universitätbonn

Ein Toolkit für die policybasierte Pseudonymisierung mit Verfügbarkeitsoptionen

Martin Wehner
Matr.-Nr. 2629465

Bachelorarbeit

1. Prüfer: Prof. Dr. Michael Meier
2. Prüfer: Jun.-Prof. Dr.-Ing. Delphine Reinhardt
Betreuerin: Dipl.-Inform. Saffija Kasem-Madani
Institut für Informatik IV
Arbeitsgruppe für IT-Sicherheit
Rheinische-Friedrich-Wilhelms-Universität Bonn

Selbstständigkeitserklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Bonn, 22. November 2016

Martin Wehner

Zusammenfassung

Das Ziel dieser Bachelorarbeit war es, ein Tool für die Pseudonymisierung von Daten anhand eines Regelwerks (engl. Policy) zu erstellen. Daten sollen dabei so pseudonymisiert werden, dass sie noch Teilinformationen der Klartextdaten enthalten können. Für die erstellten Pseudonyme soll es eine Möglichkeit geben, diese an jeweils eine Rolle oder einen Zweck zu binden, wodurch der Zugriff auf die Pseudonyme eingeschränkt wird. Diese beiden Eigenschaften heißen Verfügbarkeitsoptionen und Bindungen von Verfügbarkeitsoptionen und wurden als erstes in den Grundlagen definiert. Als Zweites wurde betrachtet, wie eine Policy aufgebaut sein soll, damit sie flexibel und erweiterbar ist. Dafür wurden verbreitete Datenformate zur Haltung von Daten, wie XML, im Hinblick auf ihre Struktur untersucht und zwei Arten der Behandlung von Datensätzen erarbeitet.
Damit beliebige Textdateiformate verarbeitet werden können, wurde die Möglichkeit der Transformation einer Datensammlung vorgestellt.
Um eine Policy in der definierten Policysprache benutzerfreundlicher zu erstellen, wurde das Tool "XML PolicyBuilder" entworfen und implementiert.
Als letztes wurde ein Pseudonymisierungstool um die neuen, in der Policysprache enthaltenen Funktionen erweitert, sodass Daten durch unterschiedliche Funktionen aufbereitet werden können und dadurch unterschiedliche Teilinformationen anbieten können.
Die implementierten Komponenten wurden abschließend funktional und bzgl. ihrer Laufzeit und dem Speicheraufwand überprüft.
Schlussfolgernd ist das gesamte erstellte Tool gut für die Pseudonymisierung von kleinen Datensammlungen nutzbar und kann durch den flexiblen und modularen Aufbau einfach um weitere Funktionen erweitert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Ein Austausch von Daten ist in vielen Bereichen gegenwärtig nicht mehr wegzudenken. Wenn es um die Sicherheit geht, kann die Weitergabe von aufgezeichneten Daten, wie Logdateien, eines angegriffene Unternehmens andere Unternehmen in der Erkennung eines Angriffs und Ergreifung von Gegenmaßnahmen unterstützen. In Deutschland gibt es z.B. das Computer Emergency Response Team des Bundes, abgekürzt CERT-Bund [1], das primär den Bundesbehörden hilft, Vorfallsmeldungen analysiert und dementsprechend Empfehlungen herausgibt. Für den globalen Austausch über Vorfälle wurde das "Forum for Incident Response and Security Teams", abgekürzt FIRST [2], gegründet.

Neben der Weitergabe von solchen Vorfallsmeldungen bestehen auch in anderen Bereichen Bestrebungen Daten auszutauschen. Im Bezug auf persönliche Daten ist in Deutschland die Techniker Krankenkasse an Daten, die den Gesundheitszustand einer Person betreffen, interessiert und plant ein Bonusprogramm, durch das Kunden Geld sparen können. Zu diesen Daten gehören die Anzahl an Schritten und andere Werte, die u.a. durch Fitnessarmbänder gemessen werden können. Kunden sollen dadurch stärker über Risiken informiert werden, aber auf der anderen Seite je nach gesundheitsbewusstem Verhalten bezahlen [3].

Bei diesen Szenarien gibt es die zwei gegensätzlichen Faktoren, Datenschutz und Verwendbarkeit von Daten. In beiden Fällen möchte die Person oder das Unternehmen, dass die Daten weitergibt, ihre Privatsphäre oder andere vertrauliche Daten schützen. Im Gegensatz dazu möchte das Unternehmen, dass die Daten erhält, konkrete und genaue Ergebnisse bei der Analyse der Daten erhalten. Nur auf das Vertrauen in ein Unternehmen zu setzen, dass es die benötigten Daten erhält und sie nur für den vereinbarten Zweck nutzt, ist nicht ausreichend. Ein aktuell in der Forschung befindlicher Ansatz ist, die weitergegebenen Daten so aufzubereiten, dass sie nur für den vereinbarten Zweck genutzt werden können und für andere Zwecke, die Unternehmen oder einzelne Personen haben könnten, unbrauchbar wären. Ein Ansatz dafür sind Verfügbarkeitsoptionen. Durch eine Verfügbarkeitsoption eines Pseudonyms können nur noch ausgewählte Informationen des Klartextes betrachtet werden. Bevor Daten weitergegeben werden, werden diese anhand einer Vereinbarung (engl. Policy) so aufbereitet, dass sie nur für bestimmte Zwecke genutzt werden können bzw. entsprechende Verfügbarkeitsoptionen anbieten. Damit auch dies von jedem durchführbar ist, wird in dieser Arbeit zum einen ein Tool für die Erstellung einer Policy und zum anderen ein Tool für die Aufbereitung von Daten anhand einer erstellten Policy vorgestellt.

1.1 Zielsetzung

Dieser Bachelorarbeit liegen drei Arbeiten [4, 5, 6] zugrunde. Das Ziel ist es, auf diesen Arbeiten aufbauend ein Toolkit anzubieten, durch das eine Policy erstellt werden kann und mit dem Daten anhand dieser Policy aufbereitet werden können.

In einer früheren Arbeit [4] wurde die Policy durch die Nutzung des XML-Schemas eXtensible Access Control Markup Language (XACML)[7] erstellt. Als ein Ergebnis stellte sich heraus, dass in solch einer Policy die einzelnen Verfügbarkeitsanforderungen aufgrund XACML nicht menschenlesbar dargestellt werden können. Deshalb ist das erste Ziel ein Tool in Python zu implementieren, das eine Policy in einer eigenen optimierten XML-basierten Sprache erstellt.

Als Zweites sollen am, bei [5] erstellten, Pseudonymization Tool Designoptimierungen vorgenommen werden und zudem angepasst werden, sodass es Policies in der eigenen XML-basierten Sprache versteht, um Daten anhand solch einer Policy zu verarbeiten.

Als Drittes sollen von den in [6] implementierten Pseudonymisierungsfunktionen ausgewählte Funktionen zum Pseudonymization Tool hinzugefügt werden, um weitere Verfügbarkeitsoptionen anbieten zu können.

Während der Implementierung sollen kontinuierlich Funktionssicherheitstests zur Qualitätssicherung duchgeführt und nach der Fertigstellung alle implementierten Tools ausführlich evaluiert werden.

1.2 Aufbau

Diese Arbeit untergliedert sich wie folgt in 8 Kapitel. In Kapitel 2 werden die grundlegenden Begriffe und Verfahren, wie personenbezogene Daten, Datensammlung, Pseudonymisierung, Verfügbarkeitsoptionen und Bindung einer Verfügbarkeitsoption erörtert, die in den nachfolgenden Kapiteln verwendet werden. In Kapitel 3 werden verbreitete Textdateiformate und Arten von Behandlungen von Datensätzen betrachtet, um dafür anschließend eine Policysprache zu definieren. Als Letztes wird in diesem Kapitel vorgestellt, wie beliebige Textdateiformate durch das zu erstellende Pseudonymisierungstool verarbeitet werden können. Im Anschluss daran werden in Kapitel 4 ein Tool zur Erstellung einer Policy in der definierten Policysprache entworfen und implementiert und in Kapitel 5 ein Pseudonymisierungstool, durch dass Daten anhand einer Policy aufbereitet werden können, angepasst und optimiert. In Kapitel 6 werden die implementierten Komponenten dieser Arbeit funktional und bzgl. der Laufzeit und dem Speicheraufwand überprüft. Im 7. und vorletzten Kapitel wird das erstellte Pseudonymisierungstool in den aktuellen Stand der Forschung eingeordnet. Abschließend wird in Kapitel 8 die gesamte Arbeit zusammengefasst und ein Ausblick auf mögliche weitere Arbeiten gegeben.

2 Grundlagen

In diesem Kapitel werden grundlegende Begriffe und Mechanismen erklärt und definiert, die in dieser Arbeit verwendet werden. Dazu gehören der Begriff "Personenbezogene Daten", Pseudonymisierung, die einzelnen Verfügbarkeitsoptionen und die Arten der Bindungen von Verfügbarkeitsoptionen.

2.1 Personenbezogene Daten

> **Definition 1 (Personenbezogene Daten):** "Im Sinne dieser Verordnung bezeichnet der Ausdruck: „personenbezogene Daten" alle Informationen, die sich auf eine identifizierte oder identifizierbare natürliche Person (im Folgenden „betroffene Person") beziehen; als identifizierbar wird eine natürliche Person angesehen, die direkt oder indirekt, insbesondere mittels Zuordnung zu einer Kennung wie einem Namen, zu einer Kennnummer, zu Standortdaten, zu einer Online-Kennung oder zu einem oder mehreren besonderen Merkmalen, die Ausdruck der physischen, physiologischen, genetischen, psychischen, wirtschaftlichen, kulturellen oder sozialen Identität dieser natürlichen Person sind, identifiziert werden kann;" ((EU) 2016/679) [8, Art. 4 Abs. 1]

Nach der europäischen Datenschutzgrundverordnung [8] sind personenbezogene Daten alle Informationen, die eine natürliche Person charakterisieren bzw. mit der Person im Zusammenhang stehen und so gespeichert sind, dass sie der Person zuordenbar sind. Diese Zuordnung kann durch eine Identifikationsnummer oder durch andere identifizierende Elemente geschehen.

2.2 Datensammlung

In dieser Arbeit werden Daten betrachtet, die allgemein in einer Datensammlung enthalten sind. Eine Datensammlung D besteht dabei aus Datensätzen d_i, die wiederum aus Datenfeldern f_k bestehen. Die Datenfelder können Namen besitzen und beinhalten Daten bzw. im Einzelnen jeweils ein Datum. Bei den in dieser Arbeit betrachteten Datensammlungen handelt es sich um kleine Datensammlungen. Das bedeutet, dass eine Datensammlung grundsätzlich in einer Datei enthalten ist und somit im ganzen in den Arbeitsspeicher geladen wird. Jedoch ist die Aufteilung einer Datensammlung auf mehrere Dateien möglich und kann entsprechend adressiert werden. [9, S. xx ff.]

2.3 Pseudonymisierung

Definition 2 (Pseudonymisierung): Das Ersetzen eines Klartextes mit Identifikationsmerkmalen durch ein Pseudonym, durch das die Bestimmung des Klartextes ausgeschlossen oder zumindest erheblich erschwert ist. Einzelne Identifikationsmerkmale können dabei bestehen bleiben.

Die Pseudonymisierung, die sich allgemein auf digitale Daten bezieht, ist zu unterscheiden von der Pseudonymisierung zum Zweck eine Person nicht mehr bestimmen zu können, definiert in §3 des Bundesdatenschutzgesetzes Abs. 6a [10] und in [11]. Eine einfache Methode um ein Datum zu pseudonymisieren, ist dessen Ersetzung durch einen Zufallswert. Im Fall, dass das gleiche Datum mehrfach vorkommt, müssen diese Daten durch das gleiche Pseudonym ersetzt werden, damit die Zuordnung zu einem Pseudonym noch bestehen bleibt. Wenn aber nicht nur der Schutz der Daten, sondern auch die Nutzbarkeit dieser gefordert ist, müssen andere Methoden herangezogen werden. Dabei ist nicht nur zu beachten, welche Aspekte der Daten benötigt werden, sondern auch wer Zugriff auf diese erhalten darf.

2.4 Verfügbarkeitsoptionen

Wie schon in der Einleitung beschrieben, ist es oft erforderlich, dass Daten nur für bestimmte und nicht alle Zwecke genutzt werden können. Dementsprechend sollen sie nur einzelne Optionen anbieten, wodurch nur einzelne Aspekte der Daten betrachtet werden können. Diese Optionen nennt man Verfügbarkeitsoptionen. Durch eine Verfügbarkeitsoption können andere Aspekte eines Werts diesem nicht mehr zugeordnet werden. Zu diesen Optionen gehören die Aufdeckbarkeit, die Verkettbarkeit und die Möglichkeit bestimmte mathematische Operationen durchzuführen. Auf welche Art und Weise und mit welchen Methoden oder Verfahren diese Verfügbarkeitsoptionen praktisch umgesetzt werden, kann unabhängig entschieden werden. Dadurch kann eine praktische Umsetzung an die Verwendung und an die benötigte Sicherheit angepasst werden.

Definition 3 (Verfügbarkeitsoption): Option die ein Pseudonym anbieten kann, durch die nur ein Aspekt des Klartextes betrachtet bzw. analysiert werden kann.

2.4.1 Aufdeckbarkeit

Definition 4 (Aufdeckbarkeit): Gegeben ein Pseudonym P_{O_1}. Dann ist die Aufdeckbarkeit die Möglichkeit den Klartext O_1 vom Pseudonym P_{O_1} anhand einer Zuordnungsvorschrift f zu offenbaren.

$$f(P_{O_1}) = O_1$$

Die Option der Aufdeckbarkeit ist direkt an eine Bedingung gebunden, durch die der Zugriff eingeschränkt wird. Wenn keine Bedingung existiert, kann jeder auf den Klartext zugreifen und es gibt kein Pseudonym. Zu diesen Bedingungen gehören die Rollenbindung und die Zweckbindung, die in Kapitel 2.5 besprochen werden.

2.4.2 Verkettbarkeit

Definition 5 (Verkettbarkeit): Gegeben zwei Pseudonyme. Dann ist die Verkettbarkeit die Möglichkeit anhand von zwei Pseudonymen P_{O_1} und P_{O_2} zu erkennen, ob die zugehörigen Klartexte O_1 und O_2 in einer Beziehung stehen bezüglich einer Relation.

Ein Beispiel für eine Relation ist die Gleichheitsrelation, durch die anhand der Pseudonyme erkannt werden kann, ob die Klartexte dieser gleich sind. Eine Möglichkeit ist das Anwenden einer deterministischen Hashfunktion. Gegeben das Szenario, dass Datensätze einer Datensammlung jeweils eine IP-Adresse und andere personenbezogene Daten enthalten und dass diese durch die IP-Adresse mit einer Person in Verbindung gebracht werden können. Wenn die IP-Adresse durch ihren Hashwert ersetzt wird, kann in einer Datensammlung nur noch überprüft werden, wie oft eine IP-Adresse vorkommt. Eine Verbindung zur Person kann durch einen Hashwert nicht mehr hergestellt werden. Eine weitere Relation die Teilerhaltung, durch die Teile eines Klartextes bei der Pseudonymisierung erhalten bleiben.

2.4.3 Mathematische Operation

Definition 6 (Mathematische Operation): Gegeben zwei Pseudonyme P_{O_1} und P_{O_2}. Dann ist die mathematische Operation die Möglichkeit mit diesen Pseudonymen eine mathematische Operation $\bullet$ durchzuführen, die äquivalent zu einer mathematischen Operation * auf den Klartexten O_1 und O_2 ist, sodass das Ergebnis das Pseudonym des Ergebnisses der Klartexte ist.

$$P_{O_1} \bullet P_{O_2} = P_{O_1 * O_2}$$

Die Operation Addition kann z.B. durch einen Homomorphismus realisiert werden. Die addierte Pseudonyme ergeben dabei das Pseudonym der Summe der Klartexte. Ein Beispielszenario ist eine politischen Abstimmung mit den Antwortmöglichkeiten Ja($\hat{=}$ 1) und Nein($\hat{=}$ 0). Es werden nur die Pseudonyme der Stimmen gespeichert und am Ende addiert. Das Ergebnis ist die entschlüsselte Summe der Pseudonyme.

2.5 Bindung einer Verfügbarkeitsoption

Damit die in Abschnitt 2.4 beschriebenen Verfügbarkeitsoptionen nicht von jedem genutzt werden können und die Daten stärker geschützt sind, sodass nur die Zugriff haben, die es dürfen oder müssen, werden in diesem Kapitel Bindungen von Verfügbarkeitsoptionen beschrieben. Dazu gehören die Rollenbindung und die Zweckbindung, die sich wiederum in zwei Teile gliedert. Eine Verfügbarkeitsoption kann an eine oder mehrere Bindungen nacheinander gebunden sein. Für diese Bindungen gilt die Unabhängigkeit zur konkreten Umsetzung im gleichen Maße, wie bei den Verfügbarkeitsoptionen beschrieben.

2.5.1 Rollenbindung

Definition 7 (Rollenbindung): Bindung der Verfügbarkeit eines Datums und ggf. weiterer zugehöriger Daten (z.B. Schlüssel) an eine Rolle, sodass nur diese Zugriff auf dieses Datum hat.

Durch die Rollenbindung kann ein Pseudonym, dass eine Verfügbarkeitsoption anbietet, an eine Rolle gebunden werden. Ein Beispiel ist die Aufdeckbarkeit eines Namens ausschließlich durch einen Verantwortlichen.
Beispiel: Gegeben das Verschlüsselungsverfahren AES mit den Operationen $E_k(x)$ und $D_k(x)$ zum Ver- und Entschlüsseln eines Klartextes x durch einen Schlüssel k.

1. Aufbereitung: E_k(Name) = IG2OEP/TlNYBAxteKU8t6sv... = Pseudonym
2. Nutzung der Aufdeckbarkeit: D_k(Pseudonym) = Name

Wenn dazu die Bedingung gehört, dass nur ein Verantwortlicher das Pseudonym aufdecken darf, um den Namen zu erfahren, ist der Schlüssel k, der bei der Entschlüsselung genutzt wird, das Geheimnis, dass nur der Verantwortliche kennt. Der Verantwortliche ist in diesem Fall die Rolle. Zum Beispiel kann dadurch eine Person ihre eigenen Daten entschlüsseln.

2.5.2 Zweckbindung

Definition 8 (Zweckbindung): Bindung der Verfügbarkeit eines Datums und ggf. weiterer zugehöriger Daten (z.B. Schlüssel) an einen Zweck, sodass nur bei Zutreffen des Zweckes der Zugriff auf dieses Datum möglich ist.

Wenn man nun das Beispiel von der Rollenbindung übernimmt, kann es außerdem sein, dass das Aufdecken des Namens vom Verantwortlichen nur unter bestimmten Umständen bzw. zur Erfüllung eines bestimmten Zwecks durchgeführt werden darf. Dies liegt zum Beispiel dann vor, wenn ein Datensatz, in dem der Name enthalten ist, mit einer Straftat in Verbindung gebracht wird und somit der zur Aufdeckung erfüllende Zweck gegeben ist.
Dabei ist durch die Art der Messung, ob eine bestimmte Situation bzw. ein bestimmter Zweck eingetreten ist, in organisatorische und technische Zweckbindung zu unterscheiden. [12]

Organisatorische Zweckbindung. Wenn die organisatorische Zweckbindung gegeben ist, gibt es eine Partei, die vom Datenherausgeber autorisiert wurde das Pseudonym nur dann aufzudecken wenn ein bestimmter Zweck eingetreten ist. Deshalb ist es wichtig, dass die Partei die Situation richtig bewertet und korrekt einschätzt, ob der festgelegte Zweck vorhanden ist. Aus diesen Gründen ist diese Partei eine vertrauenswürdige dritte Partei, der der Datenherausgeber vertraut. Der Nachteil dabei ist, dass durch die benötigte menschliche Interaktion der Aufwand umso größer ist. [12][13, p. 49]

Technische Zweckbindung. Die technische Zweckbindung geht mit dem Vorteil, dass die benötigte Interaktion automatisiert werden kann, gegenüber der organisatorischen Zweckbindung einher. Dabei gibt es einen Mechanismus, der erkennt, ob ein bestimmter Zweck eingetreten ist. Dieser zu erkennende Zustand muss dementsprechend klar definiert sein. Dabei existieren die drei Arten: Wert-basiert (engl. value-based), Zeit-basiert (engl. time-based) und Vorkommensbasiert (engl. occurrence-based).
Ein wert-basierter Zweck ist dabei an eine Variable gebunden. Wenn diese Variable einen festgelegten Wert enthält, wird das Pseudonym aufgedeckt. Solange die Variable zum Beispiel eine Zahl größer als eine vordefinierte Zahl ist, wird das Pseudonym nicht aufgedeckt. Bei einem zeit-basierten Zweck ist das Pseudonym nur in einem bestimmten Zeitfenster aufdeckbar.
Als Letztes kommt es bei einem vorkommensbasiertem Zweck darauf an, wie oft Daten, die einen festgelegten Inhalt enthalten, vorkommen müssen, bis das Pseudonym aufdeckbar ist. Wenn zum Beispiel nacheinander die Hälften des Passworts des verschlüsselten Pseudonyms veröffentlicht werden, muss dieses Ereignis genau zweimal auftreten, um das Pseudonym aufzudecken. [12]
Alle beschriebenen Bindungen können auch kombiniert werden.

3 XML basierte Policy Sprache

Um die Verfügbarkeitsoptionen und Bindungen von Verfügbarkeitsoptionen für kleine Datensammlungen anzubieten bzw. diese anhand derer aufzubereiten, wird eine Policy (zu deutsch ein Regelwerk) benötigt. In einer Policy stehen jeweils alle Regeln zur Behandlung der einzelnen Daten eines Datensatzes. Eine Regel besteht in diesem Fall aus einer Verfügbarkeitsoption und ein bis zwei Bindungen von Verfügbarkeitsoptionen. Dies wird im Folgenden genauer erörtert.

Um Daten anhand einer Policy aufbereiten zu können, benötigt es eine strukturierte und maschinenlesbare Policy Sprache. Die Extensible Markup Language, abgekürzt XML, [14] erfüllt diese Anforderungen, da sie eine hierarchisch strukturierte textbasierte Sprache ist, die es erlaubt eine eigene auf ihr basierende Sprache zu definieren. Aus diesen Gründen wird sie in dieser Arbeit verwendet.

An die Erstellung einer eigenen XML-basierten Policy Sprache sind im Rahmen dieser Arbeit mehrere Anforderungen aufgekommen.
Als **erstes** soll die Sprache nutzbar für das in [5] vorgestellte Pseudonymization Tool sein, dass in dieser Arbeit weiterverwendet werden soll. Das Pseudonymization Tool soll eine erstellte Policy verarbeiten können und anhand derer semi-strukturierte Daten dementsprechend aufbereiten. Die in [5] genutzte Policy Sprache wird in Kapitel 3.2 erörtert.
Die **zweite** Anforderung ist, dass die Sprache eine klare Struktur ohne überflüssige oder sich wiederholende Zusatzinformationen besitzt. Überflüssig bedeutet dabei, dass die Elemente der Sprache weder die Lesbarkeit, noch das Verständnis oder vor allem die Verarbeitung erschweren und keinen Vorteil (oder Nutzen) in der Definition einer Policy bringen.
Als **Drittes** sollen die für eine Policy nutzbaren Verfügbarkeitsoptionen so fein definiert werden können, dass ein Analyst (bei einer Analyse) nur die Informationen bekommen kann, die er bekommen darf. Dementsprechend soll er keine weiteren Informationen aus Zusammenhängen schließen dürfen, die nicht für ihn bestimmt sind. Damit soll auf das Prinzip der Datensparsamkeit bzw. der Erforderlichkeit [15] bei der Erstellung einer Policy geachtet werden können.
Viertens sollen dagegen die Verfügbarkeitsoptionen auch so in der Policy angegeben werden können, dass sie genau dann, wenn es erlaubt ist, dem Analyst die Informationen geben, die er durch die jeweilige Verfügbarkeitsoption erhalten soll. Wenn z.B. nur ein Wert mit der Verfügbarkeitsoption Verkettbarkeit bezüglich der Relation Gleichheit pseudonymisiert wird, ist dieses Pseudonym für einen Analysten wertlos. Es benötigt mindestens zwei Pseudonyme, damit der Analyst diese Verfügbarkeitsoption nutzen kann.
Die **fünfte** Anforderung ist, dass die Sprache für viele Datensätze bzw. Formate von Datensätzen hinsichtlich ihrer Struktur nutzbar sein soll. Egal ob es sich dabei um eine Logdatei mit zeilenweisen Einträgen, eine Textdatei mit kommagetrennten Werten (engl. Comma-separated values (CSV)) oder anwendungsspezifische Daten handelt.
Als **Sechstes** soll die zu erstellende Sprache erweiterbar sein. Genauer bedeutet das für Elemente der Sprache, dass andere Verfügbarkeitsoptionen und Bindungen von Verfügbarkeitsoptionen als die, die hier vorgestellt werden, hinzugefügt werden können. Hinsichtlich der Struktur soll

diese so aufgebaut sein, dass z.B. auch eine Teilmenge von Datensätzen gleich behandelt werden könnte.
Die siebte und letzte Anforderung ist, dass als Standard alle Daten vertraulich behandelt werden sollen. Für Daten, die nicht in der Policy referenziert werden, darf kein Pseudonym erstellt werden. Die sieben oben erläuterten Anforderungen sind in Tabelle 1 zusammengefasst aufgelistet. In den folgenden Kapiteln wird auf diese Anforderungen bei der weiteren Untersuchung, wie die zu erstellende Policy Sprache aussehen soll, verwiesen.

