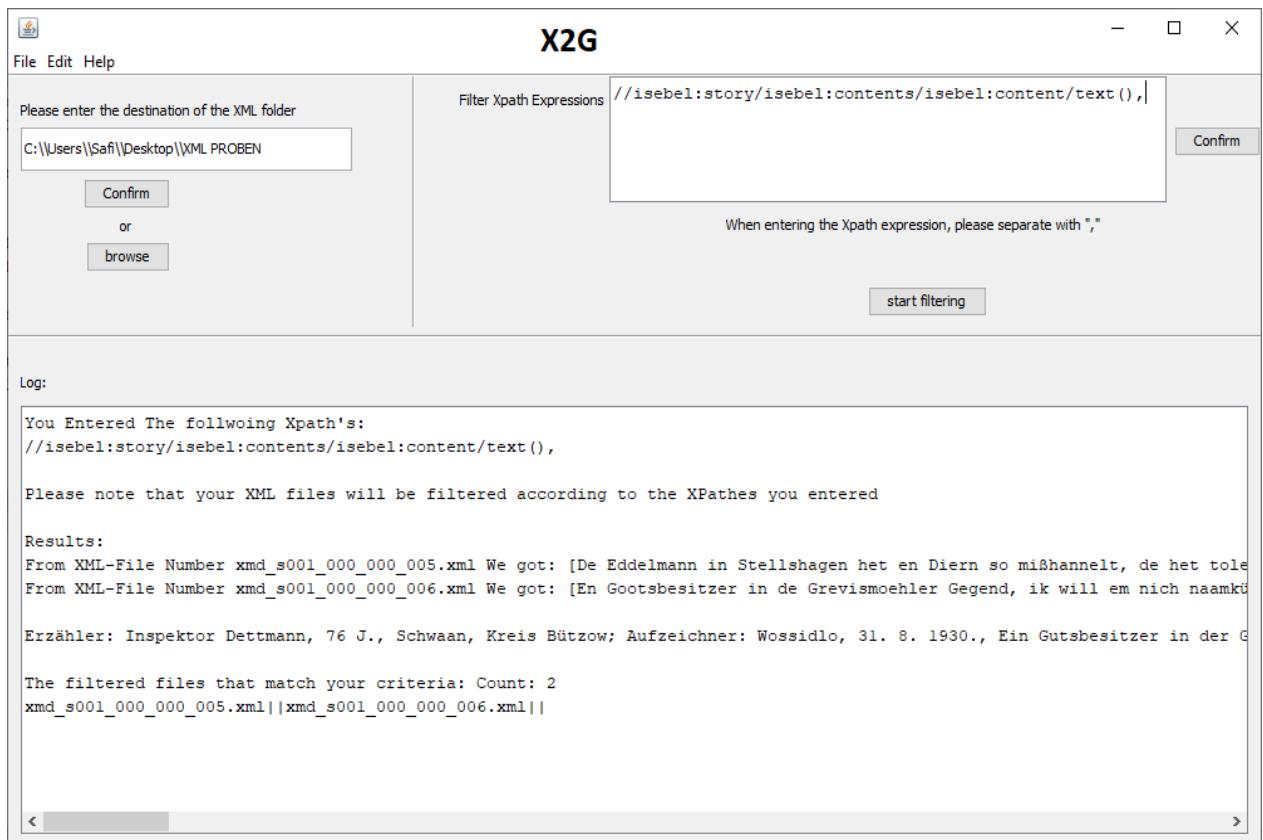


Abbildung 41: X2G Step 1

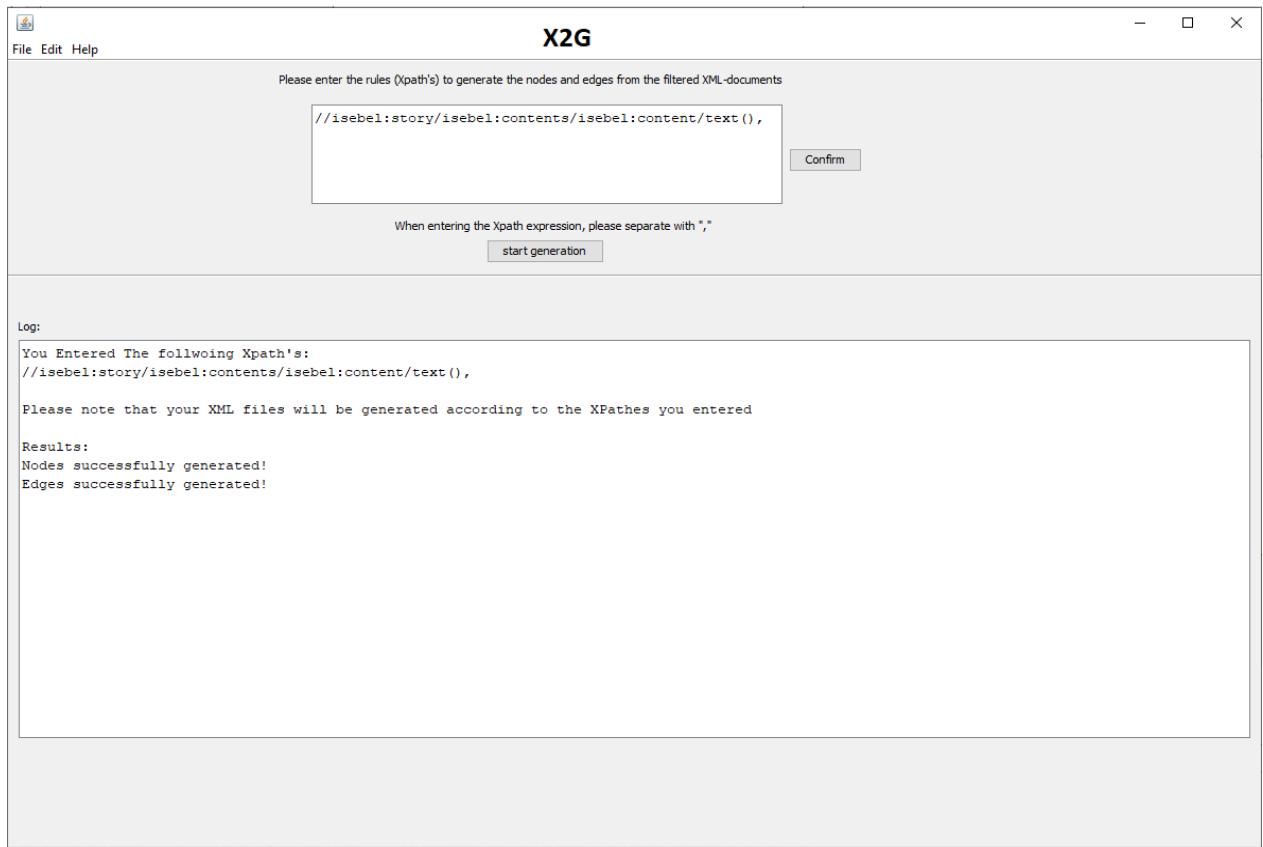


Abbildung 42: X2G Step 2

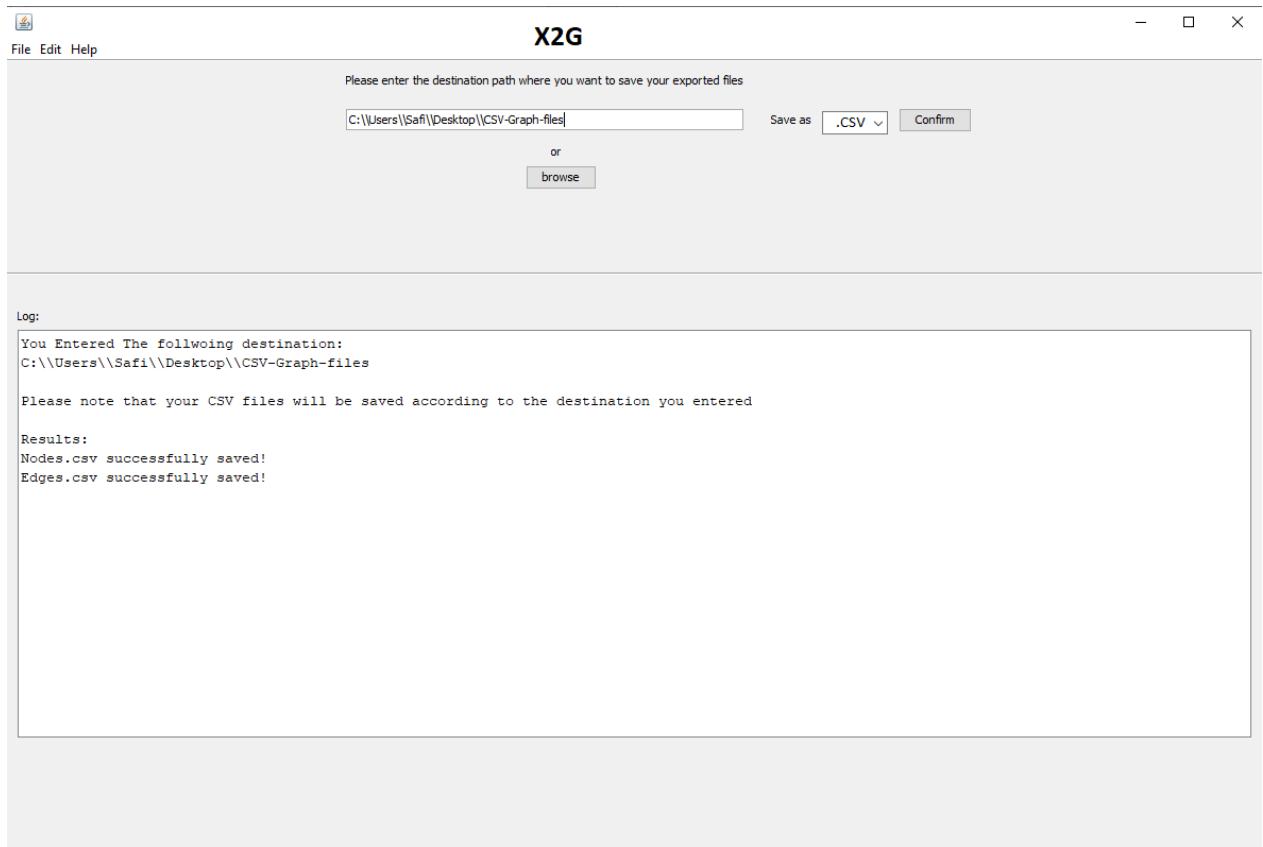


Abbildung 43: X2G Step 3

## 5 Literaturverzeichnis

- [2] Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C, November 2008, <https://www.w3.org/TR/xml/#dt-wellformed>
- [4] Transforming XML Data with XSLT  
<https://docs.oracle.com/javase/tutorial/jaxp/xslt/transformingXML.html>
- [5] yworks, application to quickly and effectively generate high-quality diagrams, <https://www.yworks.com/products/yed>
- [6] XQuery 3.1: An XML Query Language, 21 March 2017, <https://www.w3.org/TR/xquery-31/>
- [7] World Wide Web Consortium, [http://w3c.org/.](http://w3c.org/)
- [8] Namespaces in XML 1.1 (Second Edition), W3C, August 2006, <http://www.w3.org/TR/xml-names11/>
- [9] Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C, November 2008, <http://www.w3.org/TR/REC-xml/dt-doctype>
- [10] XML Schema Part 0: Primer Second Edition, W3C, October 2004, <http://www.w3.org/TR/xmlschema-0/>