Anforderungen:

1. Nutzbar für Pseudonymization Tool [5].
2. Klare und lesbare Struktur ohne überflüssige Zusatzinformationen.
3. Möglichkeit der Angabe der Verfügbarkeitsoptionen, sodass keine zusätzlichen ungewollten Informationen bei einer Analyse von verarbeiteten Daten gewonnen werden können.
4. Möglichkeit der Angabe von Verfügbarkeitsoptionen, sodass diese für einen Analyst bei einer Analyse wertvoll sind bzw. er dabei Informationen gewinnen kann.
5. Nutzbar für verschiedene Arten von Datensätzen hinsichtlich ihrer Strukturen.
6. Erweiterbarkeit.
7. Vertraulichkeit als Standard.

Tabelle 1.: *Die Anforderungen für die zu erstellende XML basierte Policy Sprache.*

3.1 Formate von Datensätzen

Da, wie in der fünften Anforderung (Tabelle 1) beschrieben, die Policy Sprache für unterschiedliche Datensätze genutzt werden soll, werden im Folgenden mehrere Formate von Datensätzen vorgestellt. Diese werden dabei auf ihre Struktur untersucht und können anwendungsspezifisch sein.
Um Daten zu halten oder auszutauschen gibt es mehrere Formate bzw. Sprachen und Standards. Da das zu erstellende Framework dieser Arbeit nur Datensätze, die als Medientyp Text bzw. in Textdateien vorliegen, verarbeitet, werden im Folgenden keine strukturierten Datenhaltungen wie Datenbanken betrachtet.

3.1.1 Allgemeine Formate und Sprachen

Unter den allgemeinen Formaten und Sprachen zur Datenhaltung werden einfach strukturierte Daten, wie Dateien im CSV-Dateiformat und semi-strukturierte Daten, wie die Auszeichnungssprachen XML und YAML, betrachtet. Diese sind verbreitete Formate zur Haltung von Daten und werden z.B. im Bereich E-Commerce genutzt.

Einfach strukturierte Daten: CSV-Dateiformat. Das CSV-Dateiformat wird zur Haltung von Daten und zum Austausch dieser zwischen unterschiedlichen Programmen genutzt. Die Abkürzung CSV steht für Comma-Separated Values (engl. für Komma-getrennte Werte). Daten werden dabei, wie der Name sagt, kommagetrennt in einer Textdatei gespeichert. Eine Zeile beschreibt dabei einen Datensatz und die Werte zwischen den Kommas einer Zeile sind die Daten eines Datensatzes. Die Anzahl an Felder für Werte soll immer gleich der maximalen Anzahl an Feldern

der CSV-Datei sein. Die erste Zeile kann eine Kopfzeile sein, die die Namen der Spalten definiert. Folgend ein Beispiel, dass die beschriebenen Eigenschaften erfüllt. [16]

Beispiel:

```
Vorname,Nachname,E-Mail,Unternehmen,Ausweisnummer
Max,Mustermann,semper.video@mail.de,Blue Dynamic,122000129
Erika,Mustermann,ipsum@nonce.de,Boston Dynamics,T220001293
```

Die Trennung der Daten kann auch durch andere Zeichen geschehen. Die Benutzerdatei `/etc/passwd` von UNIX-Systemen nutzt das CSV-Dateiformat. Dabei werden die Daten durch einen Doppelpunkt getrennt. [17]
Die Struktur einer Datei im CSV-Format besteht somit immer aus Zeilen und Spalten und ist vergleichbar mit einer zweidimensionalen Tabelle.

Semi-strukturierte Daten: XML, YAML. XML- und YAML-Daten können dagegen komplexer strukturiert sein. Sie sind beide Auszeichnungssprachen, semi-strukturiert und hierarchisch aufgebaut. Semi-strukturiert bedeutet dabei, dass es kein Datenschema gibt, das besagt, wie die Struktur der Datensätze aussieht. Die Datensätze enthalten deshalb selbst die Struktur. Dies wird selbsterklärende Daten genannt. Die Struktur kann daher unregelmäßig sein. [18, 19]
Ein durch die Extensible Markup Language (XML) definiertes Dokument besteht immer aus einem Wurzelelement, das weitere Elemente enthalten kann. Jedes Element kann beliebig viele Elemente enthalten. Elemente können Attribute und einen Text besitzen. [14]

Strukturbeispiel:

```
<root>
    <element>
        <element/>
    </element>
    <element attr="1">text</element>
</root>
```

Das obige Beispiel von einer Datensammlung im CSV-Format könnte in XML wie folgt aussehen:

```
<Personenbezogene_Daten>
    <Person>
        <Vorname>Max</Vorname>
        <Nachname>Mustermann</Nachname>
        <E-Mail>semper.video@mail.de</E-Mail>
        <Unternehmen>Blue Dynamic</Unternehmen>
        <Ausweisnummer>122000129</Ausweisnummer>
    </Person>
    <Person>
        <Vorname>Erika</Vorname>
        <Nachname>Mustermann</Nachname>
        <E-Mail>ipsum@nonce.de</E-Mail>
        <Unternehmen>Boston Dynamics</Unternehmen>
        <Ausweisnummer>T220001293</Ausweisnummer>
    </Person>
</Personenbezogene_Daten>
```

Ein Yaml-Dokument hat eine ähnliche hierarchische Struktur. Bei der Entwicklung von YAML wurde zusätzlich Wert auf die Menschenlesbarkeit gelegt. Es gibt keine schließenden End-Elemente und Attribute. Elemente, die in einer Ebene mehrfach vorkommen, werden durch

einen Bindestrich wie in einer Liste untereinander aufgelistet. Für die komplette Spezifikation siehe [20].
Zusammenfassend sind beide Sprachen wesentlich komplexer im Vergleich zum CSV-Format und können weiter untergliederte Daten, wie die Sehstärke einer Person, enthalten. In diesem Fall gäbe es jeweils ein Unterelement für jedes Auge bzw. dessen Sehstärke. Dies wäre gleichzusetzen mit einer mehr als zweidimensionalen nicht notwendigerweise vollständigen Tabelle.

3.1.2 Standards zur Haltung und zum Austausch von personenbezogenen Daten

Neben den allgemeinen Strukturen zur Datenhaltung gibt es anwendungsspezifische Strukturen. Da es in dieser Arbeit vor allem um die Pseudonymisierung von personenbezogenen Daten geht, wurde nach Standards zur Haltung und zum Austausch von Daten recherchiert. Unter anderem aus dem Grund, dass personenbezogene Daten sehr vertraulich behandelt werden und es somit schwer ist solche Daten zur Analyse der Struktur zu erlangen, wurden Standards zum Austausch und zur Haltung von Gesundheitsdaten in einer elektronischen Gesundheitsakte (im Englischen u.a. electronic health record (EHR)) betrachtet.
Eine elektronischen Gesundheitsakte ist eine Kollektion aller medizinischen Daten einer Person. Neben Laborwerten gehören unter anderem Berichte über Arztbesuche, eingenommene Medikamente und andere Daten zur Krankengeschichte der Person dazu. [21, S.188-199]
Um diese Daten einheitlich zu organisieren, wurden Standards entwickelt. Mehrere und auch weit verbreitete sind der der Organisation Health Level 7 (HL7). Die HL7 V2.x Serie ist ein Standard um Nachrichten mit Gesundheitsdaten auszutauschen. Die neue Version 3 (HL7 V3) ist für das gesamte Gesundheitswesen nutzbar, enthält für medizinische standardisierte Dokumente die Clinical Document Architecture (CDA) und ein Referenzinformationsmodell (RIM), das alle Arten von Informationen zur Gesundheit widerspiegelt. [22, 23, 24]
Um personenbezogene Gesundheitsdaten zu speichern kann das RIM, das ein konzeptionelles und objektorientiertes Modell ist und welches in der Unified Modeling Language (UML) beschrieben ist, genutzt werden. Von Li et al. [22] wurde untersucht, wie dies in ein relationales Datenmodell für eine Datenbank umgewandelt werden kann.
Um die Gesundheitsdaten in solch einem System für diese Arbeit zu nutzen, können die von HL7 standardisierten Nachrichten oder die CDA-Dokumente genutzt werden. Diese sind beide für einen Austausch von Informationen gedacht. [25, S.132-137]

HL7 V2.x und V3. Um Nachrichten zu versenden gibt es die Versionen 2 und 3. Eine Nachricht der Version 2 kann in zwei Arten von Syntaxen vorliegen. Bis zur Version 2.3.1 gab es nur eine klassische textbasierte in Zeilen unterteilte Syntax (auch ER7 genannt), die dem CSV-Format ähnelt. Dabei besteht jede Zeile aus einem Segment und jedes Segment aus Feldern, die wiederum weiter unterteilt sein können. Die erste Zeile einer Nachricht enthält das Kopfsegment (den Header), das die Art der Nachricht definiert. Das erste Feld eines Segments bestimmt den Typ des Segments. Ein Beispiel ist der Typ PID (Patient Identification), der die Daten einer Person wie den Namen, das Geschlecht oder das Geburtsdatum enthält. Das folgende Beispiel enthält die Daten einer Untersuchung für die Patientin Eve Everywoman aus Statesville.

Beispiel:

```
1 MSH|^~\&|GHH LAB|ELAB-3|GHH OE|BLDG4|200202150930||ORU^R01|CNTRL-3456|P|2.4
2 PID|||555-44-4444||EVERYWOMAN^EVE^E^^^^L|JONES|196203520|F||
  |153 FERNWOOD DR.^^STATESVILLE^OH^35292||(206)3345232
  |(206)752-121||||AC555444444||67-A4335^OH^20030520
3 OBR|1|8  ...
```

Ab der Version 2.3.1 gibt es eine zweite Syntax, die die Extensible Markup Language (XML) nutzt und die Segmente und Felder der klassischen Syntax widerspiegelt. Die Nachrichten der Version 3 kommen ausschließlich im XML-Format vor. In 90% der Krankenhäuser wird aber noch die Version 2 eingesetzt. [26, 27] [25, S.132-137]

Clinical Document Architecture (CDA). Neben den Nachrichten, die im Einzelnen nur Teilinformationen über eine Person enthalten, gibt es den Austausch von Informationen auch als Dokument durch die Clinical Document Architecture (CDA). Das Ziel dieses Standards ist es Dokumente bereitzustellen, die sowohl von einem Menschen lesbar sind, als auch durch eine Maschine verarbeitet werden können. Um dies zu gewährleisten basieren diese auf XML und können durch die Nutzung von XML-Stylesheets menschenlesbar angezeigt werden. Ein Dokument besteht aus einem Header, in dem der Kontext, der aus den Ereignissen, den Patientendaten und dem beteiligten medizinischen Personal besteht, definiert ist, und dem Body, der die Informationen zum Kontext enthält. Dabei gibt es drei Level, die darüber aussagen, wie hoch der Grad der Struktur und damit der Grad der Maschinenverarbeitbarkeit ist. Ein Beispieldokument sowohl in menschenlesbarer als auch maschinenverarbeitbarer Form kann in [28] betrachtet werden. [21, S.261-265][28]
Zusammenfassend ist die XML-Struktur der Daten genauso wie bei einer HL7 Nachricht mehrfach verschachtelt. Die Daten liegen nicht nur im Textteil sondern auch als Attribute eines XML-Elements vor.

3.2 Tag-basierte vs. Individuelle Behandlung

Wenn Daten anhand einer Policy aufbereitet werden, gibt es unterschiedliche Ansätze. Die hier genutzten vorausgehenden Arbeiten [4, 5] weisen zwei Arten auf.

In "Using XACML for the Definition of Availability Policies for Data Pseudonymization" [4] wird die eXtensible Access Control Markup Language (XACML) genutzt um Policies zur Pseudonymisierung zu erstellen. Diese Sprache basiert auf der Extensible Markup Language (XML) und ist eine Zugriffskontrollsprache. Die dabei definierten **`Rule`**s (engl. für Regel) enthalten eine Beschreibung des **`Target`**s (engl. für Ziel) und der **`Condition`**s (engl. für Bedingung), die zutreffen müssen, damit ein Zugriff erlaubt wird. Für die genaue Definition der **`Rule`** enthalten diese verschiedene Attribute. Im Folgenden eine Auflistung von Attributen, die in [4] genutzt worden sind:

- **`resource`**: Die Quelle auf die zugegriffen werden möchte bzw. das Datenfeld, das pseudonymisiert werden soll.
- **`mode`**: Die Verfügbarkeitsoption, die beim Pseudonymisieren genutzt werden soll.
- **`action`**: Die Aktion, die beim Zugriff durchgeführt werden soll. Ein möglicher Zugriff ist z.B. ein Lesezugriff.
- **`subject`**: Das Subjekt, das auf die **`resource`** zugreifen möchte. Dadurch entsteht eine Art Rollenbindung.

Jede **`Rule`** definiert demnach immer welche Datenfelder welche Verfügbarkeitsoption anbieten soll. Wenn zum Beispiel Datensätze mit personenbezogenen Daten pseudonymisiert werden sollen und in der Policy für das Datenfeld *Nachname* die Verfügbarkeitsoption Aufdeckbarkeit und die Bindung an eine Rolle gegeben ist, werden alle Pseudonyme diese Verfügbarkeitsoption für die angegebene Rolle anbieten.
Alle Datenfelder der Datensätze mit dem gleichen Bezeichner bzw. Tag (engl. für Etikett) bieten jeweils die gleichen Verfügbarkeitsoptionen und Bindungen von Pseudonymen an. Deshalb

handelt es sich bei solch einer Policy um eine tag-basierte Policy. Dies kann Vor- und Nachteile haben. Im folgenden werden alle derartigen Behandlungen als tag-basiert bezeichnet.

In "Framework Implementation: Availability Policies for Data Pseudonymization" [5] ist eine Policy dagegen anders aufgebaut. Sie nutzt die Extensible Markup Language (XML). Eine Policy besteht dabei immer aus jeweils einem Eintrag für einen Datensatz. Jeder Eintrag enthält wiederum die Datenfelder mit den jeweiligen Behandlungen (im engl. treatment) eines Datenfeldes.

```
<policy>
    <Datensatz id="0">
        <Datenfeldbezeichner>
            <treatment> ... </treatment>
            <treatment> ...
        </Datenfeldbezeichner>
    </Datensatz>
    <Datensatz id="1">
            ⋮
</policy>
```

Wenn das Beispiel von oben hierauf angewendet wird, können für jedes einzelne Datenfeld mit dem Bezeichner *Nachname* Verfügbarkeitsoptionen und Bindungen von Verfügbarkeitsoptionen definiert werden. Die Datenfelder können bei einer Pseudonymisierung individuell behandelt werden. Dies kann wie oben bereits zu [4] genannt Vor- und Nachteile haben.

Vor- und Nachteile. Bei einer tag-basierten Policy kann eine Verfügbarkeitsoption für ein Datenfeld x und damit für jedes Datenfeld x in den Datensätzen angegeben werden. Jemand der anhand solch einer Policy pseudonymisierte Datensätze erhält, kann alle Daten des Datenfeldes x unter Nutzung der Verfügbarkeitsoption analysieren. Dies ist insbesondere bei den Verfügbarkeitsoptionen Verkettbarkeit und Mathematische Operationen sinnvoll. Wenn z.B. das Datenfeld x die Verfügbarkeitsoption Verkettbarkeit in Bezug zur Relation Gleichheit anbietet, kann diese Person bzw. ein Analyst die Daten der Datenfelder x auf sich wiederholende Daten analysieren und dabei übergreifende Informationen aus allen Datensätzen gewinnen. Dies unterstützt die vierte Anforderung der Informationsgewinnbarkeit, aber kann gleichzeitig gegen die dritte Anforderung sprechen (Tabelle 1), da einzelne Werte gegebenenfalls geheimgehalten werden sollen und deshalb nicht in eine Analyse mit eingehen dürfen. Zudem könnten die gewonnenen Informationen des Analysten zu detailliert sein und damit wieder Bezüge zu den Klartexten herstellen. Wenn ein Analyst z.B. von einer Person weiß, dass sie bzw. personenbezogene Daten von ihr in einer großen Menge von Datensätzen enthalten sind, aber er diese Person nicht genau zuordnen kann, da die Identifikationsmerkmale keine Verfügbarkeitsoptionen anbieten, kann er diese trotzdem zuordnen, wenn er genug weitere Informationen über diese Person besitzt und die weiteren Daten Verfügbarkeitsoptionen (bevorzugt die Aufdeckbarkeit) anbieten. Die Voraussetzung dafür ist, dass der Analyst die Datensätze in Äquivalenzklassen unterteilen kann, wodurch er sensible personenbezogene Daten dieser Person zuordnen könnte. Dies ist vergleichbar mit den Angriffen *homogeneity attack* und *background knowledge attack* [29], die bei einer k-Anonymität [30] Datensätze deanonymisieren können. Aus diesen Gründen ist von der Nutzung der Verfügbarkeitsoption Aufdeckbarkeit im Zusammenhang mit einer tag-basierte Policy abzuraten.
Als zweites ist eine tag-basierte Policy gut nutzbar für homogene Datensätze, die eine gleiche Struktur mit gleichen Datenfeldern und der Anzahl derer aufweisen. Dadurch ist der Speicheraufwand niedrig und wächst mit der Anzahl an Datenfelder. Zusätzlich ist die Erstellung

der Policy mit einem niedrigen Aufwand verbunden, da nur für die Anzahl an Datenfelder Verfügbarkeitsoptionen und Bindungen derer angegeben werden können.
Auf der anderen Seite hat eine individuelle Policy, bei der ein Datenfeld bzw. dessen Datum jeweils pro Datensatz individuell behandelt werden kann, den Vorteil, dass, wie in der dritten Anforderung (Tabelle 1) gefordert, nur die Pseudonyme der einzelnen Datensätze weitergegeben werden können. Dadurch werden bei einer Analyse nur die minimal benötigten Informationen preisgeben. Vor allem in Bezug zur Verfügbarkeitsoption Aufdeckbarkeit ist der Vorteil, dass nicht einer alle Daten eines Datenfeldes aufdecken kann, sondern nur z.B. deren Besitzer durch eine Rollenbindung ihre Daten aufdecken können. Dadurch wird bewusst die Analysierbarkeit der Datensätze eingeschränkt, was aber auch zu dem Nachteil führen kann, dass bei einer Analyse keine Informationen mehr gewonnen werden können. Insbesondere die Verkettbarkeit könnte aus diesem Grund wenige bis keine Informationen liefern, wenn es nur noch wenige Datenfelder gibt, die verglichen werden können.
Als zweites ist eine individuelle Policy im Gegenstück zu einer tag-basierten Policy insbesondere für heterogene Datensätze, die unterschiedliche Datenfelder pro Datensatz enthalten können, in der Hinsicht auf den Berechnungsaufwand bei der Pseudonymisierung besser geeignet. Der Grund dafür ist, dass dabei nicht jeweils jeder Datensatz auf die Existenz aller angegebenen Datenfelder überprüft werden muss.
Aus diesen Gründen haben beide Arten von Policies Vor- und Nachteile. Welche davon besser ist hängt von der jeweiligen Anwendung und deren Datensätzen ab.

Exkurs. Konzepte zu weiteren aber im Folgenden nicht betrachteten Arten, die während dieser Arbeit aufgekommen sind, sind als erstes die Angabe von Behandlungen, die nicht alle Datenfelder mit einem Bezeichner der Datensätze betreffen, sondern nur eine Teilmenge dieser. Als zweites könnte es auch eine Angabe einer Behandlung geben, die für mehr als ein Datenfeld gilt.

3.3 Aufbau der Policy Sprache

Das allgemeine Ziel ist es, dass die Policy Sprache vielseitig für alle Zwecke einsetzbar ist. Dementsprechend sollen die in Tabelle 1 spezifizierten Anforderungen bei der zu erstellenden Policy Sprache erfüllt sein. Wie bereits am Anfang dieses Kapitels erwähnt, soll für die Definition einer Policy die Extensible Markup Language (XML) genutzt werden.

Behandelbarkeit einzelner Klartexte. Um für einen Klartext eine Verfügbarkeitsoption und Bindungen der Verfügbarkeitsoption anzugeben, wird jedes Datenfeld, das jeweils einen Klartext enthält, mit jeweils einem XML-Element in Beziehung gesetzt. Wenn es zum Beispiel in einer Datensammlung einen oder mehrere Datenfelder mit einem Vornamen gibt, wird es in der Policy ein XML-Element mit dem Bezeichner *Vorname* geben. Die Subelemente dieses Elements beschreiben die Verfügbarkeitsoption und Bindungen der Verfügbarkeitsoptionen, die für diesen Wert erstellt werden sollen. Um mehrere Behandlungen (im Englischen *treatments*) für ein Datenfeld anzugeben, werden diese durch XML-Elemente mit dem Bezeichner *treatment* aufgelistet. In solch einem Element gibt es für eine Verfügbarkeitsoption ein XML-Element mit dem Bezeichner *availability_option* und für die Bindungen einer Verfügbarkeitsoption ein oder mehrere XML-Elemente mit dem Bezeichner *binding*. Wenn mehrere Bindungen für eine Verfügbarkeitsoption angegeben werden, ist auf die Reihenfolge dieser zu achten, da die Pseudonyme (und die ggf. zusätzlichen Daten) nach dieser Reihenfolge mit den Bindungen versehen werden. Die minimale Angabe einer Behandlung enthält immer eine Verfügbarkeitsoption und eine Bindung. Fehlerhafte Angaben in einer Policy sollen von einem Tool nicht umgesetzt werden. Je nach gewählter Option benötigt diese zusätzliche Angaben in der Policy. Diese müssen jeweils nach der jeweiligen Option angegeben werden, damit diese der Option zuordenbar sind. Detaillierter

wird dies im Abschnitt 3.3.1 behandelt. Durch diese Struktur ist die Policy Sprache flexibel und erweiterbar (Anforderung 6 (Tabelle 1)), da dadurch theoretisch beliebig viele unterschiedliche Verfügbarkeitsoptionen und Bindungen angegeben werden können. Auch die Menge der Verfügbarkeitsoptionen und Bindungen kann um neue Optionen erweitert werden. Es müssen auch nur Angaben zu Datenfeldern in einer Policy stehen, die benötigt werden (Anforderung 7 (Tabelle 1)).

Allgemein:

```
<Datenfeldbezeichner>
    <treatment>
        <availability_option> ... </availability_option>
        <binding> ... </binding>
          ... ggf. weitere Bindungen und zusätzliche Angaben
    </treatment>
          ⋮
</Datenfeldbezeichner>
```

Beispiel:

```
<Vorname>
    <treatment> ... </treatment>
</Vorname>
```

Art der Adressierung. Neben der Behandlung der einzelnen Datenfelder muss auch die Behandlung der Datenfelder im Gesamtzusammenhang auf der Ebene der Datensätze einer Datensammlung betrachtet werden. Da die zwei Arten tag-basierte und individuelle Behandlung, wie in Abschnitt 3.2 diskutiert, je nach Anwendung vorteilhaft sind und die Anforderungen 3 und 4 (Tabelle 1) unterstützen, werden beide in der Policy Sprache angeboten.

Um die Datenfelder individuell jeweils nach einem Datensatz zu behandeln bzw. die Datensätze individuell zu adressieren, gibt es für jeden Datensatz einer Datensammlung ein XML-Element. Der Bezeichner dieses Elements muss mit dem Namen des Datensatzes einer Datensammlung übereinstimmen. Da Datensätze in manchen Datenformaten keine Namen haben, ist in dem Fall der Name aller Datensätze gleich und entspricht einem festgelegten Standardnamen. Wie in [5] bereits aus dem Grund genutzt, dass Datensätze gleiche Namen haben können, gibt es um die einzelnen Datensätze individuell adressieren zu können ein XML-Attribut als Identifikator namens *id*, das die fortlaufende Nummer eines Datensatzes enthält. Dadurch können individuelle Behandlungen für ausgewählte Datensätze aus einer Datensammlung angegeben werden. Das bringt den Vorteil mit sich, dass erstens keine Fehler in der Adressierung entstehen können und zweitens nur Einträge der ausgewählten Datensätze angegeben werden müssen (Anforderung 2 (Tabelle 1)). Ein Angreifer könnte durch die Adressierung aber ggf. Zusatzinformationen bekommen. Die Angabe eines Datensatzes mit der Nummer x in einer Policy sieht wie folgt aus:

```
<Datensatzbezeichner id="x"> ... </Datensatzbezeichner> (Nur Teilstruktur)
```

Diese Behandlungsvorschrift einer Policy kann vom Pseudonymization Tool aus [5] verarbeitet werden (Anforderung 1 (Tabelle 1)), da es der dort genutzten Policy gleicht.

Um jedes Datenfeld, mit einem Bezeichner wie Name, Nachname oder Geburtsdatum, gleich zu behandeln, also tag-basiert, gibt es genau ein XML-Element, dass alle Datenfelder einer Datensammlung (durch entsprechende XML-Elemente) enthalten kann. Der Bezeichner dieses Elements ist gleich dem Bezeichner der Datensätze, wenn alle gleich sind, oder einem allgemeinen festgelegten Namen. Zur expliziten Adressierung besitzt dieses Element das XML-Attribut

id mit dem Wert *all*. Fortgeführt am Beispiel, dass alle Vornamen und Nachnamen gleiche Verfügbarkeitsoptionen und Bindungen der Verfügbarkeitsoptionen anbieten sollen, sieht die Repräsentation in XML wie folgt aus:

```
<Datensatzbezeichner id="all">
    <Vorname> ... </Vorname>
    <Nachname> ... </Nachname>
</Datensatzname>
```

Durch diese Struktur müssen in solch einem Fall die Behandlungen der Datenfelder nicht mit der Anzahl der Datensätze in einer Policy wiederholt werden. Dies unterstützt die Anforderung 2 (Tabelle 1).
Das Wurzelelement der gesamten XML-Struktur einer Policy enthält als Name den Bezeichner des Wurzelelements der Datensammlung. Als eindeutige Erkennung, dass es sich um eine Policy handelt, enthält das Wurzelelement das XML-Attribut *type* mit dem Wert *policy*.

Anwendbarkeit für verschiedene Formate von Datensätzen. Damit die Policy Sprache verschiedene Formate von Datensätzen unterstützt, wurde in Kapitel 3.1 eine Auswahl von verbreiteten Formaten, Sprachen und Standards von Datensätzen untersucht. Diese unterscheiden sich in der Struktur und sind hierarchisch mit in den meisten Fällen mehr als zwei Ebenen aufgebaut. Das Ziel der Policy Sprache ist, Policies für all diese und auch nicht betrachtete oder existierende Formaten von Datensätzen anbieten zu können (Anforderung 5 (Tabelle 1)). Dabei ist der Zweck jeder Policy für die Datenfelder der Datensätze einer Datensammlung Verfügbarkeitsoptionen und Bindungen der Verfügbarkeitsoptionen zu enthalten. Aus diesen Gründen ist die Struktur einer Policy einheitlich bzw. gleich gewählt. Der Vorteil ist, dass erstens die Struktur einer Policy nie komplexer wird. Das Gegenteil wäre der Fall, wenn die Struktur der Policy an die Struktur der Datensammlung bzw. der Datensätze angepasst wäre. Zweitens ist eine Policy deshalb auch für gleichartige Datensammlungen nutzbar, die in unterschiedlichen Formaten vorliegen, weil die Behandlungen unabhängig vom Format für Datenfelder angegeben werden. Ein Beispiel dafür sind HL7 Nachrichten ab der Version 2.3.1, die im klassischen Format oder im XML-Format vorliegen können. Damit diese unterschiedlich strukturierten Daten anhand einer einheitlich strukturierten Policy verarbeitet werden können, wird in Abschnitt 3.4 eine Methode zur Vereinheitlichung von Datensammlungen vorgestellt.

Möglichst einfacher Aufbau. Eine Policy beinhaltet die Behandlungsvorschriften der Datensätze einer Datensammlung bzw. von gleichartigen Datensammlungen. Dementsprechend besteht eine Policy, wie in Abbildung 1 zu sehen, aus einem Wurzel-Element, dass den Bezeichner der Datensammlung trägt und mehrere Datensätze beinhaltet. Dies sind entweder Datensätze mit dem fortlaufend nummerierten Attribut *id* für die individuelle Behandlung der Datensätze oder der Datensatz mit dem Attribut *id* und dem Wert *all* für die tag-basierte Behandlung der gesamten Datensätze. Diese enthalten wiederum und wie oben schon ausführlich beschrieben die Datenfelder der Datensätze. In der Abbildung 1 sind dies das Datenfeld *Nachname* für die individuelle Behandlung des Datensatzes mit `id=''0''` und für die tag-basierte Behandlung das Datenfeld *Vorname*, die auch im obigen Beispiel enthalten sind. Diese Angaben der Datenfelder bestehen jeweils aus ein oder mehreren Behandlungen. Jede Behandlung setzt sich schließlich aus der Verfügbarkeitsoption und den Bindungen der Verfügbarkeitsoption zusammen.
Durch diesen Aufbau kann einerseits für jedes Datenfeld eines Datensatzes individuell eine Behandlung und andererseits für alle Datenfelder der Datensätze mit jeweils dem gleichen Bezeichner eine Behandlung angegeben werden.

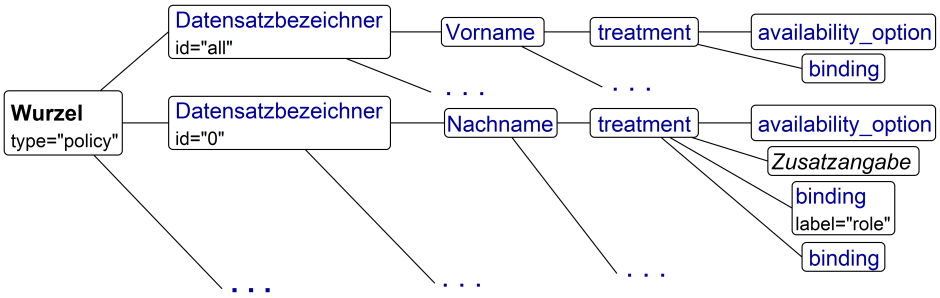


Abbildung 1.: *Allgemeine Struktur einer Policy mit den Datenfeldern Nachname für die tag-basierten Behandlung und Vorname für die individuellen Behandlung des Datensatzes mit der Nummer 0 als Beispiel.*

3.3.1 Verfügbarkeitsoptionen

Allgemein kann durch das Element mit dem Bezeichner *availability_option* jede Verfügbarkeitsoption angegeben werden. Als Konvention werden die Verfügbarkeitsoptionen auf Englisch und kleingeschrieben angegeben. Neben den in Kapitel 2.4 beschriebenen grundlegenden Verfügbarkeitsoptionen, werden die Verfügbarkeitsoptionen Verkettbarkeit und Mathematische Operation durch eine Relation oder Operation weiter unterteilt. Die in dieser Arbeit als Proof of Concept betrachteten Verfügbarkeitsoptionen stehen in der Tabelle 2. Um eine Verfügbarkeitsoption in einer Policy anzugeben, muss das Haupt-XML-Tag (a) und wenn vorhanden das Zusatz-XML-Tag (b) oder (c) verwendet werden. Diese XML-Tags sehen wie folgt aus:

(a) `<availability_option> </availability_option>`
(b) `<relation> </relation>`
(c) `<operation> </operation>`

Aufdeckbarkeit. Für die Verfügbarkeitsoption Aufdeckbarkeit muss nur das Haupt-Tag (a) mit dem Text *disclosability* angegeben werden.

Verkettbarkeit. Da bei der Verkettbarkeit auch eine Relation angegeben werden muss, muss neben dem Haupt-Tag (a) der Zusatz-Tag (b) verwendet werden. Der Haupt-Tag (a) enthält den Text *linkability* und im Zusatz-Tag (b) den Text der jeweiligen Relation hinter dem in der dritten Spalte der Tabelle 2 ein (b) steht. Dabei bezieht sich die Relation immer auf die Klartexte und beeinflusst das jeweils zugehörige Pseudonym. Die betrachteten Relationen sind erstens die Gleichheitsrelation. Gegeben zwei beliebige Klartexte aus der Menge der Klartexte, dann gilt, dass diese beiden Klartexte gleich sind, genau dann, wenn ihre zugehörigen Pseudonyme gleich sind. Dabei und im Folgenden gilt, um der Anforderung "Vertraulichkeit als Standard" gerecht zu werden, werden nur die in der Policy geforderten Informationen als Verfügbarkeitsoptionen in die aufbereiteten Datensätze eingefügt. Alle weiteren Klartextinformationen gehen verloren. Die zweite Relation ist die formaterhaltende Relation. Diese wurde für die Datentypen Integer, Decimal, E-Mail Adresse, IPv4 und IPv6 Adresse als Proof of Concept in dieser Arbeit eingeführt. Wenn diese Relation für einen Datentyp in einer Policy angegeben wird, kann anhand eines aufbereiteten Klartextes noch das Format bzw. die Struktur des Klartextes erkannt werden, da es sich bei dem aufbereiteten Klartext um einen Wert mit dem gleichen Datentyp handelt. Dazu gilt auch die Gleichheitsrelation. Durch gleiche Klartexte entstehen gleiche aufbereitete Klartexte. Zusätzlich zum Format gibt eine aufbereitete Zahl vom Datentyp Integer noch preis, wie viele Stellen die Klartextzahl hat und ob sie positiv oder negativ ist. Anhand einer aufbereiteten Dezimalzahl kann zusätzlich noch erkannt werden, wie viele Nachkommastellen die Klartextzahl hat. Aufbereitete Klartexte der Datentypen E-Mail Adresse, IPv4 und IPv6 Adresse weisen nur

noch die Struktur des jeweiligen Datentyps auf. Um dies zu erweitern wurde die Präfix- und Suffixerhaltende Relation für den Datentyp E-Mail Adresse hinzugenommen. Dadurch kann entweder der Teil vor dem "@"-Symbol, das Präfix, oder der Teil nach dem "@"-Symbol, das Suffix, einer E-Mail Adresse im aufbereiteten Klartext erhalten bleiben.

Mathematische Operation. Für die Verfügbarkeitsoption "Verfügbarkeit einer mathematischen Operation" muss der Haupt-XML-Tag (a) mit dem Text *mathematical operation* und der Zusatz-XML-Tag (c) mit dem Text der jeweiligen Operation, der in der Tabelle 2 ein nachstehendes (c) enthält, angegeben werden. Dabei gibt es als erstes die Operation bedingte Differenz, durch die bedingt die Differenz der Klartexte anhand der Pseudonyme bestimmt werden kann. Bedingt bedeutet dabei, dass ein Grenzwert der Differenz im Tool, das die Klartexte aufbereitet, festgelegt werden muss, bis zu dem die Differenz bestimmt werden kann. Als zweites wurde die Operation der Addition hinzugenommen, durch die zwei Pseudonyme addiert werden können und das Ergebnis dem Pseudonym des Ergebnisses der Addition der Klartexte entspricht.

Name	**Angabe in einer Policy**	**im Tag**
Aufdeckbarkeit	`disclosability`	(a)
Verkettbarkeit bzgl. der Relation:	`linkability`	(a)
Gleichheit	`equality`	(b)
Formaterhaltend (Datentyp Integer)	`formatpreserving integer`	(b)
Formaterhaltend (Datentyp Decimal)	`formatpreserving decimal`	(b)
Formaterhaltend (Datentyp E-Mail Adresse)	`formatpreserving email`	(b)
Präfixerhaltend (Datentyp E-Mail Adresse)	`prefix preserving email`	(b)
Suffixerhaltend (Datentyp E-Mail Adresse)	`suffix preserving email`	(b)
Formaterhaltend (Datentyp IPv4 Adresse)	`formatpreserving ipv4`	(b)
Formaterhaltend (Datentyp IPv6 Adresse)	`formatpreserving ipv6`	(b)
Mathematische Operation bzgl. der Operation:	`mathematical operation`	(a)
Bedingte Differenz	`conditional difference`	(c)
Addition	`addition`	(c)

Tabelle 2.: *Verfügbarkeitsoptionen und deren Angabe in einer Policy.*

Weitere Relationen und Operationen können durch die Festlegung neuer Angaben in einer Policy hinzugefügt werden.

3.3.2 Bindungen von Verfügbarkeitsoptionen

Für die Angabe einer Bindung einer Verfügbarkeitsoption an eine weitere Bedingung, wie die Rollen- oder Zweckbindung, wird wie schon beschrieben der XML-Tag

`<binding (e)>(d)</binding>`

verwendet. In der Tabelle 3 sind die in Kapitel 2.5 betrachteten Bindungen von Verfügbarkeitsoptionen aufgelistet. Die Angaben, markiert durch (d), werden als Text im XML-Element angegeben. Bei der Rollenbindung gibt es den Zusatz, der an der Stelle (e) gesetzt werden muss. Durch diesen wird der Name der Rolle, an den die Rollenbindung gebunden werden soll, angegeben. Für den Fall, dass keine Bindung für eine Verfügbarkeitsoption angegeben werden soll, muss an die Stelle (d) der Text *no binding* eingetragen werden. Wie bei den Verfügbarkeitsoptionen (Abschnitt 3.3.1) gilt als Konvention, dass die Notation der Bindungen von Verfügbarkeitsoptionen in der Policy auf Englisch und kleingeschrieben ist.

Name	Angabe in einer Policy	im Tag
Keine Bindung	`no binding`	(d)
Rollenbindung	`role binding` `label="`*role*`"`	(d) (e)
Zweckbindung		
Organisatorische Zweckbindung	`organizational purpose binding`	(d)
Technische Zweckbindung	`technical purpose binding`	(d)

Tabelle 3.: *Bindungen von Verfügbarkeitsoptionen und deren Angabe in einer Policy.*

Da eine Verfügbarkeitsoption und eine Bindung einer Verfügbarkeitsoption nicht ihre exakte Umsetzung in einem Tool beschreiben, dass Datensammlungen anhand einer Policy aufbereitet, kann es in solchen Programmen unterschiedliche Umsetzungen geben. Bei einer Analyse von aufbereiteten Datensammlungen muss dies später beachtet werden. Wenn weitere Verfügbarkeitsoptionen und Bindungen dieser festgelegt werden, muss auch darauf geachtet werden, ob ein Programm kompatibel mit diesen ist bzw. Methoden für Umsetzungen dieser enthält.

3.4 Transformation von Daten in eine einheitliche Struktur

Damit die einheitlich gewählte Policysprache für alle Formate von Datensätzen, wie unter anderem in Abschnitt 3.1 betrachtet, genutzt werden kann, wurde in dieser Arbeit die Transformation der Datensätze aus ihrer bisherigen Struktur in eine einheitliche Struktur als Lösung gewählt. Eine Transformation bedeutet für die Datensätze, dass sich ihre Struktur und ggf. das Format ändert, die enthaltenen Daten jedoch unverändert bleiben. Der Vorteil liegt darin, dass die Datensätze unabhängig von der jeweiligen Struktur verarbeitet werden können. Wenn ein Format von Datensätzen noch nicht unterstützt wird bzw. es keine Transformation gibt, muss nur eine Transformation in die einheitliche Struktur und das Format entwickelt werden. Dadurch kann ein hohes Maß an Flexibilität und gleichzeitig die Minimierung des Aufwandes für die Unterstützung eines anderen Formats einer Datensammlung gewährleistet werden.

Einheitliche Struktur und Format der Daten. Für die Zuordnung einer Policy zu einer zugehörigen Datensammlung wurde die einheitliche Struktur und das Format einer Policy, wie in Abschnitt 3.3 beschrieben, auch als einheitliche Struktur und Format einer Datensammlung übernommen. Das genutzte Format ist demnach das XML-Format. Diese Struktur und das Format wurden bereits in der Implementierung des Pseudonymization Frameworks [5] in dieser Weise genutzt. Dabei galt es aber nur als Proof of Concept, bei dem aus einer Logdatei zwei Einträge in dieses Format und die Struktur umgewandelt worden sind, um damit für das Framework erste Daten zum Testen generieren zu können.
Die Struktur einer Datensammlung sieht demanach wie in Abbildung 2 zu sehen aus.

```
<Datensammlung type="data">
    <Datensatz id="0">
        <tag1>Inhalt1</tag1>
        <tag2>Inhalt2</tag2>
                ⋮
    </Datensatz>
    <Datensatz id="1">
            ⋮
</Datensammlung>
```

Abbildung 2.: *Die einheitlich gewählte Struktur einer Datensammlung.*

Eine Datensammlung besteht aus einem Wurzel-XML-Element, wie in Abbildung 2 das XML-Element mit dem Bezeichner *Datensammlung*. Dieser Bezeichner kann beliebig gewählt werden. Für die Zuordnung sollte aber ein aussagekräftiger Name als Bezeichner angegeben werden. Als eindeutige Erkennung, dass es sich um eine Datensammlung handelt, enthält das Wurzelelement das XML-Attribut *type* mit dem Wert *data*. Da eine Datensammlung aus Datensätzen besteht, besitzt das Wurzelelement Subelemente für diese. Die Bezeichner der Datensätze können wiederum beliebig gewählt werden. In Abbildung 2 besitzen die Subelemente den Bezeichner *Datensatz*. Genau wie bei einer Policy müssen diese Datensätze mit einer fortlaufenden Nummer, gespeichert im Attribut *id*, versehen sein, damit jeder Datensatz eindeutig identifizierbar und adressierbar ist. In der letzten Ebene liegen die einzelnen Datenfelder mit den Daten eines Datensatzes. Die Datensatz-Elemente enthalten dafür die Subelemente, deren Bezeichner in einem Datensatz beliebig aber einzigartig sein muss, wie z.B. das Element mit dem Bezeichner *tag1* in der Abbildung 2.

Transformationen. Damit die in Abschnitt 3.1 beschriebenen Formate von Datensätzen in diese einheitliche Struktur und das Format gebracht werden, wurde jeweils eine Transformation entworfen.

In einer im CSV-Format vorliegenden Datensammlung repräsentiert jede Zeile einen einzelnen Datensatz. Die Spalten, die durch die Kommatrennung entstehen, enthalten die Daten der Datensätze. Wenn die im CSV-Format vorliegende Datensammlung eine Kopfzeile enthält, werden die Beschriftungen der Spalten als Beschriftung der Tags eines Datensatzes verwendet.

Bei semi-strukturierten Daten wie zum Beispiel XML-Daten, können Datensammlungen bzw. Datensätze, wie bereits in Abschnitt 3.1 beschrieben, wesentlich komplexer und ungleich strukturiert sein. Zusätzlich können Daten nicht nur als Text eines Elements sondern auch als Attribut vorliegen. Aus diesem Grund muss erstens auch eine Datensammlung im XML-Format in die einheitliche Struktur transformiert werden und zweitens kann für solch eine Datensammlung keine allgemeine Transformation, wie für eine Datei im CSV-Format angegeben werden. Ein einfaches Beispiel einer Transformation zeigt die Abbildung 3. Bei diesem Datensatz handelt es sich um eine Nachricht. Da die Identifikationsnummer der Nachricht keine mit Null beginnende fortlaufende Zahl ist, wurde wie auf der rechten Seite zu sehen durch die Transformation in die einheitliche Struktur die Identifikationsnummer als Datum im Datensatz gespeichert und eine neue fortlaufende Identifikationsnummer hinzugefügt. Bei komplexeren bzw. stärker verschachtelten XML-Daten muss bei der Transformation auf die eindeutige Benennung der XML-Tags der Daten geachtet werden. Dazu gehören zum Beispiel eine Datensammlung bestehend

aus gespeicherten HL7-Nachrichten der Version 3 oder Dokumenten der Clinical Document Architecture.

```
<messages>
  <note id="3fe">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Re: Reminder</heading>
    <body>That's great.</body>
  </note>
</messages>
```

```
<messages>
  <note id="0">
    <ID>3fe</ID>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Re: Reminder</heading>
    <body>That's great.</body>
  </note>
</messages>
```

Abbildung 3.: *Ein Ausschnitt einer Transformation von einem Datensatz einer Datensammlung. Als Beispiel wurde ein Datensatz von einer Nachricht gewählt, bei dem die Struktur noch sehr ähnlich zur einheitlichen Struktur ist. Auf der linken Seite ist der Datensatz mit der originalen und auf der rechte Seite mit der einheitlichen Struktur zu sehen.*

Eine HL7 Nachricht der Version 2 besitzt dagegen wiederum eine sich stark unterscheidende Struktur und Format, wie zuvor in Abschnitt 3.1.2 gezeigt. Eine Nachricht ist dementsprechend ein Datensatz, der verschiedene Informationen über einen Patienten enthalten kann. Bei einer Transformation eines Datensatzes könnten auch nur Teile eines Datensatzes transformiert werden, die als interessant angesehen werden. Ein Beispiel wäre die Transformation der Daten zur Identifikation eines Patienten. Dementsprechend kann auch je nach Anwendung eine Transformation definiert werden.

Damit diese Transformationen auch praktisch genutzt werden können, wurde für die drei oben beschriebenen Transformationen jeweils eine Python-Methode implementiert. Als Erstes gehört dazu die Umwandlung einer CSV-Datei. Diese ist im Modul `csv_to_xml` enthalten. Dabei wird die erste Zeile einer CSV-Datei als Kopfzeile zur Benennung der Tags eines Datensatzes genutzt. Da eine CSV-Datei immer die gleiche Struktur aufweist, kann das Modul für alle CSV-Dateien mit einer Kopfzeile genutzt werden. Zweitens wurde die Transformation einer XML-Datei umgesetzt. Für diese gibt es die Transformationssprache "XSL Transformations" [31], durch die Transformationen von XML-Dateien in einer .xsl-Datei definiert werden können. Für die Durchführung einer Transformation wurde deshalb nur eine Python-Methode implementiert, die eine XML-Datei anhand einer XSLT-Datei aufbereitet. Diese Methode ist im Modul `xml_transformation` zu finden. Als Drittes wurde die Umwandlung einer Datensammlung bestehend aus HL7-Nachrichten der Version 2 in der klassischen Syntax, die durch jeweils eine leere Zeile getrennt sind, umgesetzt. Dabei wurde die oben beschriebene Transformation nur der Daten zur Identifikation einer Person im Modul `hl7_pid_to_xml` implementiert. Da für die Verarbeitung das Python-Package `hl7` genutzt wird, können durch kleine Anpassungen auch andere Segmente einer Nachricht transformiert werden. Für die Benennung der Datenfelder eines Datensatzes wird der Segmenttyp, der an der ersten Stelle des Segments steht, und eine fortlaufende Nummer genutzt. Wenn Daten eines Segments nicht angegeben sind, wird kein Datenfeld erstellt.
Zusammenfassend können durch Transformationen wie gezeigt unterschiedliche Datenformate so in eine einheitliche Struktur und Format transformiert werden, dass weitere Schritte bei der Verarbeitung einer Datensammlung immer gleich bleiben können.