- [13] XML Path Language (XPath) <https://www.w3.org/TR/1999/REC-xpath-19991116/>
- [14] XML DTD [https://www.w3schools.com/xml/xml\\_dtd\\_intro.asp](https://www.w3schools.com/xml/xml_dtd_intro.asp)
- [15] Document Object Model, W3C, January 2005, <http://www.w3.org/DOM/>
- [16] Simple API for XML, November 2001, <http://www.saxproject.org/>
- [17] Clark, J., DeRose, S.: XML Path Language, W3C, November 1999, <http://www.w3.org/TR>xpath/>
- [22] James Clark and Steve DeRose. XML path language version 1.0, November 1999. W3C Recommendation. <http://www.w3.org/TR/1999/REC-xpath-19991116>.
- [29] R. Bardohl, H. Ehrig, J. de Lara, and G. Taentzer. Integrating Meta Modelling Aspects with Graph Transformation for Efficient Visual Language Definition and Model Manipulation. In M. Wermelinger and T. Margaria, editors, Proceedings of FASE 2004, pages 214–228, 2004.
- [31] Holger Meyer, Alf-Christian Schering and Christoph Schmitt, WossidA — The Digital Wossidlo Archive, in: Holger Meyer, Christoph Schmitt, Thomas Jansen and Alf-Christian Schering (Hrsg.), Corpora ethnographica online — Strategien der Digitalisierung kultureller Archive und ihrer Präsentation im Internet, Volume 5 of Rostocker Beiträge zur Volkskunde und Kulturgeschichte, Waxmann, 2014, 61–84.
- [32] selecting nodes in an XML document [https://www.w3schools.com/xml/xpath\\_syntax.asp/](https://www.w3schools.com/xml/xpath_syntax.asp) W3Schools is Powered by W3.CSS. Zugriff-datum 22.02.2022
- [33] XPath Axes [https://www.w3schools.com/xml/xpath\\_axes.asp/](https://www.w3schools.com/xml/xpath_axes.asp) W3Schools is Powered by W3.CSS. Zugriff-datum 22.02.2022

[34] Property Graph <https://neo4j.com/docs/getting-started/current/graphdb-concepts/>  
Zugriff-datum 22.02.2022

[35] XPath axes: partition of tree nodes. <https://www.researchgate.net/figure/XPath-Axes-Partition-from-Context-Nodefig329646041/> Zugriff-datum 18.01.2022

[36] XPath Axes and their Shortcuts