4 XML PolicyBuilder

Obwohl XML im Allgemeinen und auch die hier spezifizierte Policy Sprache gut für die Haltung und Verarbeitung von Daten geeignet sind, sind sie auf der anderen Seite nicht automatisch menschenlesbar. Die Gründe dafür können im Gesamten an einem schlecht lesbar formatiertem XML Dokument oder im Einzelnen an XML-Elementen mit zu langen XML-Tags oder Attributen liegen.
Um Policies erstellen und bearbeiten zu können, wurde deshalb ein Python Programm namens XML PolicyBuilder erstellt. Dieses bietet die Definition der in Kapitel 3 spezifizierten Policy Sprache in vollem Umfang an. Es ist in zwei Teile, einem Front-End und und einem Back-End, unterteilt. Dadurch kann es als Programm direkt von einer Konsole oder mit einem Interface genutzt werden. Ein weiterer Vorteil ist, dass dadurch beide Teile unabhängig angepasst werden können. Die Abbildungen 4 - 6 zeigen das Interface des XML PolicyBuilder. Wie in Abbildung 4 links oben zu sehen, kann eine neue Policy erstellt oder eine existierende Policy eingelesen und bearbeitet werden. Im unteren Bereich befindet sich eine Statusleiste, die dem Benutzer ein Feedback über die getätigte Aktion gibt. Abbildung 5(a) zeigt die tag-basierte Behandlung der Datensätze mit der Möglichkeit Tags bzw. Datenfelder manuell durch ein Hilfsfenster (zu sehen in Abbildung 5(b)) hinzuzufügen. In Abbildung 6 ist dagegen die individuelle Behandlung der Datensätze zu sehen, bei der jeweils durch eine Dropdown-Liste der Datensatz und ein darin enthaltenes Datenfeld ausgewählt werden kann. Wie bei beiden Behandlungsarten der Datensätze zu erkennen, wird das Laden von Daten durch das Klicken eines Buttons (engl. für Knopf) initiiert. Damit ist das Nachladen einer Datensammlung gemeint, die den gleichen Aufbau wie eine Policy ggf. durch eine, wie in Kapitel 3.4 beschrieben, Transformation erzeugt, aufweist. Für die individuelle Behandlung ist dies notwendig um zu gewährleisten, dass nur Behandlungen für Datenfelder in der Policy angegeben werden, wenn diese Datenfelder in Datensätzen enthalten sind. Die Vertraulichkeit einzelner Daten wird dadurch zusätzlich geschützt. Bei der tag-basierten Behandlung können dadurch für den Benutzer alle Datenfelder einer Datensammlung automatisch in einer Dropdown-Liste aufgelistet werden. Als zweites ist die restliche Oberfläche beider Behandlungsarten identisch. Unten links wie in den Abbildungen 5(a) und 6 zu erkennen kann eine Behandlung bestehend aus Verfügbarkeitsoption und Bindungen dieser angegeben werden. Hierbei können ein bis zwei Bindungen für eine Verfügbarkeitsoption ausgewählt werden. Der Grund dafür ist, dass dadurch eine Rollenbindung mit einem Zweck bzw. einer Zweckbindung verbunden werden kann. Durch den nebenstehenden *Add*-Button kann eine Behandlung eines Datenfeldes zu einer Policy hinzugefügt werden. Auf der rechten Seite des Fensters ist die Liste aller hinzugefügten Behandlungen zu sehen. Durch den *Delete*-Button kann eine Behandlung gelöscht werden. Schlussendlich kann über das obenliegende Menü die Policy in einer .xml-Datei abgespeichert werden.

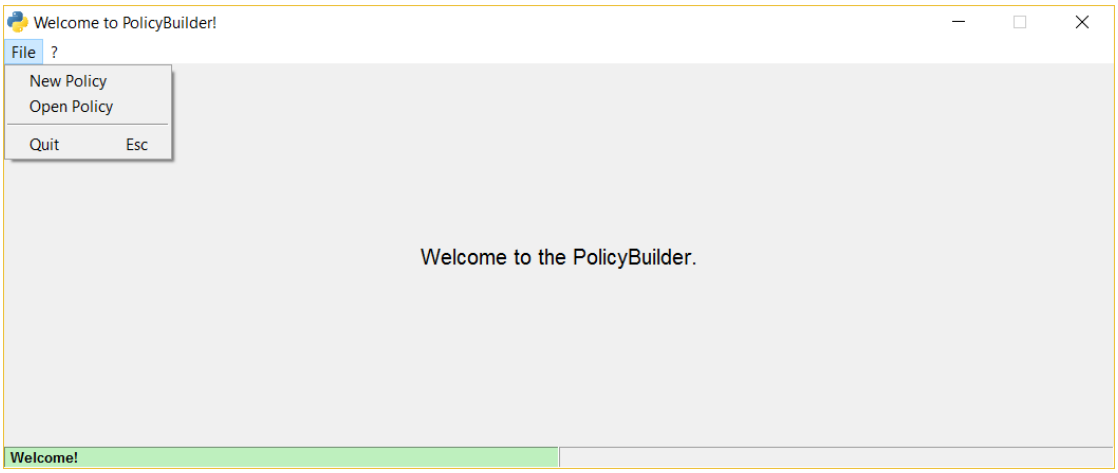
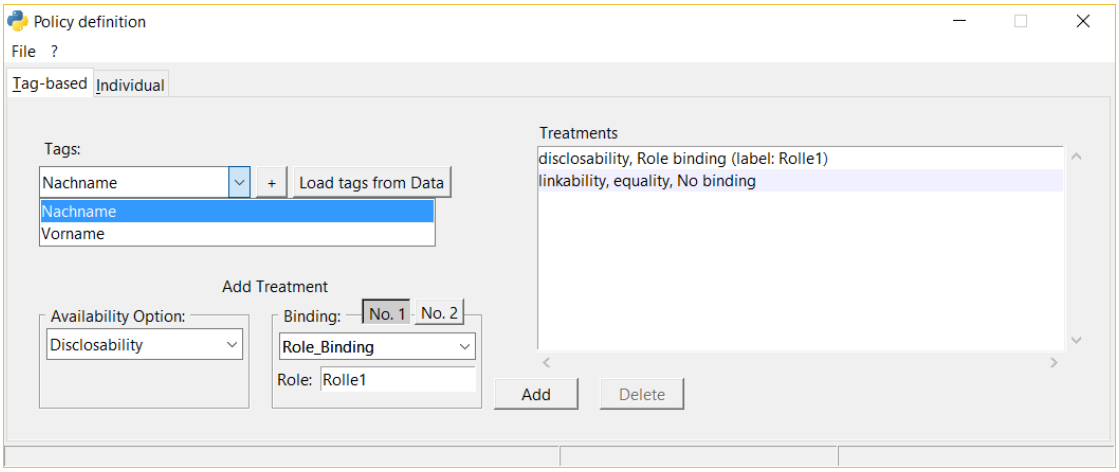
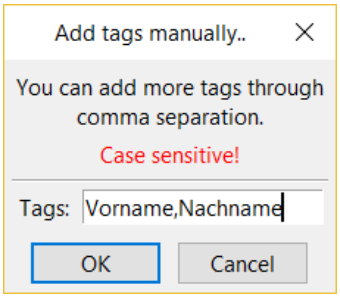


Abbildung 4.: *Benutzeroberfläche des PolicyBuilders: Startfenster*



(a) tag-basierte Behandlung



(b) Datenfeld hinzufügen.

Abbildung 5.: *Benutzeroberfläche des PolicyBuilders: tag-basierte Behandlung*

4.1 Entwurf und Aufbau

Für den Entwurf des XML PolicyBuilders wurden als Erstes alle benötigten Funktionalitäten des Back-End aufgelistet, verfeinert und als Klasse umgesetzt in einem UML-Diagramm aufgelistet.

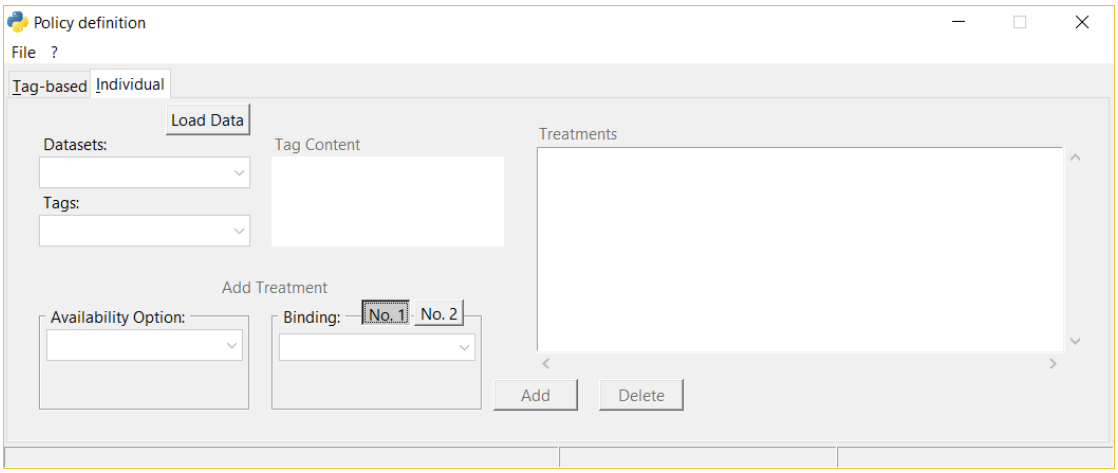


Abbildung 6.: *Benutzeroberfläche des PolicyBuilders: Individuelle Behandlung*

Dieses UML-Diagramm ist in Abbildung 7 zu sehen. Als Nächstes wurde dies in der gleichen Art und Weise für das Interface bzw. das Front-End durchgeführt. Das Hauptaugenmerk wurde beim Entwurf auf einen modularen Aufbau für eine einfache Erweiterbarkeit und eine Trennung der Aufgaben gelegt. Dafür wurde das Entity-Control-Boundary Muster [32] angewendet. Das Front-End enthält die Boundary-Elemente. Im Back-End gibt es einen Controller mit der gleichnamigen Klasse, die den gesamten Kontrollfluss steuert und gleichzeitig als Fassade (Facade-Pattern [33, S.185 ff.]) des Back-Ends dient. Dadurch kann das Back-End wie bereits erwähnt eigenständig von einer Konsole bedient und genutzt werden. Die Entity-Elemente bestehen aus einer allgemeinen Klasse zur Haltung von XML-Daten, den Spezialisierungen dieser erstens zur Haltung der Policy und zweitens zur Haltung der zur Policy gehörenden Daten und den Klassen für die Angabe der Verfügbarkeitsoptionen und den Bindungen dieser. Damit die Verfügbarkeitsoptionen und Bindungen flexibel und einfach hinzugefügt werden können, gibt es erstens eine allgemeine Klasse für beide Gruppen mit den Namen `Availability_Option` und `Bindings` von denen geerbt werden soll und zweitens für jeweils eine dieser eine `Loader`-Klasse, durch die alle vorhanden Optionen der jeweiligen Gruppe automatisch geladen werden. Um die Daten der Entity-Elemente in einer Anzeige bzw. einem Boundary aufzulisten, wurde das Beobachter-Entwurfsmuster (engl. Observer Pattern) [33, S.293 ff.] genutzt. Die Entity-Elemente speichern dabei alle Beobachter. Für das Hinzufügen eines Beobachters gibt es die `subscribe`-Methode, für das Benachrichtigen dessen die `notify_observers`-Methode, die eine `update`-Methode des Beobachters aufruft, und für das spätere Entfernen des Beobachters die `unsubscribe`-Methode. Dadurch sind die Entity- und Boundary-Elemente auf Klassenebene unabhängig voneinander. Um dies zu verstärken wird das Push-Modell des Beobachter-Entwurfsmusters verwendet, bei dem einem Boundary-Element alle benötigten Daten durch seine `update`-Methode übergeben werden.

4.2 Implementierung

Damit das resultierende Programm betriebssystemunabhängig ist, wurde die Programmiersprache Python genutzt.

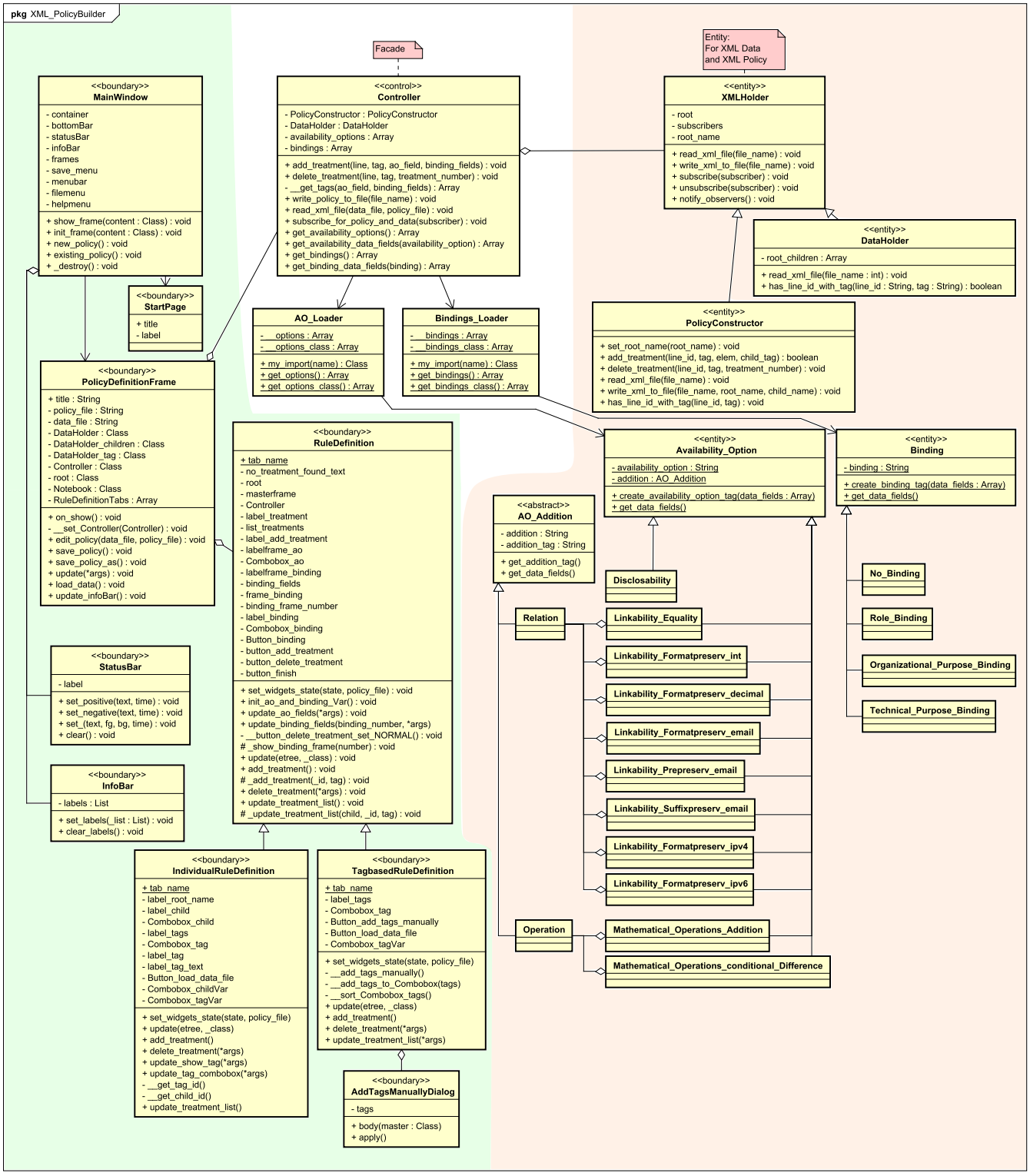


Abbildung 7.: *Ein UML-Diagramm mit allen Klassendiagrammen des XML PolicyBuilders. Leicht grün hinterlegt die Boundary-Klassen, weiß die Controller-Klasse mit den Hilfsklassen und leicht rot hinterlegt die Entity-Klassen.*

Entities. Angefangen mit den Entity-Elementen wurde als erstes die Klasse `XMLHolder` erstellt. Dieses ist, wie beim Entwurf beschrieben, ein Subjekt bzw. ein zu beobachtendes Element des Observer-Patterns und besitzt die dafür bereits beschriebenen Methoden zum verwalten und benachrichtigen der Beobachter. Für die Haltung von XML-Daten wird das `xml.etree.ElementTree`-Package (abgekürzt `ET`) [34] verwendet. Dieses besitzt einen Parser zum Verarbeiten von XML-Daten aus einer Datei oder direkt von einem String, der das XML-Wurzelelement der eingelesenen Daten zurückgibt. Die Klasse `XMLHolder` besitzt dafür die `read_xml_file`-Methode zum Laden von XML-Daten aus einer Datei und zusätzlich zur Speicherung des XML-Wurzelelements und des Bezeichners dessen in jeweils einer Instanzvariable. Um XML-Daten wiederum in einer .xml-Datei zu speichern gibt es die `write_xml_to_file`-Methode, die für eine hierarchisch formatierte Ausgabe zusätzlich die `toprettyxml`-Methode vom `xml.dom.minidom`-Package [35] nutzt, da das `ET`-Package dies nicht anbietet.
Da die Verwaltung einer Policy und einer Datensammlung sich unterscheidet, wurden die zwei von der Klasse `XMLHolder` erbenden Klassen `PolicyConstructor` und `DataHolder` erstellt. Beim Laden sowohl einer Policy als auch einer Datensammlung durch die `read_xml_file`-Methode wird zusätzlich anhand des Attributes *type* des Wurzelelements validiert, ob es sich bei den geladenen Daten um eine Policy bzw. eine Datensammlung handelt. Andernfalls wird eine Exception geworfen. Beim Schreiben der Policy in eine Datei durch die `write_xml_to_file`-Methode werden zusätzlich alle Behandlungen von Datensätzen anhand ihres Attributs *id* sortiert und optional können der Bezeichner des Wurzelelements durch die Hilfsmethode `set_root_name` und der Bezeichner des XML-Elements für die tag-basierte Behandlung geändert werden. Letzteres ist für den Fall, dass anfangs nur eine Policy und später erst die Daten geladen wurden, vorgesehen. Bei der Änderung des Bezeichners des XML-Elements für die tag-basierte Behandlung wird zusätzlich überprüft, ob nur eine Behandlung bzw. ein XML-Element mit dem Attribut *id* und dem Wert *all* für die tag-basierte Behandlung existiert.

Damit eine Behandlung für ein Datenfeld hinzugefügt werden kann, wurde die Methode `add_treatment` erstellt, der die ID des Datensatzes, der Bezeichner des Datenfeldes und das hinzuzufügende `treatment`-XML-Element übergeben werden muss. Nicht zwingend kann auch der Bezeichner des Datensatzes übergeben werden. Der Grund, warum dies nicht zwingend ist, ist das im Fall, dass die Datensammlung nicht bekannt ist oder die Bezeichner der Datensätze unterschiedlich sind, ein vorgegebener Bezeichner für die tag-basierte Behandlung gewählt wird. Da es sein kann, dass eine Behandlung aus verschiedenen Gründen nicht mehr in einer Policy enthalten sein soll, wurde zum Löschen einer Behandlung die Methode `delete_treatment` implementiert. Diese benötigt wie beim Hinzufügen die ID des Datensatzes, den Bezeichner des Datenfeldes und die Position des `treatment`-Elements im angegebenen Tag.

Bevor im Allgemeinen ein Behandlung hinzugefügt oder gelöscht wird, kann durch die `has_line_id_with_tag`-Methode überprüft werden, ob ein Element mit einer angegebenen ID und ein Subelement dessen mit einem angegebenen Bezeichner in der Policy existiert.
Die Klasse `DataHolder` zur Haltung einer Datensammlung besitzt auch diese Methode. Für die Überprüfung wird aber die bereits vorhandene bzw. durch die `read_xml_file`-Methode geladene Liste der Datensatz-Elemente (gespeichert in der Variable `root_children`) genutzt.

Bevor eine Behandlung, die aus einer Verfügbarkeitsoption und mindestens einer Bindung der Verfügbarkeitsoption besteht, hinzugefügt werden kann, müssen die entsprechenden XML-Elemente erstellt werden. Dafür wurden als Erstes die beiden allgemeinen abstrakten Klassen `Availability_Option` und `Binding` implementiert. Diese besitzen jeweils eine Klassenvariable für den Namen einer Option. Da eine Verfügbarkeitsoption eine zusätzliche Angabe besitzen kann, wie eine Relation bei der Verkettbarkeit oder eine Operation bei der Mathematischen Operation, enthält die Klasse `Availability_Option` dafür die zusätzliche Klassenvariable `addition`

(engl. für Zusatz). Diese Variable muss eine Instanz einer Klasse mit der Oberklasse `AO_Addition` enthalten, wenn es eine zusätzliche Angabe gibt. Für die Art und für den Namen der zusätzlichen Angabe sind die Klassenvariablen `addition_tag` und `addition` vorhanden. Als konkrete Klassen wurden für eine Relation und eine Operation die gleichnamigen Klassen als Unterklassen von `AO_Addition` implementiert.

Um schließlich eine konkrete Verfügbarkeitsoption oder Bindung einer Verfügbarkeitsoption zu definieren, muss eine Klasse implementiert werden, die von der Klasse `Availability_Option` oder `Binding` erbt und die Klassenvariablen für den Namen und ggf. den Zusatz initialisiert. Der Konvention folgend setzt sich der Name der Klasse aus dem Namen der Option und in dem Fall einer Verfügbarkeitsoption mit einer zusätzlichen Angabe aus dem Namen des Zusatzes zusammen. Diese Klasse muss in einer Python-Datei mit dem gleichen Namen abgelegt werden. Ein Beispiel von solch einer Klasse ist in der Abbildung 8 zu sehen. Dabei handelt es sich um die Verfügbarkeitsoption Verkettbarkeit bzgl. der Relation Gleichheit. Die Klassenvariable `availability_option` enthält dafür das entsprechende englische Wort für Verkettbarkeit und die Klassenvariable `addition` ist mit der Instanz der Klasse `Relation`, der für die Gleichheitsrelation das englische Wort *equality* übergeben wird, initialisiert.
Insgesamt wurden alle in den Tabellen 2 und 3 beschriebenen Verfügbarkeitsoptionen und Bindungen von Verfügbarkeitsoptionen als Klassen implementiert und liegen jeweils als Datei in den entsprechenden Unterordnern (`availability_options` oder `bindings`) des XML PolicyBuilders.

Datei: Linkability_Equality.py

```
1 from Availability_Option import Availability_Option, Relation
2
3 class Linkability_Equality(Availability_Option):
4     """Availability Option: Linkability with respect to the relation equality"""
5     availability_option = 'linkability' # for XML element text
6     addition = Relation('equality')
```

Abbildung 8.: *Der Aufbau einer Klasse von einer Verfügbarkeitsoption am Beispiel von der Verfügbarkeitsoption Verkettbarkeit bzgl. der Relation Gleichheit.*

Für die interne Darstellung solch einer Option, zur Kontrolle und vor allem damit individuelle Angaben, wie bei der Rollenbindung der Name der Rolle, hinzugefügt werden können, wird ein sortiertes assoziatives Datenfeld (in Python `OrderedDict` [36] genannt) genutzt. Dieses verwendet im Vergleich zu einem gewöhnlichem Datenfeld (engl. array) beliebige Schlüssel und wird im Folgenden Wörterbuch genannt. Um solch ein Wörterbuch zu erzeugen, besitzen die beiden abstrakten Oberklassen `Availability_Option` und `Binding` die Methode `get_data_fields`.
Das Wörterbuch der Klasse aus Abbildung 8 sieht wie folgt aus:

```
OrderedDict([('availability_option', 'Linkability_Equality')])
```

Und für die Rollenbindung an eine Rolle "Rolle1":

```
OrderedDict([('binding', 'Role_Binding'), ('Role', 'Rolle1')])
```

Wenn alle benötigten Daten in den Wörterbüchern für eine Verfügbarkeitsoption und eine Bindung dieser enthalten sind und diese Angaben als Treatment-XML-Element für ein Datenfeld in eine Policy hinzugefügt werden sollen, müssen die Wörterbücher jeweils durch die entsprechende `create_`*Option*`_tag`-Methode der Option zu XML-Elementen verarbeitet werden. Anschließend können diese als Subelemente eines `treatment`-Elements der Policy hinzugefügt werden.