<https://our.umbraco.com/documentation/reference/template-macros/xslt/xpath-axes-and-their-shortcuts/> Zugriff-datum 18.01.2022

[38] Wohlgeformtes Dokument [https://de.wikibrief.org/wiki/Well-formed\\_document](https://de.wikibrief.org/wiki/Well-formed_document) / Zugriff-datum 22.01.2022

[39] CSV Comma-separated values [https://en.wikipedia.org/wiki/Comma-separated\\_values](https://en.wikipedia.org/wiki/Comma-separated_values) / Zugriff-datum 02.03.2022

[40] .GRAPHML File Extension <https://de.wikipedia.org/wiki/GraphML> / Zugriff-datum 02.03.2022

[41] .GEXF File extension Read and write graphs in GEXF format.

<https://networkx.org/documentation/stable/reference/readwrite/gexf.html> / Zugriff-datum 02.03.2022

[42] XML-Namespace, [https://en.wikipedia.org/wiki/XML\\_namespace](https://en.wikipedia.org/wiki/XML_namespace)

[43] XML namespace W3C Recommendation 8 December 2009 <https://www.w3.org/TR/xml-names/>

[44] SAX XML <https://de.wikikinht.com/162844-simple-api-for-xml-JDFYHB>

[45] XML Schema W3C XML Core Working Group. <https://www.w3.org/XML/Schema>