Controller. Diese Aufgabe übernimmt u.a. die Klasse `Controller`. Sie ist, wie bereits im Entwurf beschrieben, für den gesamten Kontrollfluss also die Verwaltung einer Policy und einer Datensammlung zuständig. Gespeichert in Instanzvariblen hält sie jeweils eine Instanz der Klasse `PolicyConstructor` und `DataHolder` und jeweils ein gewöhnliches Datenfeld mit den Namen und Klassen der Verfügbarkeitsoptionen und der Bindungen der Verfügbarkeitsoptionen. Für das Laden der Optionen sind für die Verfügbarkeitsoptionen die Hilfsklasse `AO_Loader` und für die Bindungen einer Verfügbarkeitsoption die Hilfsklasse `Bindings_Loader` verantwortlich. Diese Hilfsklassen liegen jeweils in dem entsprechenden Unterordner, importieren automatisch alle sich im Ordner befindlichen Dateien bzw. deren gleichnamige Klassen und erstellen jeweils ein Datenfeld mit ihren Namen und den Klassen. Die Dateien, die keine konkrete Verfügbarkeitsoption oder Bindung einer Verfügbarkeitsoption enthalten, werden ausgelassen. Durch diesen Aufbau können neue Optionen ohne großen Aufwand und flexibel hinzugefügt werden.
Da die Klasse `Controller` als Fassade des Back-End dient, delegieren die Methoden `delete_treatment`, `read_xml_file` und `subscribe_for_policy_and_data` die übergebenen Werte nur weiter an die entsprechenden Methoden der Klassen `PolicyConstructor` und `DataHolder`.

Zu Beginn kann als erstes eine Policy oder eine Datensammlung aus einer Datei durch die `read_xml_file`-Methode geladen werden. Für das Erstellen einer neuen tag-basierten Policy ist dies nicht zwingend nötig. Um immer über Änderungen der Daten informiert zu werden, kann sich eine Instanz einer beobachtenden Klasse wie ein Boundary durch die `subscribe_for_policy_and_data`-Methode in den Klassen `PolicyConstructor` und `DataHolder` als Beobachter registrieren. Dafür muss eine Instanz nur eine Referenz auf sich selbst übergeben.
Als Zweites können durch die Methode `get_availability_options` die vorhandenen Verfügbarkeitsoptionen und durch die Methode `get_bindings` die vorhandenen Bindungen einer Verfügbarkeitsoption abgefragt werden. Wenn anschließend eine Behandlung eines Datenfeldes in eine Policy hinzugefügt werden soll, können durch die Methoden `get_availability_data_fields` und `get_binding_data_fields` und die Angabe des Namens der jeweiligen Option die Wörterbücher dieser geladen werden. Nachdem diese wenn nötig vervollständigt wurden, werden sie an die `add_treatment`-Methode mit der Angabe der ID des Datensatzes und des Bezeichners des enthaltenden Datenfeldes weitergegeben. Dabei werden die XML-Elemente erstellt und nachfolgend an die entsprechende Position in die Policy hinzugefügt.
Ist die Erstellung einer Policy vollendet bzw. soll diese gesichert werden, muss dafür der `write_policy_to_file`-Methode ein Pfad einer Datei zur Speicherung angegeben werden.

Boundaries. Damit diese bzw. die gesamte Bedienung benutzerfreundlicher ist, wurde, wie bereits am Anfang des gesamten Kapitels erklärt und durch die Abbildungen 4 - 6 gezeigt, ein Boundary bzw. ein Interface implementiert. Für die Erstellung dessen wurde das Python-Package `TKinter` genutzt, das der De-Facto-Standard in Python zur Darstellung von graphischen Benutzeroberflächen ist [37]. Um das gesamte Interface darzustellen, wurden die Oberflächenelemente Tk (das Hauptfenster), Frame, Label, Menu, Listbox, LabelFrame, Entry, Button und Toplevel und die neueren Oberflächenelemente aus dem Python-Package `ttk` namens Notebook, Scrollbar, Combobox und Separator verwendet. Um die Änderung eines Entry- oder Combobox-Elements zu erkennen, wurde die variable Klasse StringVar genutzt, die bei einer Änderung eine Aktion auslösen kann.
Das Hauptfenster des Front-End ist die Klasse `MainWindow`. Für die Flexibilität des Fensterinhalts wurde dieser in die Klassen `StartPage` und `PolicyDefinitionFrame` ausgelagert. Durch die Methoden `init_frame` und anschließend `show_frame` der Klasse `MainWindow` kann die grafische

Benutzeroberfläche einer der beiden Klassen angezeigt werden. Beim Start des Programmes wird durch die Klasse `StartPage` der Willkommenstext, wie in Abbildung 4 zu sehen, angezeigt.
Durch das obenliegende Menü des Hauptfensters kann eine neue Policy erstellt oder eine Existierende geladen werden. In diesen Fällen wird entweder die `new_policy`- oder `existing_policy`-Methode aufgerufen und anschließend die Oberflächenelemente der Klasse `PolicyDefinitionFrame` angezeigt. Dabei werden dem Menü, wenn noch nicht vorhanden, die Menüelemente *Save Policy* und *Save Policy as...* zum allgemeinen Speichern oder dem expliziten Speichern in einer Datei hinzugefügt. Die Klasse `PolicyDefinitionFrame` hält eine Instanz der Klasse `Controller` und stellt diese zur Bearbeitung einer Policy bereit.
Um die in Kapitel 3.2 erörterten Arten von Behandlungen der Datensätze zu nutzen und zu verwalten, wurden die Klassen `IndividualRuleDefinition` und `TagbasedRuleDefinition`, wie in Abbildung 5(a) und 6 zu sehen, für die individuelle und tag-basierte Behandlung implementiert. Da diese Beiden in Teilen einen gleichen Aufbau besitzen, wurden die in beiden Klassen genutzten Methoden, Variablen und Oberflächenelemente in die gemeinsame Oberklasse `RuleDefinition` verlagert. Wie in den Abbildungen 5(a) und 6 zu erkennen, gehören dazu die unten liegenden Oberflächenelemente für die Verfügbarkeitsoption und die Bindungen derer und die Oberflächenelemente in der rechten Fensterhälfte für die Liste der Behandlungen eines Datenfeldes. Die zwei Unterklassen `IndividualRuleDefinition` und `TagbasedRuleDefinition` wurden jeweils als Registerkarte des in der Klasse `PolicyDefinitionFrame` definierten `Notebook`-Elements hinzugefügt. Sie enthalten Methoden für das Aktualisieren, wie zum Beispiel des Listbox-Elements, und für das Hinzufügen und Löschen von Behandlungen. In dem Fall, dass keine Datensammlung eingelesen ist, werden, wie bereits erklärt, die einzelnen Oberflächenelemente durch die `set_widget_state`-Methode deaktiviert und beim einlesen einer Datensammlung durch die gleiche Methode aktiviert. Da bei der tag-basierten Behandlung (implementiert in der Klasse `TagbasedRuleDefinition`) die Möglichkeit gegeben ist, dass einzelne Tags bzw. Datenfelder manuell hinzugefügt werden können, wurde die Klasse `AddTagsManuallyDialog`, durch die ein Hilfsfenster, zu sehen in Abbildung 5(b), dargestellt wird, implementiert. Durch dieses können neue Tags kommagetrennt zur Liste des Combobox-Elements hinzugefügt werden, damit für diese anschließend eine Behandlung definiert werden kann.
Als letztes besitzt das Interface eine Statusleiste am unteren Rand, die sich aus den Klassen `StatusBar` und `InfoBar` zusammensetzt. Dadurch werden in der linken Hälfte der Statusleiste positive Informationen, wie der erfolgreichen Speicherung einer Policy in einer Datei, durch eine grüne Hintergrundfarbe und negative Informationen, wie der Anzeige einer Fehlermeldung beim nicht erfolgreichen Hinzufügen einer Behandlung, durch eine rote Hintergrundfarbe durch die Klasse `StatusBar` für einen definierten Zeitraum ausgegeben. Der Zeitraum für negative Informationen ist mit 8 Sekunden und für positive Informationen mit durchschnittlich 2 Sekunden definiert. Die rechte Hälfte dargestellt durch die Klasse `InfoBar` beinhaltet von links angefangen den Dateinamen einer bereits geladenen oder gespeicherten Policy und rechts weiter den Dateinamen einer geladenen Datensammlung.

4.3 Ablauf einer Erstellung einer Policy

Wie bereits vorstehend detailliert erklärt, beginnt das Erstellen einer Policy mit dem Klicken auf den Button *New Policy* im *File*-Menü. Als nächstes kann eine Datensammlung geladen werden. Dadurch müssen bei der tag-basierten Behandlung der Datensätze die Datenfelder nicht manuell hinzugefügt werden und es entstehen auch keine Eingabefehler. Als zweites können dadurch individuelle Behandlungen angegeben werden, da dafür die explizite Struktur einer Datensammlung bekannt sein muss. Wenn nur eine tag-basierte Policy erstellt werden soll,

kann dieser Schritt weggelassen werden und es können die Datenfelder manuell über den +-Button hinzugefügt werden. Der nächste Schritt ist die Auswahl eines Datenfeldes, im Fall einer tag-basierten Behandlung, oder eines Datensatzes und Datenfeldes, im Fall einer individuellen Behandlung. Anschließend kann eine Verfügbarkeitsoption und eine oder auch eine zweite Bindung (durch das Klicken auf den *No. 2*-Button) ausgewählt werden. Bei einer Rollenbindung muss, wie in Abbildung 5(b) zu sehen, noch eine weitere Angabe eingegeben werden. Wenn alles angegeben ist, kann diese Behandlung durch einen Klick auf den *Add*-Button hinzugefügt werden. Im Fall einer Eingabe einer Behandlung, die wieder rückgängig gemacht werden soll, kann diese Behandlung in der rechten Liste ausgewählt werden und durch einen Klick auf den *Delete*-Button gelöscht werden. Wenn alle Behandlungen für Datenfelder angegeben sind, kann die Policy durch die Tastenkombination *Ctrl+S* in einer Datei gespeichert werden.
Im Fall, dass eine bereits erstellte Policy geladen wird (durch *Open Policy* im *File*-Menü), können ohne das Laden der zugehörigen Datensammlung auch die individuellen Behandlungen der Datensätze bearbeitet werden. Dies beschränkt sich aber auf die bereits in der Policy angegebenen Datenfelder der Datensätze.

4.4 Benutzerfreundlichkeit

Ein Ziel war es, die Benutzeroberfläche benutzerfreundlich zu gestalten.
Im Folgenden wird deshalb die Benutzerfreundlichkeit der Benutzeroberfläche anhand der zehn allgemeinen Regeln für den Entwurf eines Benutzerinterfaces von Jakob Nielsen [38, S. 20] heuristisch analysiert und charakterisiert.
Die **erste Regel** besagt, dass ein Interface bzw. Dialog einfach und natürlich gehalten werden soll, sodass keine überflüssigen Informationen dem Benutzer angezeigt werden. Durch die kompakte Darstellung der verschiedenen Aspekte der Benutzeroberfläche, wie der Darstellung der Wahl einer Verfügbarkeitsoption anhand des Textes *Availability Option:*, der Abgrenzung durch einen Rahmen und des Combobox-Elements, soll ein Benutzer einfach und ohne Ablenkung, da zu den anderen Elementen ein Abstand gehalten wird, erkennen, für welchen Zweck Elemente in einem Bereich genutzt werden können.
Als **zweite Regel** soll die Sprache des Benutzers gesprochen werden. Dies bedeutet, dass auf bekannte Konzepte eines Benutzers geachtet wird. Zu diesen Konzepten gehört, dass die meisten europäischen Sprachen vor allem die bei der Benutzeroberfläche genutzte englische Sprache auf dem lateinischen Schriftsystem basieren, bei der die Schreibrichtung rechtsläufig und weiter von oben nach unten verläuft [39]. Durch die Positionierung der Auswahl des Datenfeldes (engl. *Tags*) in der tag-basierten Behandlung und des Datensatzes und Datenfeldes in der individuellen Behandlung im oberen linken Bereich, der darunter folgenden Angabe einer Behandlung angefangen links mit der Verfügbarkeitsoption und rechts folgend den Bindungen einer Verfügbarkeitsoption und der Anzeige der bereits vorhandenen Behandlungen in der Liste im rechten Bereich, ist der Ablauf, der beim Hinzufügen einer Behandlung durchgeführt wird, an die englische Schreibrichtung angepasst. Auch an die Anzeige eines Menüs am oberen Rand sind Nutzer gewöhnt, da dieser Aufbau in anderen verbreiteten Programmen, wie dem Dateiexplorer unter Windows, wiederzufinden ist.
Die **dritte Regel** ist, dass sich ein Benutzer so wenig wie möglich merken muss. Dies soll durch die Speicherung jeder Auswahl von einem Combobox-Element und die Anzeige der Eingabe in einem Entry-Element gewährleistet werden. Ob eine und auch welche Policy oder Datensammlung bereits von einer Datei geladen oder in einer Datei gespeichert worden ist, kann an der rechten Hälfte der Statusleiste erkannt werden. Aus diesen Gründen muss sich ein Benutzer wenig merken und kann stattdessen alle Informationen beim Hinzufügen einer Behandlung

ablesen.
Die **vierte Regel** fordert die Konsistenz einer Benutzeroberfläche. Alle Elemente mit der gleichen Bedeutung und Aktion sollen die gleiche Bezeichnung besitzen. Da sich die Benutzeroberflächen der Arten von Behandlungen einen großen Teil der angezeigten Elemente u.a. wegen der Nutzung einer gemeinsamen Oberklasse teilen und die Registerkarte für die individuelle Behandlung nur zusätzliche andere Elemente mit einer anderen Bedeutung enthalten, gibt es keine Elemente mit unterschiedlichen Bezeichnungen aber gleichen Aktionen.
Als **Fünftes** soll ein Benutzer Feedback zu den durchgeführten Aktionen bekommen. In der Benutzeroberfläche wird dies durch die Statusleiste bzw. genauer den linken Teil dieser realisiert. Wie bereits beschrieben werden dadurch positive und negative Benachrichtigungen für einen definierten Zeitraum angezeigt.
Die **sechste Regel** besagt, dass das Beenden eines Programmes und allgemein das Schließen eines Dialogs klar zu erkennen sein soll. Aus diesem Grund kann der XML PolicyBuilder durch die Option *Quit* im *File*-Untermenü und durch den Standard Schließen-Button rechts oben vom Fenster geschlossen werden. Für den Abbruch des manuellen Hinzufügens eines Datenfeldes für die tag-basierte Behandlung enthält das Hilfsfenster einen Abbrechen-Button.
Die **siebte Regel** ist, dass Abkürzungen (im englischen Shortcuts) für Experten angeboten werden sollen. Der XML PolicyBuilder enthält durch unterschiedliche Tastenkombinationen Abkürzungen. Diese Tastenkombinationen sind konsistent im Bezug zur Bedeutung der gleichen Tastenkombination in anderen Programmen gewählt. Das Haupt- und Hilfsfenster kann durch die Taste *ESC* geschlossen werden. Durch die Taste *Enter* kann ein angegebenes Datenfeld im Hilfsfenster hinzugefügt werden. Zum Wechseln zwischen den Registerkarten kann die Tastenkombination *Ctrl+Tab* gedrückt werden. Wenn eine Policy in einer Datei gespeichert werden soll, kann dies durch die Tastenkombination *Ctrl+S* in eine ggf. vorher gespeicherte Datei geschehen oder durch die Tastenkombination *Ctrl+Shift+S* der *Speichern unter* Dialog geöffnet werden.
Als **achte Regel** sollen verständliche Fehlermeldungen einem Benutzer angezeigt werden. Wie bereits bei der fünften Regel beschrieben, geschieht dies durch die untenliegende Statusleiste.
Die **vorletzte Regel** besagt, dass Fehler, die durch eine Aktion eines Benutzers entstehen könnten, präventiv vermieden werden sollen. Solch ein Fall könnte auftreten, wenn ein Benutzer eine nicht vollständig angegebene Behandlung zur Policy hinzufügen möchte. Aus diesem Grund wird als Erstes beim Hinzufügen einer Behandlung überprüft, ob eine Verfügbarkeitsoption und mindestens eine Bindung für diese angegeben ist. Falls dies nicht zutrifft, wird der Vorgang abgebrochen und es wird ein Hinweis als negative Benachrichtigung in der Statusleiste angezeigt.
Die **zehnte und letzte Regel** ist, dass Hilfe und eine Dokumentation für ein Programm angeboten werden soll. Dies ist durch die ausführliche Dokumentation des Entwurf und der Implementierung des XML PolicyBuilders in diesem Kapitel gegeben.
Aus den angeführten Gründen für die zehn heuristischen Regeln von Jakob Nielsen [38, S. 20] ist das Interface des XML PolicyBuilders benutzerfreundlich. Für eine detailliertere und aussagekräftigere Bewertung der Benutzerfreundlichkeit müssten trotzdem weitere Evaluationen, vor allem mit Benutzern, die dieses Programm nutzen sollen, durchgeführt werden.

5 Pseudonymization Framework

Um schließlich Daten in der in Kapitel 3 definierten einheitlichen Struktur anhand einer Policy, die durch die einheitlich definierte Policysprache erstellt wurde, zu verarbeiten bzw. aufzubereiten, wurde auf der Arbeit, beschrieben in der Ausarbeitung "Framework Implementation: Availability Policies for Data Pseudonymization" [5], aufgebaut. Das dort in Python implementierte Pseudonymization Framework wird wie bereits in der Zielsetzung dieser Arbeit beschrieben, an die definierte Policysprache angepasst, sodass es eine Policy in der in dieser Arbeit definierten Policysprache verarbeiten kann. Noch nicht enthaltene Verfügbarkeitsoptionen, definiert in der Policysprache dieser Arbeit, werden unter Nutzung des Python-Packages `Pseudonymization with availability options` (abgekürzt `pao`) dem Framework hinzugefügt. Dieses Package wurde in der Arbeit "Praktikum: Erweiterung eines Packages um Funktionen für Pseudonymisierung mit Verfügbarkeitsoptionen" [6] erstellt und enthält Methoden für die Erstellung von Pseudonymen, die Verfügbarkeitsoptionen anbieten. Zusätzlich wird die Struktur des Pseudonymization Frameworks optimiert.

Das Pseudonymization Framework [5] wurde für den Zweck entworfen, dass aufgezeichnete Daten an einen Analysten (in [5] *data analysis center* genannt) gesendet werden, wobei vor dem Senden jeweils für alle Daten Verfügbarkeitsoptionen und Bindungen dieser festgelegt werden können. Eine in dieser Arbeit formulierte Anforderung, Informationen nach Möglichkeit vertraulich zu behandeln (Tabelle 1), wurde beim Pseudonymization Framework auch beachtet. Wie bereits in Kapitel 3.4 beschrieben, wurde als Datenformat für die zu verarbeitenden Daten das XML-Format genutzt. Die Abbildung 2 zeigt die Struktur einer Datensammlung, so wie sie in dieser Arbeit verwendet wurde. Als reale Datensammlung wurde dort nur ein Ausschnitt einer Logdatei verwendet, die in das beschriebene Format und die Struktur umgewandelt wurde.
Wie in Kapitel 3.2 bereits betrachtet, wurden bei [5] die Daten anhand einer von Hand (ohne ein Hilfstool) erstellten Policy verarbeitet. Die Struktur und das Format der Policy entsprach dabei der Struktur und dem Format einer Datensammlung. Das bedeutet, dass für jedes Datenfeld eines Datensatzes eine Behandlung für das enthaltene Datum angegeben werden konnte. Dadurch konnte für jedes einzelne Datenfeld jedes Datensatzes eine beliebige Anzahl an Behandlungen angegeben werden. Eine tag-basierte Behandlung, die in dieser Arbeit entwickelt wurde, wurde bei [5] nicht betrachtet. Deshalb wird diese Art der Behandlung der Datensätze dem Pseudonymization Framework hinzugefügt werden.
Als Verfügbarkeitsoption wurden die Aufdeckbarkeit und die Verkettbarkeit in Bezug zur Gleichheitsrelation implementiert. Bei der Aufdeckbarkeit wurde das zu verarbeitende Datum unverarbeitet weitergeleitet und konnte anschließend, wenn angegeben, an eine Person oder einen Zweck gebunden werden. Damit ein Datum die Verfügbarkeitsoption Verkettbarkeit bzgl. der Gleichheitsrelation anbietet, wurde als Pseudonymisierungsfunktion die kryptographische Hashfunktion MD5 genutzt.
Bei den angebotenen Bindungen von Verfügbarkeitsoptionen handelt es sich um die gleichen wie in Tabelle 3 angegebenen Bindungen von Verfügbarkeitsoptionen nämlich Rollenbindung,

Organisatorische und Technische Zweckbindung. Für den Fall, dass ein Datum nicht gebunden werden soll, wurde genauso *no binding* angegeben. Umgesetzt wurden diese Bindungen als erstes durch die Verschlüsselung des Datums, dass die Verfügbarkeitsoption anbietet, mit dem symmetrisches Verschlüsselungsverfahren AES. Da bei der Rollenbindung der genutzte Schlüssel mehrfach verwendet wurde, wurde in diesem Fall einem Datum vorher ein Salt hinzugefügt. Als zweites wurden der Schlüssel und ggf. der Salt durch das asymmetrische Verschlüsselungsverfahren RSA verschlüsselt, damit auch diese bei einem Austausch der Daten sicher geschützt sind. Die resultierenden Daten wurden am Ende in einer XML-Datei gespeichert, die in der gleichen Weise strukturiert war, wie eine Datensammlung. Ein Datenfeld eines Datensatzes war dabei aber nicht einzigartig, da es, je nach der Anzahl der angegebenen Behandlungen für dieses Datenfeld, in der aufbereiteten Datensammlung vorkommen kann. Die genutzten Schlüssel und ggf. Salts wurden in Ordnern, aufgeteilt nach der Art der als letztes genutzten Bindung oder der angegebenen Rolle im Fall einer Rollenbindung, gespeichert. Das bedeutet, dass jede Rolle für ihren Schlüssel und ihre Salts einen Ordner hatte. Die Salts lagen in Unterordnern. Dabei gab es für jede vorhandene Kombination von Datensatz und Datenfeld einen Unterordner in dem für jede Behandlung ein Unterordner mit dem jeweiligen Salt lag. Für die Zweckbindungen sah diese Struktur genauso aus.

5.1 Anpassung der Implementierung

Das Ziel der Anpassung des Pseudonymization Frameworks ist, dass eine Policy dieser Arbeit verarbeitet werden kann und dass die Struktur des Pseudonymization Frameworks optimiert wird, sodass sie u.a. flexibler ist. Damit das Pseudonymization Framework auch eine Policy mit einer tag-basierten Behandlung verarbeiten kann, wurde als erstes das Modul `create_rsa_keys_and_exchange_rsa_public_key`, dass für die Erstellung der RSA Schlüsselpaare beim Analysten zuständig ist und als zweites das Modul `data_appearances_computation`, durch das Daten anhand einer Policy aufbereitet werden, angepasst.

Für das Erstellen der RSA-Schlüsselpaare wird jede Behandlung jedes Datenfeldes jedes Datensatzes der Policy einer Datensammlung abgearbeitet. Da eine tag-basierte Behandlung alle Datensätze betrifft und deshalb das Pseudonymization Framework wissen muss, um wie viele Datensätze es sich in einer Datensammlung handelt, muss diesem Modul die Anzahl der Datensätze übergeben werden. Dies wurde durch die Angabe einer Datei gelöst, in der alleine die Anzahl der Datensätze enthalten ist. Die in Kapitel 3.4 beschriebenen Methoden für die Transformation einer Datensammlung bieten die Erstellung solch einer Datei an. Das beim Pseudonymization Framework enthaltene Modul `create_xml`, durch das eine Logdatei in das benötigte XML-Format umgewandelt wird, bietet dies nicht und wurde deshalb um diese Option erweitert.
Da ein Datensatz betreffender Eintrag in einer Policy (von der in dieser Arbeit definierten Policysprache) nicht nur einmal wegen der unterschiedlichen Arten von Behandlungen vorkommen kann, wurde der direkte Zugriff auf eine Policy abstrahiert. Anstelle des direkten Zugriffs wurden für jedes der zwei Module jeweils zwei Methoden implementiert, die für den Zugriff zuständig sind. Diese Entscheidung macht das Pseudonymization Framework flexibler für mögliche weitere Behandlungen neben der individuellen und tag-basierten Behandlung, wie die die im Exkurs in Kapitel 3.2 kurz beschrieben wurden.

Die Anpassungen am Modul `create_rsa_keys_and_exchange_rsa_public_key` sehen wie folgt aus. Anstelle dass jeder Datensatzeintrag in einer Policy betrachtet wird, wird anhand der Identifikationsnummer "0" bis zur Anzahl an Datensätzen jeder Datensatzeintrag in einer Policy abgearbeitet. Durch die Methode `get_tag_elements_of_line` wird mittels einer übergebenen

Identifikationsnummer jeder Datensatzeintrag gesucht, der den entsprechenden Datensatz betrifft. Wenn eine individuelle und eine tag-basierte Behandlung für ein Datenfeld angegeben ist, sind dies zwei Datensatzeinträge. In diesen Einträgen wird im nächsten Schritt nach allen enthaltenen Datenfeldern gesucht. Die gefundenen Datensatzeinträge und Datenfeldbezeichner werden von der Methode zurückgegeben. Über die erhaltenen Datenfeldbezeichner wird im nächsten Schritt ähnlich wie vorher iteriert. Anstelle einer Liste von XML-Elementen ist dies eine Liste von Strings der Tags.
Um wie bereits vorher die Behandlungen der Datenfelder durchzugehen, war die zweite und letzte Anpassung die Suche der Behandlungen durch die Methode `find_policy_agreement_tag_treatments` anstelle der direkten Suche in den XML-Daten. Diese Methode sucht in den übergebenen Datensatzeinträgen, die von der vorher genutzten Methode `get_tag_elements_of_line` stammen, nach den Behandlungen für den angegebenen Datenfeldbezeichner und geben diese zurück. Ab dieser Stelle läuft die Verarbeitung der Daten weiter wie in der Ausarbeitung des Pseudonymization Frameworks [5] beschrieben.

Bei dem Modul `data_appearances_computation` wird dagegen über die Datenfelder der Datensätze einer Datensammlung iteriert und dabei nach den passenden Einträgen in der Policy gesucht. Wenn im Durchlauf nach einem Eintrag in einer Policy gesucht wird, der einen Datensatz betrifft, wird anstelle einer direkten Suche (in den XML Daten) die Methode `find_policy_agreement_lines` genutzt. Diese sucht die Datensatzeinträge mittels eines angegebenen Datenfeldbezeichners und einer Identifikationsnummer.
Wenn im nächsten Schritt über die Datenfelder eines Datensatzes iteriert wird, wird die Methode `find_policy_agreement_tag_treatments` für das Suchen der entsprechenden Behandlungen eines Datenfeldes genutzt. Von diesem Punkt aus geht es mit der Verarbeitung der Daten weiter, wie in [5] beschrieben. Die gefundenen Daten werden anhand ihrer Behandlungen aufbereitet.

Bei diesen beiden Modulen wird beim Laden entweder nur der Policy oder auch der Datensammlung überprüft, ob es sich dabei um die benötigten Daten handelt bzw. ob im Wurzelelement das Attribut *type* den jeweils richtigen Wert enthält.

Zum Überprüfen des Pseudonymization Frameworks existiert bereits von vorher das Modul `decrypt_data_appearances_in_analysis_center`. Durch dieses werden die aufbereiteten Daten, die ein Analyst erhält, unter der Annahme, dass die Voraussetzung für die Entschlüsselung vorliegt, entschlüsselt. Da dieses Modul nicht zur Funktionalität des Pseudonymization Frameworks gehört, sondern dieses überprüfen soll, wurde das Modul und der zugehörige Ordner `retrieve_keys_and_salt_and_decryption` in den Unterordner *tests* verschoben. Damit dieses Modul auch zum Überprüfen des angepassten Pseudonymization Frameworks genutzt werden kann, wurde dieses auch angepasst. Aus dem Grund, dass das Modul genauso wie das Modul `create_rsa_keys_and_exchange_rsa_public_key` jede Behandlung jedes Datenfeldes jedes Datensatzes der Policy einer Datensammlung abarbeitet, wurden dort die gleichen Anpassungen vorgenommen und auch die beiden oben beschriebenen Methoden hinzugefügt. Zusätzlich wurde dabei ein kleiner Designfehler behoben, durch den vorher bei der Iteration über die Behandlungen jeweils nach allen Datenfeldern des Datensatzes gesucht wurde. Dies kann Entschlüsselung der aufbereiteten Daten verlangsamen.

Designoptimierung. Neben der Anpassung für das Hinzunehmen einer tag-basierten Behandlung wurde auch die Struktur des Pseudonymization Frameworks optimiert. Die Methoden der Verfügbarkeitsoptionen lagen vorher in dem Python-Modul `availability_option_computation` im Ordner *availability_options*. Der Python-Code einer neue Methode für eine Verfügbarkeitsoption konnte durch das Modul `add_availability_option` und eine Python-Datei, die diese neue Methode enthält, hinter die vorhandenen Methoden in das Modul `availability_option_computation` hineinkopiert

werden. Für die Methoden der Relationen der Verfügbarkeitsoption Verkettbarkeit galt dies ebenfalls. Die zugehörige Datei lag im Unterordner *relations*.
Um dies zu optimieren, um das Pseudonymization Framework flexibler zu gestalten, wurde eine eigene Importiermethode implementiert, die anhand des Namens einer angegebenen Verfügbarkeitsoption die zugehörige Datei und Methode abfragt. Als Konvention gilt dabei, dass die Dateien und Methoden neuer Verfügbarkeitsoptionen den gleichen Namen wie eine angegebene Verfügbarkeitsoption in einer Policy besitzen müssen. Wenn ein Name einer Verfügbarkeitsoption Leerzeichen enthält, müssen diese durch Unterstriche ersetzt werden. Der Unterordner *relations* wurde zur Verallgemeinerung in *options* umbenannt, da die Verfügbarkeitsoption Mathematische Operation auch wie die Verkettbarkeit weiter in Operationen unterteilt ist. Die beschriebene Konvention der Verfügbarkeitsoptionen gilt auch für die Zusätze der Verfügbarkeitsoptionen. Zusätzlich muss der Dateiname einer Relation mit "relation_" und der Dateiname einer mathematischen Operation mit "operation_" beginnen.
Da es mehrere Verfügbarkeitsoptionen geben wird, die einen Zusatz besitzen, wurde die implementierte Importiermethode für die Dateien der Verfügbarkeitsoptionen in das Modul `addition_loader` ausgelagert.
Die Struktur im Ordner, der die Bindungen der Verfügbarkeitsoptionen enthält, sah wie bei den Verfügbarkeitsoptionen vorher aus. Genau wie dort wurden die Bindungen in eigene Dateien unter der gleichen Namenskonvention ausgelagert und werden durch die gleiche Methode vom Modul `binding_computation` aufgerufen.
Durch diese angepasste Struktur des Pseudonymization Frameworks können später neue Verfügbarkeitsoptionen, deren Zusätze und neue Bindungen flexibel ergänzt oder entfernt werden.

5.2 Hinzugefügte Verfügbarkeitsoptionen

Die in der Tabelle 2 (S.17) im Kapitel der Policysprache definierten Verfügbarkeitsoptionen wurden dem Pseudonymization Framework hinzugefügt, wenn diese noch nicht vorhanden waren. Zu den vorhandenen gehören die *Aufdeckbarkeit* und die *Verkettbarkeit* bzgl. der *Gleichheitsrelation*. Die Verfügbarkeitsoption *Verkettbarkeit* erhält die neuen Relationen *Formaterhaltend* für die Datentypen Integer, Decimal, E-Mail Adresse, IPv4 und IPv6 Adresse, *Präfixerhaltend* für den Datentyp E-Mail Adresse und *Suffixerhaltend* für den Datentyp E-Mail Adresse. Die Verfügbarkeitsoption *mathematische Operation* bietet die Erstellung von Pseudonymen für die Operationen *bedingte Differenz* und *Addition*.

Umsetzung. Für die Umsetzung der neuen Relationen und Operationen der Verfügbarkeitsoption *Verkettbarkeit* und *mathematische Operation* wird, wie bereits erwähnt, das in der Arbeit "Praktikum: Erweiterung eines Packages um Funktionen für Pseudonymisierung mit Verfügbarkeitsoptionen" [6] erstellte Python-Package `pao` genutzt, dass Funktionen bzw. Methoden enthält, durch die Pseudonyme aufbereiteter Daten jeweils eine Verfügbarkeitsoption anbieten können.
Für die Relationen *Format-*, *Präfix-* und *Suffixerhaltend* wird das Modul `pseudo` des Packages verwendet. Die darin enthaltenen Methoden benötigen neben dem Klartext einen Salt. Dieser wurde global jeweils für alle Relationen festgelegt und kann für jede Datensammlung, die aufbereitet werden soll, verändert werden. Der Vorteil davon ist, dass Datensammlungen, die gleiche Daten enthalten, nicht auf Gleichheit der aufbereiteten Daten überprüft werden können.
Wie bereits erklärt, wird die Relation *Formaterhaltend* für den Zweck angeboten, dass einerseits anhand eines Pseudonyms noch erkannt werden kann um welchen Datentyp es sich handelt bzw. wie die Struktur des Klartextes aussieht und dass andererseits die Gleichheit von zwei Klartexten

anhand ihrer Pseudonyme überprüft werden kann. Für eine IP-Adresse der Version 4 wird die Methode `ip_adress_v4` und für eine IP-Adresse der Version 6 die Methode `ip_adress_v6` eingesetzt.
Durch die Methode `integer`, die für eine Ganzzahl bzw. Integer genutzt wird, kann anhand des resultierenden Pseudonyms zusätzlich erkannt werden, aus wie vielen Stellen der Klartext besteht. Dies kann z.B. für der Analyse von Gehältern genutzt werden, wenn ihr genauer Betrag nicht angegeben werden darf.
Eine pseudonymisierten Dezimalzahl, für die die Methode `decimal` verwendet wird, gibt zusätzlich preis wie viele Nachkommastellen der Klartext enthält.
Neben der Formaterhaltung einer E-Mail Adresse, für das die Methode `email` des Moduls genutzt wird, gibt es noch die Präfix- und Suffixerhaltung. Als Präfix einer E-Mail Adresse werden dabei alle Zeichen vor dem "@"-Symbol angesehen. Im Gegensatz dazu werden bei der Suffixerhaltung alle Zeichen nach dem "@"-Symbol erhalten. Die dafür genutzten Methoden sind die Methode `pre_preserv_email` und `suf_preserv_email`.

Für die Erstellung der Pseudonyme auf denen je nach angegebener Verfügbarkeitsoption noch die Operation *bedingte Differenz* oder *Addition* ausgeführt werden kann, werden zwei Verfahren genutzt, die im Package `pao` implementiert wurden.
Damit zwei Pseudonyme addiert werden können, werden ihre Klartexte durch das Paillier-Kryptosystem [40] verschlüsselt. Bei diesem handelt es sich um ein asymmetrisches Kryptosystem, dass additiv homomorph ist. Zwei Chiffrate bzw. hier Pseudonyme genannt, können unter bestimmten Bedingungen multipliziert werden und es entsteht das Pseudonym des Resultats der addierten Klartexte. Die Bedingung hierbei ist, dass die Summe der Klartexte nicht größer ist, als das gewählte Modul n, dass die Größe der Restklassengruppe $(\mathbb{Z}/n\mathbb{Z})$ angibt, da das resultierende Pseudonym ansonsten nicht mit der Summe der Klartexte übereinstimmt. Um die pseudonymisierten Summen der Klartexte rekonstruieren zu können, ist der private Schlüssel erforderlich. Durch diesen Schlüssel könnte er aber auch die anderen Pseudonyme entschlüsseln. Aus diesem Grund gibt es bereits den Ansatz eine vertrauliche dritte Partei für die Entschlüsselung zu nutzen [41]. Die genutzte Methode für die Erstellung eines Pseudonyms ist die Methode `encrypt` im Modul `paillier` des Packages `pao`.
Als zweite Operation wurde die bedingte Differenz eingeführt. Die dafür genutzte Methode stammt von Kerschbaum [42]. Seine Methode wurde dafür entwickelt um den Abstand zwischen zwei pseudonymisierten Unix-Zeitstempeln berechnen zu können, wenn der Abstand nicht größer als ein vorher definierter Grenzwert ist. Da ein Unix-Zeitstempel eine positive Ganzzahl ist, kann die Methode für die Differenz von zwei positiven Ganzzahlen genutzt werden. Für die Erstellung des Pseudonyms wird in diesem Fall die Methode `pseudonymize_timestamp` im Modul `kerschbaum` vom Package `pao` genutzt.

Da die Methoden der Verfügbarkeitsoptionen zur Aufbereitung von Klartextdaten teils zusätzliche Angaben bzw. Parameter benötigen, wurde eine zentrale Konfigurationsdatei *config.cfg* erstellt, die diese Parameter enthält. Diese liegt im Hauptverzeichnis des Pseudonymization Frameworks. Dadurch können diese Parameter zentral mit wenig Aufwand verändert werden, damit, wie bereits beschrieben, die Möglichkeit besteht diese pro Datensammlung zu verändern.

Neben den neuen Verfügbarkeitsoptionen wurde die Umsetzung der Verkettbarkeit in Bezug zur Gleichheitsrelation in der Hinsicht auf die Sicherheit optimiert. Anstelle der kryptographischen Hashfunktion MD5 wird die kryptographische Hashfunktion SHA-256 verwendet, da gegen MD5 bereits Kollisionsangriffe [43] existieren und SHA-256, dass zur SHA-2 Familie gehört, als sicher angesehen wird [44].

6 Evaluation

Im Folgenden werden die in dieser Arbeit implementierten Komponenten des Frameworks evaluiert. Die erste Komponente sind die **Transformationen**, durch die Datensammlungen in das einheitliche Format umgewandelt werden, dass in dieser Arbeit definiert wurde. Die genauen Transformationen die implementiert worden sind, sind die Transformation einer CSV-Datei, einer HL7 Datensammlung der Version 2 in der klassischen Syntax genauer des PID Segments oder einer XML-Datei unter Nutzung einer XSLT-Datei in eine XML-Datei, die das definierte einheitliche Format aufweist. Die zweite Komponente des Frameworks ist der **XML PolicyBuilder**, durch den Policies erstellt werden können. Diese Policies legen für die letzte und dritten Komponente, das **Pseudonymization Framework**, fest, in welcher Art und Weise eine Datensammlung, zu der eine angegebene Policy gehört, verarbeitet werden soll.

In der Evaluation dieser drei Komponenten wurden jeweils zwei Aspekte betrachtet. Sowohl die Funktionssicherheit der drei Frameworkkomponenten als auch die Effizienz dieser soll überprüft werden. Für die Funktionssicherheit wurden Tests definiert, durch die alle in dieser Arbeit implementierten Funktionen und Klassen überprüft werden, um die fehlerfreie Funktion dieser zu gewährleisten. Zur Bestimmung der benötigten Ressourcen der drei Frameworkkomponenten, wurden mehrere Messungen durchgeführt. Dabei wurden die Zeit und der benötigte Arbeitsspeicher gemessen, die bei der Verwendung einer Frameworkkomponente bzw. bei der Durchführung einer Aufgabe benötigt werden. Vor jeder Messung wurde ein Neustart des Rechners durchgeführt, damit während der Durchführung einer Messung nur Prozesse von Programmen liefen, die für die Messung benötigt wurden, damit es so gut wie keine Faktoren gab, die die Messungen beeinflussen konnten. Bei allen Messungen wurde für die Ermittlung des benötigten Arbeitsspeichers das Überwachungstool "Process Monitor" [45] genutzt, durch dass die Aktivitäten im Dateisystem, im Prozessor und im Arbeitsspeicher von allen laufenden Prozessen unter Windows gemessen werden können. Genauer ist im "Process Monitor" der verwendete Arbeitsspeicher eines Prozesses sein "Working Set" [46]. Um den verwendeten Arbeitsspeicher eines Prozesses zu messen, der eine Aufgabe ausführt, die evaluiert werden soll, wurde die Aufnahme aller Aktivitäten im "Process Monitor" vor dem Start solch einer Aufgabe gestartet und nach dem beenden dieser gestoppt. Anschließend wurde der Prozess im Prozessbaum gesucht, um alle Aktivitäten dessen zu filtern und im Weiteren den verwendeten Arbeitsspeicher in der "Process Activity Summary" auszulesen.

Die Spezifikationen des Rechners, der für die Durchführung der Messungen genutzt wurde, sind in der Tabelle 4 zu finden. Dadurch und durch die Angabe aller zur Durchführung der Messungen gehörenden Informationen, soll die Reproduzierbarkeit und die Nachvollziehbarkeit der gemessenen Ergebnisse ermöglicht werden.

Computerspezifikationen	
Betriebsystem	Microsoft Windows 10 Home 64-bit (Version 10.0.14393)
Prozessor	Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz 1.80GHz
Physischer Speicher (RAM)	8,00 GB
Speichermedium	Festplatte

Tabelle 4.: *Spezifikationen des Rechners, der für die Durchführung der Messungen in der Evaluation genutzt wurde.*

6.1 Transformationen

Für die Evaluation der drei implementierten Transformationen, wurden als erstes für jedes Modul Tests definiert. Der erste Test überprüft dabei immer, ob alle implementierten Methoden im geladenen Modul enthalten sind, um sicher zu stellen, ob es sich bei dem geladenen Modul um das korrekte Modul handelt. Da jedes Modul eine `main`-Methode beinhaltet, gibt es immer mindestens einen Test der diese Methode überprüft. Die Transformation einer XML-Datei in das einheitliche XML-Format verwendet die bereits existierende Methode einer Transformation einer XML-Datei durch eine XSL-Transformation. Dabei werden nur die XML- und die XSLT-Datei eingelesen, die Methode zur Transformation der Daten genutzt und schließlich die resultierenden Daten in die entsprechenden Dateien geschrieben. Aus diesem Grund gibt es neben dem Test, ob die `main`-Methode im geladenen Modul enthalten ist nur noch einen Test, der überprüft ob bei richtiger Angabe der benötigten Daten die Methoden zur Transformation der Daten aufgerufen werden.
Bei Transformation einer CSV-Datei in das einheitliche XML-Format müssen neben dem Parsen der CSV-Datei, die XML-Elemente erstellt und die CSV-Daten in diese eingefügt werden, um anschließend in die entsprechenden Dateien geschrieben zu werden. Damit diese Schritte getestet werden, wurden zwei Tests erstellt. Im ersten Test wird eine Attrappe (im Englischen ein sogenanntes Mock-Objekt) einer Datei mit CSV-Daten an den Parser übergeben und es wird getestet, ob die resultierende Ausgabe mit der richtigen Ausgabe übereinstimmt. Dadurch wird überprüft ob die gegebenen Daten korrekt aufbereitet werden. Im zweiten Test wird zusätzlich, durch das Einlesen einer CSV-Datei, überprüft, ob es keine Probleme beim Einlesen und anschließenden Verarbeitung einer CSV-Datei gibt.
Bei der dritten Transformation der Transformation des PID-Segments einer HL7-Datei, werden neben der `main`-Methode die weiteren Hilfsmethoden getestet. Nur wenn diese korrekt funktionieren und die benötigten XML-Elemente und Daten für diese Elemente erstellen, kann die `main`-Methode ihre Aufgabe korrekt durchführen. Als letztes wird die `main`-Methode durch die Eingabe eines HL7-Datensatzes und dem Vergleich des Resultats mit dem korrekten Resultat im gesamten überprüft.
Diese beschriebenen Tests liegen im Unterordner *tests* vom Ordner der Transformationen und können durch den Aufruf des Befehls `python -m nose` in einer Konsole im Unterordner gestartet werden.

Um die Laufzeit und den Speicheraufwand zu überprüfen, wurden als nächstes Messungen durchgeführt. Wie bereits beschrieben wurde für die Messung des verwendeten Arbeitsspeichers bei der Durchführung einer Transformation das Überwachungstool “Process Monitor” genutzt. Um die dabei benötigte Zeit zu messen, wurde vor dem Aufruf der `main`-Methode, die die Transformation durchführt, der Programmcode um die Speicherung der aktuellen Zeit in einer Variable ergänzt. Nach Vollendung des Aufrufs wurde aus der gespeicherten Zeit und der aktuellen Zeit die verstrichene Zeit berechnet und ausgegeben.

Verwendete Datensätze. Damit die Messergebnisse vergleichbar sind und es keine störenden und Ergebnis verzerrenden Einflüsse gibt, wurden Datensammlungen gewählt, die eine annähernd gleiche Struktur aufweisen. Für jede der zu transformierenden Formate wurde eine Datensammlung erstellt, die aus zehn Datensätzen mit jeweils genau zehn Daten besteht. Die Daten der CSV-Datensammlung stammen von einer Logdatei [47]. Für die anderen beiden Formate wurden fiktive Daten für die Datensätze gewählt, für die als Vorlage die Beispieldatensätze genutzt wurden, die in Kapitel 3 enthalten sind. Neben den Datensammlungen mit zehn Datensätzen wurden jeweils Datensammlungen mit hundert und tausend Datensätzen durch das wiederholen der zehn Datensätze der ersten Datensammlung erstellt.

Im nächsten Schritt wurden diese Datensätze jeweils durch die zugehörige Transformation in das in dieser Arbeit definierte einheitliche XML-Format umgewandelt. Zur Verringerung des Einflusses von Ausreißern auf das gemessene Ergebnis, wurde jede Messung zehn Mal durchgeführt und es wurde der Durchschnitt dieser Ergebnisse als Wert genutzt. Diese gemittelten Werte sind in den beiden Graphen der Abbildung 9 zu sehen. Die Anzahl der Datensätze ist die unabhängige Variable, die auf der x-Achse aufgetragen ist und die Zeit (in Sekunden) im linken und der Arbeitsspeicher (in Megabyte) im rechten Graphen ist die abhängige Variable, die durch die unabhängige Variable beeinflusst wird und von der y-Achse abzulesen ist. Damit die Anzahl an Datensätzen pro Datensammlung gut von der x-Achse abzulesen ist, wurden die x-Achsen logarithmisch skaliert. Im Bezug auf die Zeit ist zu erkennen, dass der Zeitverbrauch der beiden Transformationen von CSV- und XML-Datensammlungen fast gleich ist. Bei zehn und hundert Datensätzen liegt die benötigte Zeit unter 0,03 Sekunden. Die Erhöhung der Datensätze um den Faktor zehn von hundert auf tausend Datensätze beeinflusst die Zeit nur ca. um den Faktor vier. Die Transformation einer HL7 Datensammlung dauert dagegen etwas länger. Von einer Zeit von ca. 0,06 Sekunden bei zehn Datensätzen, über die Anzahl an hundert Datensätzen, bei der die Zeit um den Faktor fünf steigt, bis hin zu tausend Datensätzen, bei denen es sechsmal länger dauert, erhöht sich die gemessene Zeit bei der Transformation der Datensätze. Der Verlauf des benötigten Arbeitsspeichers sieht vergleichsweise ähnlich aus. Im ersten Bereich von zehn bis hundert beträgt die Steigung nur ca. 0,008 Prozent. Im zweiten Bereich von Hundert auf Tausend Datensätze sänkt sich dieser Wert auf ca. 0,006 Prozent.
Abschließend ist zu erkennen, dass die Transformationen zumindest für kleine Datensammlungen verschiedener Größe in sehr kurzer Zeit und mit wenig Arbeitsspeicher durchgeführt werden können.

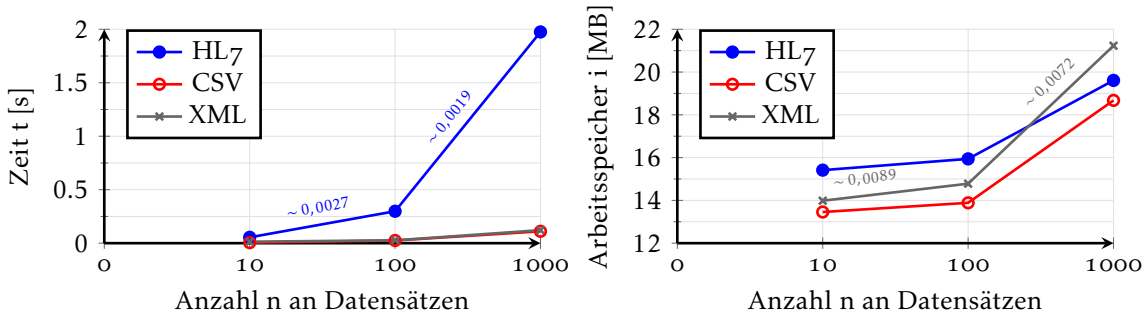


Abbildung 9.: *Die benötigte Zeit (links) und der verwendete Arbeitsspeicher (rechts) bei den Transformationen von HL7-, CSV- und XML-Datensammlungen mit den Anzahlen an Datensätzen 10, 100 und 1.000 und jeweils 10 Daten pro Datensatz.*

6.2 XML PolicyBuilder

Der im Abschnitt 6.1 beschriebene Versuchsaufbau wurde ebenfalls bei der Evaluation des XML PolicyBuilders genutzt.
Angefangen mit den Funktionstests wurde für jede Methode der implementierten Back-End Klassen ein Test definiert. Bei den Klassen der Verfügbarkeitsoptionen wurde die Klasse `AO_Loader` direkt und die anderen Klassen indirekt getestet. Das bedeutet, dass überprüft wurde, ob alle Verfügbarkeitsoptionen durch das Loader-Modul geladen werden können. Wenn dies erfolgreich ist, wird durch die geladenen Module der Verfügbarkeitsoptionen überprüft, ob diese einen Namen enthalten, ein korrektes Datenfeld zurückgeben und für das Hinzufügen in eine Policy das richtige XML-Element mit den benötigten Bestandteilen erstellen. Die Tests für die Methoden und Klassen einer Bindung einer Verfügbarkeitsoption wurden äquivalent definiert, weil sie gleich aufgebaut sind. Da diese Klassen von einer Oberklasse erben, im Fall einer Verfügbarkeitsoption von der Klasse `Availability_Option` und im Fall einer Bindung einer Verfügbarkeitsoption von der Klasse `Binding`, wird durch die Tests auch die Funktionalität dieser Oberklassen überprüft.
Für die Evaluation der weiteren Back-End-Klassen, die die XML-Daten halten oder verwalten, wurde für jede Klasse eine Testklasse erstellt, die für jede öffentliche Methode der Klassen Testmethoden enthalten. Bei der Oberklasse `XMLHolder` für die Haltung der Daten, wurde getestet, ob Daten, wenn sie vorhanden sind, gelesen und geschrieben werden können und ob Objekte, die über Änderungen informiert werden möchten, bei einer Instanz der Klasse angemeldet, abgemeldet und benachrichtigt werden können. Für das letztere wurde eine Attrappe eines Objekts erstellt, dass angemeldet, benachrichtigt und abgemeldet wurde. Im Test der erbenden Klasse `DataHolder` wurden zusätzlich eine Datensammlung eingelesen, um zu überprüfen, ob alle Datensätze und Daten der Datensätze gefunden werden können. Bei der Klasse `PolicyConstructor` wurde neben diesen Tests auch überprüft, ob eine Behandlungsvorschrift hinzugefügt und wieder gelöscht werden kann. Dabei wurde auch überprüft, ob bei einer fehlerhaften Nutzung, der Angabe von nicht validen Daten, eine Ausnahme geworfen wird. Als letztes wurde die Klasse `Controller` getestet, die für das Zusammenspiel bzw. die Verwaltung von allen anderen Back-End-Klassen zuständig ist. Im Einzelnen ist dies, dass diese Klasse auf alle Klassen und deren benötigten Methoden zugreifen kann und die Operationen, wie das Hinzufügen und Löschen von validen Daten, erfolgreich durchführen kann.
Am Ende wurde noch ein Systemtest erstellt, durch den die gesamte Funktionalität des Back-Ends überprüft wird. Dabei wurden Daten geladen, eine tag-basierte und eine individuelle Behandlungsvorschrift hinzugefügt und schließlich überprüft, ob die ausgegebene Policy korrekt ist. Diese Tests befinden sich im Unterordner *tests* und können dort durch den Befehl `python -m nose` in einer Konsole ausgeführt werden.

Für einen kostenlosen, möglichst vollständigen und zeitlich effizienten Test der graphischen Benutzeroberfläche, wurde für das Front-End ein Protokoll mit zwanzig Tests erarbeitet, die während der Programmierung der grafischen Oberfläche einzeln und am Ende im gesamten durchgeführt wurden. Dieses Protokoll ist im Anhang A.1 zu finden. Durch diese Tests werden Eigenschaften, wie die Unterscheidung zwischen dem Schließen des Fensters ohne oder mit Nachfrage, als auch die Hauptkomponenten zum Laden von Daten und zum Erstellen einer Policy überprüft. Dabei wird nicht nur getestet, ob eine Policy durch valide Eingaben, wie dem Auswählen einer Verfügbarkeitsoption und ein bis zwei Bindungen einer Verfügbarkeitsoption, der Auswahl eines Datenfeldes oder bei der individuellen Behandlungsvorschrift zusätzlich des Datensatzes und dem Hinzufügen oder Löschen einer Behandlungsvorschrift, erstellt werden kann, sondern auch, ob eine fehlerhafte Bedienung nicht zu einem ungewollten Verhalten des Programms führt, wodurch es im schlimmsten Fall nicht mehr für den gewollten Zweck genutzt werden kann. Zu diesen Fehlbedienungen gehört das Hinzufügen einer Behandlungsvorschrift,

obwohl mindestens eine der benötigten Angaben, wie eine Verfügbarkeitsoption, fehlt oder das manuelle Hinzufügen eines Datenfeldes für die tag-basierte Behandlung der Datensätze, das bereits vorhanden ist und nicht mehrmals hinzugefügt werden soll.

Neben diesen Funktionssicherheitstests wurde evaluiert, welche Komponenten zu einer Verzögerung bei der Bedienung führen und wie viel Arbeitsspeicher vom XML PolicyBuilder verwendet wird, wenn eine Datensammlung geladen wird. Bei der Durchführung der Funktionssicherheitstests wurde erkannt, dass im Vergleich zu allen anderen Aktionen nur das Laden einer Datensammlung zu einer Verzögerung führt. Da auch nur eine geladene Datensammlung die Größe des genutzten Arbeitsspeichers stark beeinflusst, wurde eine Messung wie bereits bei den Transformationen durchgeführt, bei der die benötigte Zeit und anschließend der verwendete Arbeitsspeicher aufgezeichnet wurden. Für die Ermittlung der benötigten Zeit, die das Laden einer Datensammlung dauert, enthält der XML PolicyBuilder bereits die Funktion, die vor dem Parsen der Datensammlung die aktuelle Zeit speichert und nach dem Laden daraus die benötigte Zeit berechnet und in der Statusleiste ausgibt. Für die Ermittlung des verwendeten Arbeitsspeichers wurde (wie vorher bereits) das Überwachungstool "Process Monitor" genutzt. Als Datensammlung wurde die Datensammlung verwendet, die bei der Evaluation der XML-Transformation in Kapitel 6.1 erstellt wurde. Weitere unterschiedlich großen Datensammlungen wurden durch die Wiederholung der Datensätze erzeugt. Die gemessenen Ergebnisse bestehen aus den gemittelten Werten aus fünf Messungen und sind in den Graphen der Abbildung 10 zu finden. Dabei ist zu erkennen, dass die benötigte Zeit bei einer Erhöhung der Datensätze nur sehr gering im Vergleich zur Anzahl der Datensätze steigt. Dies gilt auch beim verwendeten Arbeitsspeicher. Bei beiden ist die Steigung kleiner als 0,04 Prozent und verändert sich bei einer Erhöhung der Datensätze nur kaum messbar. Dementsprechend handelt es sich um eine niedrige aber lineare Steigung. Zusammenfassend können Datensammlungen mit zumindest bis zu zehntausend Datensätzen auf einem durchschnittlichen Rechner durch den XML-PolicyBuilder geladen werden, um eine Policy für diese zu erstellen.

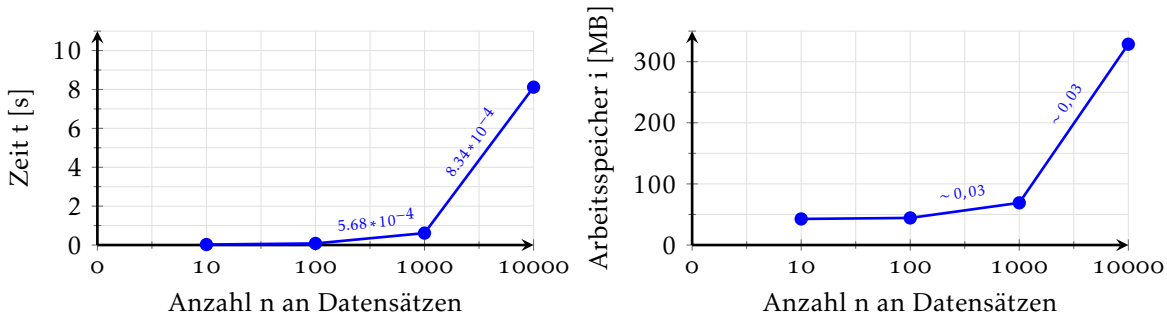


Abbildung 10.: *Die benötigte Zeit (links) beim Laden einer Datensammlung und anschließend verwendete Arbeitsspeicher (rechts) des XML PolicyBuilders mit den Anzahlen 10, 100, 1.000 und 10.000 an Datensätzen und jeweils 10 Daten pro Datensatz.*

6.3 Pseudonymization Framework

Für die Evaluation des erweiterten Pseudonymization Frameworks wurden Tests definiert, die alle neu hinzugefügte Methoden testen, der Zeit- und Speicheraufwand gemessen, der für das Erstellen des Schlüsselmaterials beim Analysten und anschließend für die Verarbeitung der Daten anhand einer Policy benötigt wird, und als letztes die dabei verarbeiteten Daten unter der

Annahme, dass die Voraussetzungen für die Entschlüsselung der Daten vorliegen, entschlüsselt und mit den separat nur durch die Verfügbarkeitsoption verarbeiteten Daten verglichen. Die definierten Tests bzw. Testmethoden liegen im Unterordner *tests* des Pseudonymization Frameworks und können (wie bereits bei den anderen Tests) durch den Befehl `python -m nose` in einer Konsole ausgeführt werden. In Bezug zu den Verfügbarkeitsoptionen, den Zusätzen (Relationen und Operationen) der Verfügbarkeitsoptionen und Bindungen wird überprüft, ob diese vorhanden sind bzw. importiert werden können und ob sie ihren Zweck, die Aufbereitung oder Verschlüsselung von Daten, erfüllen. Dafür werden Beispieldaten übergeben und das Ergebnis, wie z.B. ein aufbereitetes Datum oder der erfolgreiche Aufruf der AES-Verschlüsselung, wird mit dem gewünschten Ergebnis verglichen. Zusätzlich wird bei den Verfügbarkeitsoptionen getestet, ob die zusätzlich benötigten Daten, die in der *config.cfg*-Datei enthalten sind, vorhanden sind und abgerufen werden können. Des Weiteren werden die Methoden, die durch die Veränderungen an den Modulen zum Erstellen des Schlüsselmaterials und zur Aufbereitung der Daten und dem Testmodul zur Entschlüsselung der aufbereiteten Daten, getestet. Dabei wird überprüft, ob die Angaben zu den Datensätzen, den enthaltenen Datenfelder der Datensätze und deren Behandlungen in einer Policy gefunden werden, wenn keine, nur individuelle Behandlungsvorschriften, nur tag-basierte Behandlungsvorschriften oder beide Arten von Behandlungen angegeben sind. Nachdem diese Tests erfolgreich waren, wurde ein abschließender Test durchgeführt, der alle Module im Gesamten testet. Dafür wurde eine Datensammlung mit einem Datensatz, der für jede Verfügbarkeitsoption ein Datum enthält, anhand einer Policy aufbereitet, die alle Kombinationen von Verfügbarkeitsoptionen und Bindungen dieser beinhaltet. Anschließend wurden die aufbereiteten Daten unter der Annahme, dass die Voraussetzungen für die Entschlüsselung vorliegen, entschlüsselt und mit separat nur durch die Verfügbarkeitsoptionen verarbeiten Daten verglichen. Dieser Test war auch erfolgreich. Für das Entschlüsseln liegt im Unterordner *tests* des Ordners *data_analysis_center* das Pythonmodul `decrypt_data_appearances_in_analysis_center.py`. Diesem muss die Datei der aufbereiteten Daten, die Policy-Datei und die Anzahl der Datensätze der Klartextdatensammlung in dieser Reihenfolge übergeben werden. Anschließend wird der Ordner *decrypted_data_appearances* im gleichen Verzeichnis erstellt, der alle entschlüsselten Daten sortiert nach ihrer vorherigen Bindung enthält. Nachdem die Funktionalität der Module getestet wurde, wurde die Laufzeit und der Speicheraufwand dieser evaluiert. Dafür wurden Datensammlungen mit einer unterschiedlichen Anzahl an Datensätzen anhand einer Policy, die alle Kombinationen von Verfügbarkeitsoptionen und Bindungen dieser als tag-basierte Behandlungsvorschriften enthält, aufbereitet und es wurde dabei durch das Überwachungstool "Process Monitor" der Speicheraufwand und durch den Befehl `Measure-Command` in einer Windows PowerShell Konsole der Zeitaufwand gemessen. Die verwendete Datensammlung ist die gleiche, die bereits zur Evaluation des XML PolicyBuilders genutzt wurde und von der XML Transformation stammt. Diese Datensammlung enthält Datensätze, die Daten von Nachrichten von zwei fiktiven Personen beinhalten und für die unterschiedlichen Größen der Datensammlungen wiederholt wurden. Die Ergebnisse der Messungen sind in den Graphen der Abbildung 11 zu sehen. Bei den verwendeten Datensammlungen mit 5, 10 und 15 Datensätzen bleibt der verwendete Arbeitsspeicher relativ konstant. Dagegen verändert sich die Zeit mit einer gleichbleibenden bzw. linearen Steigung. Die gemessenen Werte für die Erstellung des Schlüsselmaterials durch das Modul `create_rsa_keys_and_exchange_rsa_public_key.py` (im Graph "create and exchange keys") und für die Aufbereitung der Daten anhand einer Policy durch das Modul `data_appearances_computation.py` (gleichnamig im Graph) sind sehr ähnlich. Durch den Verlauf der Steigung der Zeit ist zu erkennen, dass alleine die Aufbereitung von größeren als den hier genutzten Datensätzen noch viel länger bzw. länger als fünf Minuten dauern wird.

Da die benötigte Zeit auch stark von der genutzten Verfügbarkeitsoption abhängt, wurden Testdaten durch die Methoden der Verfügbarkeitsoptionen aufbereitet und dabei die benötigte

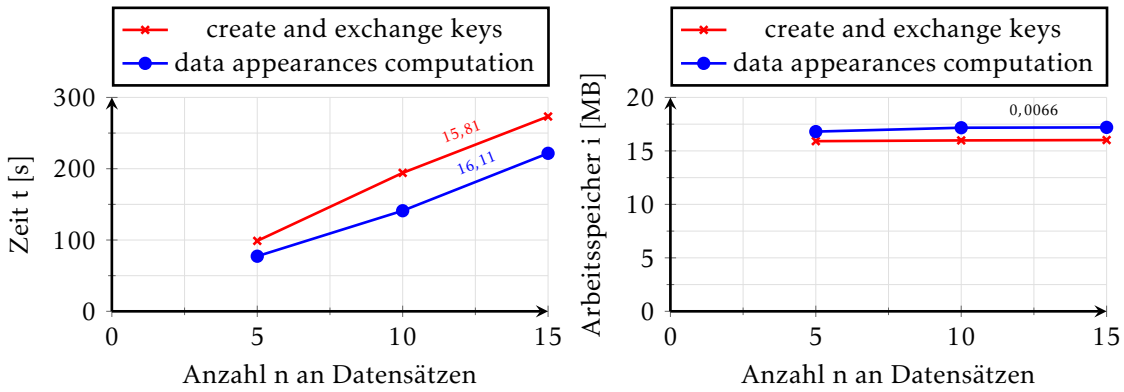


Abbildung 11.: *Die benötigte Zeit (links) und der verwendete Arbeitsspeicher (rechts) als erstes von dem Modul* `create_rsa_keys_and_exchange_rsa_public_key`*, durch das das benötigte Schlüsselmaterial des Data-Analysis-Center erstellt wird und als zweites von dem Modul* `data_appearances_computation`*, durch das Daten anhand einer zugehörigen Policy aufbereitet werden. Die Datensammlungen bestanden dabei aus 5, 10 und 15 Datensätzen und wurden mit einer Policy aufbereitet, in der als tag-basierte Behandlung jede Verfügbarkeitsoption in Kombination mit jeder Bindung genau einmal enthalten ist.*

Zeit gemessen. Die gemessenen Zeiten sind in der Tabelle 5 zu finden. Demnach ist zu erkennen, dass die Verfügbarkeitsoption der mathematischen Operation mit der Operation Addition mit Abstand am längsten dauert. Das liegt daran, da für die Erstellung des Pseudonyms die Klartextdaten durch das asymmetrische Verschlüsselungsverfahren Paillier verschlüsselt werden. Alle anderen Methoden benötigen weniger als eine Zehntelsekunde. Da die Methode der Aufdeckbarkeit nur den übergebenen Wert zurückgibt und keine Bindung stattfindet, war zu erwarten, dass diese Methode die wenigste Zeit benötigt.

Zusammenfassend ist zu erkennen, dass die Aufbereitung der Daten länger im Vergleich zum Laden der Daten im XML PolicyBuilder dauert und je nach der Anzahl an Datensätzen stark ansteigen kann. Das Gute dabei ist, dass die Steigung linear ist. Die Dauer hängt aber auch, wie oben gezeigt, von den verwendeten Verfügbarkeitsoptionen ab.

Verfügbarkeitsoption	**Zeit in Sekunden**
Mathematische Operation, Addition	3,6323s
Verkettbarkeit, Relation: Formaterhaltend mit Typ: Decimal	0,0433s
Verkettbarkeit, Relation: Formaterhaltend mit Typ: IPv6	0,0419s
Verkettbarkeit, Relation: Formaterhaltend mit Typ: IPv4	0,0208s
Verkettbarkeit, Relation: Formaterhaltend mit Typ: E-Mail	0,0108s
Verkettbarkeit, Relation: Formaterhaltend mit Typ: Integer	0,0060s
Verkettbarkeit, Relation: Suffixerhaltend mit Typ: E-Mail	0,0059s
Verkettbarkeit, Relation: Präfixerhaltend mit Typ: E-Mail	0,0058s
Mathematische Operation, Differenz	0,0023s
Verkettbarkeit, Relation: Gleichheit	0,0007s
Aufdeckbarkeit	0,0004s

Tabelle 5.: *Gemittelte Zeiten von fünf Messungen, die die Methoden der Verfügbarkeitsoptionen für die Aufbereitung von Testdaten benötigen, absteigend sortiert.*

7 Verwandte Arbeiten

Im folgenden werden verwandte Pseudonymisierungstools vorgestellt.

LiDSec [48] ist ein einfaches Pseudonymisierungs-Framework, dass ein Strukturidentifikationsmodul enthält. Dabei handelt es sich um die Komponente *Data Adapter*, die die Struktur von XML- und CSV-Dateien erkennen kann. Für weitere Formate, wie JSON, muss ein eigener Adapter implementiert werden. Der Eigentümer der Daten legt fest, wie seine Daten aufbereitet werden sollen. Dabei können Daten erhalten, gelöscht oder ersetzt werden. Ersetzen bedeutet dabei, dass ein Klartext durch einen vom Benutzer angegebenen Text ersetzt wird. Im Gegensatz zu LiDSec werden in dieser Arbeit Transformationen genutzt, um unterschiedliche Datenformate zu verarbeiten. Für jedes Format muss dabei nur einmal eine Transformation definiert werden. Dadurch gibt es keine Probleme bei der Identifikation der Struktur der Daten. Auch die Möglichkeiten, auf welche Weise die Daten aufbereitet werden können, sind im Framework dieser Arbeit vielfältiger. Alle nicht enthaltenen Datenfelder kommen auch nicht in den resultierenden aufbereiteten Daten vor und müssen vorher auch nicht betrachtet werden. Für ein Datum können mehr als nur ein Pseudonym erstellt werden. Durch die Bindungen der Verfügbarkeitsoptionen können die Pseudonyme zusätzlich geschützt werden.

FLAIM [49] ist dagegen ein Pseudonymisierungs-Framework, dass genauso wie das Framework dieser Arbeit modular aufgebaut und mehrere Optionen zur Aufbereitung von Daten anbietet. Es ist aber nur für Netzwerk- und Computer-Logdateien nutzbar und bietet keine Möglichkeit die Verfügbarkeit der Pseudonyme an eine Rolle bzw. einen Zweck zu binden.

Durch CryptDB [50] haben Popa et al. ein Framework für eine Datenbank vorgestellt, durch das trotz der Verschlüsselung der Daten Abfragen auf diesen ausgeführt werden können. Dabei können Daten einfach oder verschachtelt mit unterschiedlichen Verfahren verschlüsselt werden. Ein Datum kann dabei probabilistisch, durch die Option Random (RND), deterministisch, durch die Option Deterministic (DET), Reihenfolge-Erhaltend, durch die Option Order-preserving encryption (OPE), und additiv homomorph, durch die Option Homomorphic encryption (HOM), verschlüsselt werden. Für einen Spaltenübergreifenden Vergleich gibt es die Option Join und für unterteilte Daten die Option Word Search. Die Entschlüsselung ist an den Schlüssel eines Nutzers gebunden. Ein Proxy-Server ist für die Verschlüsselung der Daten und die Verarbeitung der Abfragen zuständig. Das Ziel ist die Vertraulichkeit der Daten, sodass trotzdem die meisten SQL-Anfragen durchgeführt werden können.
Für Google BigQuery, das ein Dienst für die Analyse großer Datenmengen ist, gibt es den "Encrypted BigQuery Client" [51, S. 445 ff.]. Dieser basiert auf der Idee von CryptDB. Dabei fehlen die Reihenfolge-Erhaltende Verschlüsselung. Daten werden, wie beim Framework dieser Arbeit, beim Datenhalter aufbereitet und dann weitergeleitet. Das Framework dieser Arbeit enthält bis auf die Reihenfolge-Erhaltende Verschlüsselung und die Verschlüsselung einzelner Wörter eines unterteilten Datums noch weitere Optionen, die beide Frameworke nicht enthalten. Im Gegensatz zu CryptDB und dem "Encrypted BigQuery Client" ist das Framework dieser Arbeit weiter eingeteilt in Verfügbarkeitsoptionen und Bindungen dieser, wodurch Behandlungsvorschriften

im Framework dieser Arbeit viel feiner angegeben werden können. Da für den Vergleich auf Gleichheit eine Hashfunktion zur Aufbereitung der Daten genutzt wird, besteht keine Gefahr, dass diese wieder entschlüsselt werden können. CryptDB und der "Encrypted BigQuery Client" können für größere Datenmengen eines Formats genutzt werden. Das Framework dieser Arbeit ist zwar nicht für große Datenmengen konzipiert, kann dagegen aber unterschiedliche Arten von Formate durch die Nutzung von Transformationen unterstützen. Pseudonyme können zusätzlich auch an einen Zweck gebunden werden. Für ein Datum kann es mehr als ein Pseudonym geben. Bei CryptDB und dem "Encrypted BigQuery Client" gibt es dagegen immer nur ein Pseudonym pro Datenfeld. Im Gegensatz zu diesen beiden muss im Framework dieser Arbeit eine Policy nicht vollständig sein, wodurch alle nicht enthaltenen Datenfelder nicht in den aufbereiteten Daten vorkommen. Als letztes gibt es bei CryptDB und dem "Encrypted BigQuery Client" keinen PolicyBuilder, durch den eine Policy bzw. in diesem Fall ein Datenbankschema einfach erstellt werden kann.

8 Zusammenfassung

Das Ziel dieser Bachelorarbeit war es, ein Toolkit für die Pseudonymisierung von Daten anhand einer Policy zu erstellen. Dazu wurden als erstes in Kapitel 2 grundlegende Begriffe und Verfahren aus den Bereichen Datenschutz und Datenverarbeitung definiert und erklärt. Zu diesen gehören personenbezogene Daten, die Bedeutung des Begriffs Datensammlung im Zusammenhang mit dieser Arbeit, Pseudonymisierung, Verfügbarkeitsoptionen und Bindung einer Verfügbarkeitsoption. Zu den Verfügbarkeitsoptionen, durch die Pseudonyme noch Teilinformationen anbieten können, gehören die Aufdeckbarkeit eines Klartextes, die Verkettbarkeit, durch die Pseudonyme bzgl. einer Relation verglichen werden können, und die Aufbereitung von Klartexten in der Hinsicht, dass die Möglichkeit besteht auf ihren Pseudonymen eine mathematische Operation durchführen zu können. Zu den Bindungen einer Verfügbarkeitsoption gehören die Bindung eines Datums an eine Rolle oder an einen Zweck.
Anschließend wurde eine Policysprache entwickelt. Dabei wurden sieben Anforderungen an die Policysprache identifiziert. Die Ziele dieser Anforderungen waren, dass die Policysprache flexibel und erweiterbar ist und dass nur benötigte Daten in einer Policy angegeben werden müssen. Aus diesem Grund wurden zwei Behandlungsarten von Datensätzen erarbeitet, durch die entweder Datenfelder von Datensätzen individuell oder gleiche Datenfelder in allen Datensätze gleich pseudonymisiert werden können. Ein Ausblick auf weitere mögliche Arten wurde auch gegeben. Eine weitere Anforderung war, dass die Policysprache für unterschiedliche Formate von Datensätzen verwendet werden kann. Aus diesem Grund wurden die verbreiteten Formate CSV, XML und die Standards für den Austausch und die Haltung von Gesundheitsdaten der HL7 Organisation betrachtet. Dabei ist aufgefallen, dass diese Formate unterschiedliche Strukturen aufweisen. Um in dieser Hinsicht flexibel zu sein, wurde die Methode der Transformation von Datensätzen gewählt, durch die Datensammlungen in ein einheitliches Format gebracht werden, damit die weiteren Schritte der Verarbeitung immer gleich bleiben können. Als Format einer Policy und einer einheitlichen Datensammlung wurde das XML-Format gewählt.
Für die einfache Erstellung einer Policy wurde das Tool der XML PolicyBuilder entworfen und implementiert. Es ist für die Flexibilität modular aufgebaut. Beim Entwurf der graphischen Benutzeroberfläche wurde u.a. auf die Benutzerfreundlichkeit Wert gelegt.
Um abschließend eine Datensammlung zu pseudonymisieren wurde das Pseudonymisierungstool aus [5] erweitert und optimiert, damit dieses alle erarbeiteten Funktionalitäten anbietet. Die in der Policysprache definierten Verfügbarkeitsoptionen und Bindungen dieser entsprechen denen in den Grundlagen beschriebenen. Für die Verfügbarkeitsoption der Verkettbarkeit wurden acht Relationen und für die Verfügbarkeitsoption der mathematischen Operation wurden zwei Operationen definiert, die auch vom PolicyBuilder und Pseudonymization Framework unterstützt werden.
Anschließend wurde eine Evaluation durchgeführt, bei der als erstes die Funktionssicherheit durch Tests und als zweites die Laufzeit und der Speicheraufwand der drei implementierten Komponenten überprüft wurden. Für das letztere wurden Messungen mit unterschiedlich großen Datensammlungen durchgeführt, bei denen die Zeit und der verwendete Arbeitsspeicher

gemessen wurden. Die Steigungen der Kurven (bestimmt durch die Messwerte) war dabei positiv und entweder linear oder fallend. Als letztes wurde das erstellte Toolkit in den aktuellen Stand der Forschung durch den Vergleich mit verwandten Arbeiten eingeordnet.

Zusammenfassend wurden alle Ziele dieser Arbeit erfüllt. Das Resultat ist ein Pseudonymisierungs-Toolkit durch das Daten so aufbereitet werden können, dass sie Verfügbarkeitsoptionen anbieten und für den weiteren Schutz der Pseudonyme an eine Rolle oder einen Zweck gebunden werden können.

8.1 Zusammenhang der Komponenten des Frameworks

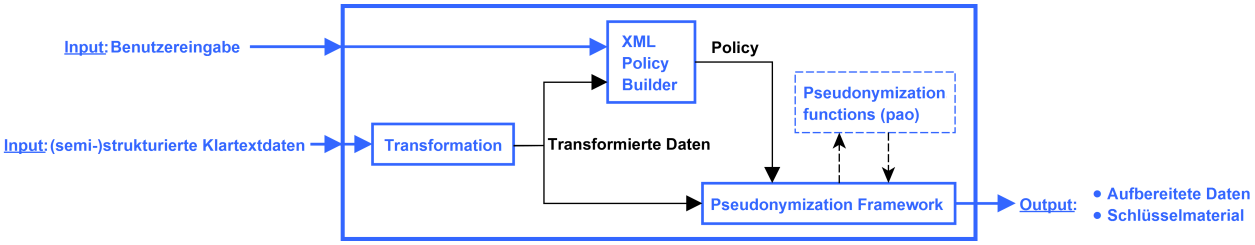


Abbildung 12.: *Gesamter Datenfluss des Frameworks dieser Arbeit.*

Der gesamte Ablauf der Nutzung des Toolkits bzw. seiner Komponenten wird im folgenden zusammengefasst und ist in der Abbildung 12 zu sehen. Als erstes muss eine Datensammlung übergeben werden. Diese wird durch eine Transformation in das einheitliche Datenformat umgewandelt. Damit diese Daten verarbeitet werden können, kann im nächsten Schritt eine Policy durch den XML PolicyBuilder erstellt werden. Einerseits für die Unterstützung der Erstellung von tag-basierten Behandlungsvorschriften der Datensätze und andererseits für die Nutzung von individuellen Behandlungsvorschriften kann durch den XML PolicyBuilder die umgewandelte Datensammlung geladen werden. Nachdem eine Policy durch eine Benutzereingabe erstellt wurde, kann sie im Pseudonymization Framework als Konfiguration für die Verarbeitung der Datensammlung genutzt werden. Während der Verarbeitung einer Datensammlung werden u.a. die Pseudonymisierungsfunktionen des Packages `pao` [6] genutzt. Die Ausgabe des Pseudonymization Frameworks besteht aus den aufbereiteten Daten und dem Schlüsselmaterial, das durch die Verwendung der Bindungen entsteht.

8.2 Ausblick

Während dieser Bachelorarbeit konnten weitere Möglichkeiten zur Verbesserung und Erweiterung des Toolkits identifiziert werden. Dazu gehört als erstes, dass die erarbeitete Policysprache um Behandlungen erweitert werden kann. Ein Ausblick auf weitere mögliche Behandlungen wurde bereits in einem Exkurs in Kapitel 3.2 gegeben. Als zweites kann, wenn größere als die hier betrachteten Datensammlungen verarbeitet werden sollen, der Speicheraufwand des XML PolicyBuilders durch z.B. das Parsen einer Datensammlung, ohne die gesamte Datensammlung in den Arbeitsspeicher zu laden, verringert werden. Auch die Laufzeit des Pseudonymization Frameworks könnte durch eine Verbesserung, durch die eine Verarbeitung auf mehreren Prozessorkernen stattfindet, beschleunigt werden. Als letztes könnte eine weitere große Aufgabe die Erstellung eines Tools sein, durch dass eine Riskikoabschätzung einer Policy durchgeführt werden könnte.

9 Literaturverzeichnis

[1] *BSI - CERT-Bund*. – https://www.bsi.bund.de/EN/Topics/IT-Crisis-Management/CERT-Bund/cert-bund_node.html (Abruf: 05.08.2016)

[2] *FIRST.org / FIRST - Improving security together*. – https://www.first.org/ (Abruf: 05.08.2016)

[3] Elektronische Patientenakte: Techniker Krankenkasse will auf Fitnessdaten zugreifen. (2016), Februar. – http://www.spiegel.de/gesundheit/ernaehrung/techniker-krankenkasse-will-fitnessdaten-nutzen-a-1076388.html (Abruf: 05.08.2016)

[4] vom Dorp, J.: Lab Report: Using XACML for the Definition of Availability Policies for Data Pseudonymization. (2015). – Rheinische Friedrich-Wilhelms-Universität Bonn, Institut für Informatik

[5] Sierra, S. V.: Framework Implementation: Availability Policies for Data Pseudonymization. (2016). – Rheinische Friedrich-Wilhelms-Universität Bonn, Institut für Informatik

[6] Faiz, S. ; Wehner, M.: Praktikum: Erweiterung eines Packages um Funktionen für Pseudonymisierung mit Verfügbarkeitsoptionen. (2016). – Rheinische Friedrich-Wilhelms-Universität Bonn, Institut für Informatik

[7] *OASIS eXtensible Access Control Markup Language (XACML) TC*. – http://www.oasis-open.org/committees/xacml (Abruf: 06.08.2016)

[8] *Verordnung (EU) 2016/679 des Europäischen Parlaments und des Rates vom 27. April 2016 zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, zum freien Datenverkehr und zur Aufhebung der Richtlinie 95/46/EG (Datenschutz-Grundverordnung) (Text von Bedeutung für den EWR)*. – http://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:32016R0679 (Abruf: 16.11.2016)

[9] Berman, J.J.: *Principles of Big Data: Preparing, Sharing, and Analyzing Complex Information*. 1st. Morgan Kaufmann Publishers Inc., (2013). – ISBN 9780124045767

[10] *Bundesdatenschutzgesetz (BDSG) §3 Weitere Begriffsbestimmungen*. – http://www.gesetze-im-internet.de/bdsg_1990/__3.html (Abruf: 06.08.2016)

[11] Pfitzmann, A. ; Hansen, M.: A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. (2010)

[12] Kasem-Madani, S. ; Meier, M.: Definition of Availability Policies for Data Pseudonymization Using XACML. In: *IFIP Summer School on Privacy and Identity Management* (2015), August

[13] Flegel, U.: *Privacy-Respecting Intrusion Detection*. Springer US, (2007) (Advances in Information Security). – ISBN 9780387682549

[14] *Extensible Markup Language (XML) 1.0 (Fifth Edition) W3C Recommendation 26 November 2008*. – https://www.w3.org/TR/2008/REC-xml-20081126/ (Abruf: 22.09.2016)

[15] Bizer, J.: Sieben goldene Regeln des Datenschutzes. In: *Datenschutz und Datensicherheit-DuD* 31 (2007), Nr. 5, S. 350–356

[16] Shafranovich, Y.: *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. RFC 4180 (Informational). http://www.ietf.org/rfc/rfc4180.txt. Version: Oktober 2005 (Request for Comments). – Updated by RFC 7111

[17] *passwd(5) - Linux manual page*. – http://man7.org/linux/man-pages/man5/passwd.5.html (Abruf: 28.09.2016)

[18] Abiteboul, S. ; Buneman, P. ; Suciu, D.: *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, (2000) (Data Management Systems Series). – ISBN 9781558606227

[19] Bry, F. ; Kraus, M. ; Olteanu, D. ; Schaffert, S.: Aktuelles Schlagwort "Semi-strukturierte Daten". (2001), Juni. – Universität München, Institut für Informatik, Lehr- und Forschungseinheit für Programmier- und Modellierungssprachen

[20] Ben-Kiki, O. ; Evans, C. ; Ingerson, B.: *YAML Ain't Markup Language (YAML) (tm) Version 1.2*. Oktober 2009. – http://yaml.org/spec/1.2/spec.html (Abruf: 22.09.2016)

[21] Haas, P.: *Medizinische Informationssysteme und Elektronische Krankenakten*. Springer Berlin Heidelberg, (2005). – ISBN 9783540268550

[22] Li, Ke ; Xu, Jin ; Li, Jiang-Xiong ; Huang, Guo-Sheng ; Zhou, Ming ; Qiao, Cong-Cong ; Wang, Hai-Sheng: A data model of EHR storage based on HL7 RIM. In: *Journal of electronic science and technology* 10 (2012), Nr. 4, S. 334–341

[23] HL7 Deutschland e.V.: *HL7 CDA - Clinical Document Architecture: HL7 Deutschland e.V.*. – http://hl7.de/themen/hl7-cda-clinical-document-architecture/ (Abruf: 29.09.2016)

[24] HL7 Deutschland e.V.: *HL7 V3 RIM - das Referenzinformationsmodell: HL7 Deutschland e.V.*. – http://hl7.de/themen/hl7-v3-rim-das-referenzinformationsmodell/ (Abruf: 29.09.2016)

[25] Aspden, P. ; Corrigan, J. M. ; Wolcott, J. ; Erickson, S. M. u. a.: *Patient Safety: Achieving a New Standard for Care*. Washington, D.C. : National Academies Press, (2004). – ISBN 0–309–09077–6

[26] HL7 Deutschland e.V.: *HL7 v2.x Nachrichten: HL7 Deutschland e.V.*. – http://hl7.de/themen/hl7-v2x-nachrichten/ (Abruf: 29.09.2016)

[27] Heitmann, K. U. ; Biron, Co-Chair P. V.: XML encoding rules for HL7 v2 messages. v2.xml Revision 2.361 (1999). – Universität zu Köln, Deutschland

[28] Dolin, R. H. ; Alschuler, L. ; Beebe, C. ; Biron, P. V. ; Boyer, S. L. ; Essin, D. ; Kimber, E. ; Lincoln, T. ; Mattison, J. E.: The HL7 clinical document architecture. In: *Journal of the American Medical Informatics Association* 8 (2001), Nr. 6, S. 552–569

[29] Machanavajjhala, A. ; Kifer, D. ; Gehrke, J. ; Venkitasubramaniam, M.: l-diversity: Privacy beyond k-anonymity. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1 (2007), März

[30] Sweeney, L.: k-anonymity: A model for protecting privacy. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10 (2002), Nr. 05, S. 557–570

[31] *XSL Transformations (XSLT) - Version 1.0 - W3C Recommendation 16 November 1999*. – https://www.w3.org/TR/xslt (Abruf: 03.11.2016)

[32] *Conceito: Entity-Control-Boundary Pattern*. – http://epf.eclipse.org/wikis/openuppt/openup_basic/guidances/concepts/entity_control_boundary_pattern,_uF-QYEAhEdq_UJTvM1DM2Q.html (Abruf: 25.10.2016)

[33] Gamma, E. ; Johnson, R. ; Helm, R. ; Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, (1994). – ISBN 978–0–201–63361–0

[34] *19.7. xml.etree.ElementTree - The ElementTree XML API - Python 2.7.12 documentation*. – https://docs.python.org/2/library/xml.etree.elementtree.html (Abruf: 26.10.2016)

[35] *19.9. xml.dom.minidom - Minimal DOM implementation - Python 2.7.12 documentation*. – https://docs.python.org/2/library/xml.dom.minidom.html (Abruf: 26.10.2016)

[36] *8.3. collections - High-performance container datatypes - Python 2.7.12 documentation*. – https://docs.python.org/2/library/collections.html#collections.OrderedDict (Abruf: 27.10.2016)

[37] *TkInter - Python Wiki*. – https://wiki.python.org/moin/TkInter (Abruf: 26.10.2016)

[38] Nielsen, J.: *Usability Engineering*. Elsevier Science, (1994) (Interactive Technologies). – ISBN 9780080520292

[39] Consortium, World Wide W.: *Script direction and languages*. – https://www.w3.org/International/questions/qa-scripts.en (Abruf: 01.11.2016)

[40] Paillier, Pascal: Public-key cryptosystems based on composite degree residuosity classes. In: *International Conference on the Theory and Applications of Cryptographic Techniques* Springer, (1999), S. 223–238

[41] Kasem-Madani, Saffija: A Mechanism Design for Privacy-Preserving Computation on Shared Data. In: *GI Edition Proceedings Band 256 Sicherheit 2016* Gesellschaft für Informatik e.V. (GI), (2016). – ISBN 9783885796503, S. 191–196

[42] Kerschbaum, Florian: Distance-preserving pseudonymization for timestamps and spatial data. In: *Proceedings of the 2007 ACM workshop on Privacy in electronic society* ACM, (2007), S. 68–71

[43] Stevens, Marc: Fast Collision Attack on MD5. In: *Cryptology ePrint Archive* (2006)

[44] *NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition*. Oktober 2012. – https://www.nist.gov/news-events/news/2012/10/nist-selects-winner-secure-hash-algorithm-sha-3-competition (Abruf: 06.11.2016)

[45] Russinovich, M.: *Process Monitor v3.20*. – https://technet.microsoft.com/de-de/sysinternals/processmonitor.aspx (Abruf: 10.11.2016)

[46] *Evaluating Memory and Cache Usage*. – https://technet.microsoft.com/en-us/library/cc958292.aspx (Abruf: 10.11.2016)

[47] *Server-Logfile von redlug.com*. – http://redlug.com/logs/access.log (Abruf: 29.06.2016)

[48] Rawassizadeh, R. ; Heurix, J. ; Khosravipour, S. ; Tjoa, A. M.: LiDSec: A Lightweight Pseudonymization Approach for Textual Personal Information. In: *Sixth International Conference on Availability, Reliability and Security (ARES)*, (2011), S. 603–608

[49] Slagell, A. ; Lakkaraju, K. ; Luo, K.: FLAIM: A multi-level anonymization framework for computer and network logs. In: *Proceedings of the 20 th USENIX Large Installation System Administration Conference*, (2006), S. 63–77

[50] Popa, R. A. ; Redfield, C. ; Zeldovich, N. ; Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* ACM, (2011), S. 85–100

[51] Tigani, J. ; Naidu, S.: *Google BigQuery Analytics*. John Wiley & Sons, (2014)

A Anhang

A.1 Protokoll: Manueller GUI-Test vom XML-PolicyBuilder

A.1.1 Setup für Tests:

1. Starte XML-PolicyBuilder: **python** -m XML_PolicyBuilder.View.multi_window

A.1.2 Test: Schliessen ohne Nachfrage

Klicke im Menü auf: "File" → "Quit"
⇒ Anwendung schließt sich sofort.

A.1.3 Test: Schliessen mit Nachfrage

Klicke auf den "X"-Knopf vom Hauptfenster oder drücke die Taste "ESC":
⇒ Kleines Fenster mit Nachfrage: "Do you really want to quit?" öffnet sich.

1. Klick auf "OK"
   ⇒ Anwendung schließt sich sofort.
2. Klick auf "Abbrechen"
   ⇒ Kleines Fenster schließt sich.

A.1.4 Test: ?

1. Klicke im Menü auf: "?" → "About..." Oder drücke die Taste "F1"
   ⇒ Erwartet wird, dass sich ein About-Fenster öffnet.
2. Wenn About-Fenster geöffnet: Kein weiteres About-Fenster kann geöffnet werden und es kann erst weitergearbeitet werden, wenn About-Fenster geschlossen wird.

A.1.5 Test: Neue Policy erstellen

Klicke im Menü auf: "File" → "New Policy"
⇒ Im Fenster erscheint der Frame zum Bearbeiten einer Policy.

A.1.6 Test: Existierende Policy laden

Klicke im Menü auf: "File" → "Open Policy"
⇒ Im Fenster erscheint der Frame zum Bearbeiten der Policy.

A.1.7 Test: Policy bearbeiten

(Test 1.5 oder 1.6 muss vorher durchgeführt werden.)

Test: "Save Policy" und "Save Policy as..." vorhanden?

Im Untermenü "File" sind zwei Menüpunkte dazugekommen:

1. Save Policy
2. Save Policy as...

Test: "Save Policy"

Klicke im Menü auf: "File" → "Save Policy"

1. Wenn es noch keine Policy-Datei gibt ⇒ Verhalten wie bei *"Test: "Save Policy as..." "*
2. Wenn es schon eine Policy-Datei gibt ⇒ Policy wurde gespeichert in der in der InfoBar angegebenen Datei(linke) und in StatusBar steht "Saved Policy.."

Test: "Save Policy as..."

Wenn im Menü auf: "File" → "Save Policy as..." geklickt wird:
⇒ "Speichern unter"-Fenster öffnet sich

a) Gebe Name für die Policy-Datei an und klicke auf "Speichern"
   ⇒ Policy wurde gespeichert, in der InfoBar erscheint links der Dateiname und in der StatusBar steht "Saved Policy.."
b) Gebe keinen Namen an und klicke auf "Abbrechen"
   ⇒ Alles bleibt wie vor diesem Test. Keine Änderung in Info- oder StatusBar.

Test: Erneutes "New Policy" und "Open Policy"

Führe Test 1.5 und 1.6 erneut durch.

Test: Lade Datensatz

Wenn noch kein Datensatz geladen wurde:
Klicke im Tab "Tag-based" auf "Load tags from Data" oder im Tab "Individual" auf "Load Data"
⇒ "Öffnen"-Fenster erscheint.

a) Wähle Datensatz-Datei aus und klicke auf "Öffnen"
   ⇒ Datensatz wird geladen: In Statusbar steht "Data loaded.. <*time*>" und in InfoBar steht rechts der Name der Datensatz-Datei.
   - Die Buttons im Tab "Tag-based" "Load tags from Data" und im Tab "Individual" "Load Data" sind verschwunden.
   - Die Comboboxen in den jeweiligen Tabs enthalten Tags und Datasets.
   - Die Comboboxen sind leer und in der "Treatments"-Liste steht "No treatments found".

b) Wähle keine Datensatz-Datei aus und klicke auf "Abbrechen"
   ⇒ Alles bleibt wie vor diesem Test. Keine Änderung in Info- oder StatusBar.

Test: "Tag-based" Füge Tag manuell hinzu.

Klicke auf den Button mit dem "+"-Symbol
⇒ "Add tags manually.."-Fenster öffnet sich.

a) Schreibe mehrere Tags kommagetrennt mit und ohne Leerzeichen in Entry und klicke auf "OK".
   ⇒ Tags sind in "Tags"-Combobox auswählbar. Ohne Duplikate!

b) Schreibe einen Tag in Entry und klicke auf "Cancel"
   ⇒ Kein Tag wurde zur "Tags"-Combobox hinzugefügt.

c) Füge keine Tags ein und klicke auf "OK" oder "Cancel"
   ⇒ Kein Tag wurde zur "Tags"-Combobox hinzugefügt.

Test: "Tag-based" Füge Treatment für Tag hinzu.

1. Wähle Tag in der "Tags"-Combobox aus, wähle eine Verfügbarkeitsoption und ein Binding aus.
   ⇒ Wenn noch kein Treatment vorhanden: "No treatments found" in "Treatments"-Liste, ansonsten die Treatments.
2. Klicke auf "Add" um Treatment hinzuzufügen.
   ⇒ Treatment erscheint in "Treatments"-Liste.

Test: "Individual" "Tags"-Combobox wird aufgefüllt?

Wähle ein Dataset in der "Datasets"-Combobox aus.
⇒ "Tags"-Combobox enthält tags

Test: "Individual" Füge Treatment hinzu.

1. Wähle in der "Datasets"-Combobox und in der "Tags"-Combobox einen Eintrag aus.
   ⇒ Inhalt von der Auswahl erscheint im "Tag Content"-Label.
   Wenn noch kein Treatment vorhanden: "No treatments found" in "Treatments"-Liste, ansonsten die Treatments.
2. Wähle eine Verfügbarkeitsoption und ein Binding aus und klicke auf "Add" um Treatment hinzuzufügen.
   ⇒ Treatment erscheint in "Treatments"-Liste.

Test: Lösche Treatment

Erstens wähle im Tab "Tag-based" ein tag in der "Tags"-Combobox oder im Tab "Individual" ein Dataset und einen Tag in der jeweiligen Combobox aus.
Zweitens wähle in der "Treatments"-Liste ein Treatment aus und klicke auf den "Delete"-Knopf.
⇒ Treatment in der "Treatments"-Liste verschwindet.

Test: Verfügbarkeitsoptionen sind vorhanden?

Die Combobox bei "Availability Option" enthält Einträge?

Test: Bindings sind vorhanden?

Die Combobox bei "Binding" enthält Einträge?

Test: Zwei Bindings können als Treatment hinzugefügt werden?

1. Wähle eine Verfügbarkeitsoption, jeweils ein Binding und wechsle zwischen den Bindings durch die Knöpfe "No. 1" und "No. 2".
   ⇒ Die Bindings bleiben vorhanden.
2. Klicke auf "Add" um Treatment hinzuzufügen.
   ⇒ Treatment mit zwei Bindings erscheint in "Treatments"-Liste.
   ⇒ Binding "No. 2" ist wieder leer.