Fehlerklassen in XML Daten und Erzeugung „schmutziger“ XML Daten

– Studienarbeit –

Sven Puhlmann

17. Februar 2004

Betreuer: Prof. Dr. Felix Naumann

Inhaltsverzeichnis

Kapitel 1

Einleitung

In den letzten Jahren hat die Entwicklung von verteilten Informationssystemen stark an Bedeutung gewonnen. Auf der einen Seite ist eine große Anzahl autonomer Datenbanksysteme entstanden, bei denen im Nachhinein ein Bedarf zur Integration der Daten beispielsweise in Data Warehouses besteht. Auf der anderen Seite werden gezielt autonome, verteilte und heterogene Datenbanksysteme entworfen, um etwa bestimmte Bereiche eines großen Konzerns abzugrenzen - auch hier soll dann in bestimmten Fällen eine Datenintegration vorgenommen werden.

Dabei ist die Wahrscheinlichkeit, dass sich bei einer Integration dieser Daten bestimmte Datenmengen überschneiden und Datenfehler auftreten, sehr groß. Zudem können Daten bereits vor ihrer Integration Datenfehler aufweisen, die erst im Zusammenhang mit Daten aus anderen Datenquellen auffällig werden - selbst wenn diese unabhängigen Datenquellen zunächst sauber erscheinen.

Data Cleansing-Algorithmen und -Methoden werden verwendet, um Daten, die durch Methoden der Datenintegration fehlerhaft wurden, zu reinigen. Damit diese Methoden eingehend getestet werden können, werden Benchmarks benötigt. Dabei handelt es sich um eine Menge von Daten, die bestimmte vordefinierte Fehler aufweisen. Für relationale Daten existieren bereits Programme, die Tabellen erzeugen und mit Datenfehlern versehen.

Gleichzeitig mit der Verbreitung von verteilten Informationssystemen ist auch der Anteil an vorhandenen XML Daten gestiegen, die sowohl physisch als XML Dateien vorliegen als auch von einer Schnittstelle als XML Datenstrom geliefert werden können. Damit stieg der Bedarf, vorhandene XML Datenquellen in Informationssystemen zu integrieren. Die dabei entstehenden Fehler sind zum Teil ähnlich den Fehlern integrierter relationaler Datenbanksysteme, unterscheiden sich jedoch auch in einigen Punkten. Die Aufgabe dieser Studienarbeit war es, zunächst Fehlerklassen für relationale Daten und ihre Anwendung auf geschachtelte XML Daten zu untersuchen und anschliessend ein Tool zu entwickeln, dass parametrisiert verschmutzte XML Daten erzeugt. Als Basis der Entwicklung sollte ToXgene dienen, ein Tool zur Generierung normaler, unverschmutzter XML Daten.

Das folgende Kapitel beschäftigt sich mit den Fehlern, die in integrierten XML Daten auftreten können – ausgehend von den Fehlerklassen relationaler Daten. Das dritte Kapitel beinhaltet alles über die DirtyXMLFactory, dem Tool zur Erzeugung schmutziger XML Daten. Es werden darin sowohl seine Verwendung und die Benutzung der Parameter als auch Vorgänge im Programm und dessen Erweiterungsmöglichkeiten aufgezeigt.

Kapitel 2

Fehler in XML Daten

In diesem Kapitel werden die Fehlerklassen relationaler Daten vorgestellt und ihre Anwendung auf geschachtelte XML Daten untersucht.

2.1 Überblick

Nach D. Tonn [1] können Fehler relationaler Daten in die folgenden vier Fehlerklassen unterschieden werden:

1. Fehlerhafte Attributwerte,
2. Duplikate,
3. Inkonsistenzen und
4. Fehlende Datensätze.

Diese Fehlerklassen lassen sich auf unterschiedliche Weise auf geschachtelte XML Daten anwenden. Da die Informationen in XML Daten in anderer Weise vorliegen als in relationalen Datenbanken, ist meiner Meinung nach die Bezeichnung der Fehlerklasse "Fehlerhafte Attributwerte" irreführend. Während in relationalen Datenbanken als Attributwert ein Datum angesehen wird, das ein Attribut in der Extension seiner Relation (ohne Bezug auf andere Daten zu nehmen) enthält, ist ein Attributwert in XML Daten nur der Datenwert eines Attributes von einem Element. Da jedoch in XML auch Daten direkt in Elementen enthalten sein können (dabei handelt es sich um den Inhalt von Textknoten), kann man den Namen dieser Fehlerklasse für XML Daten in *Fehlerhafte Textknoten und Attributwerte* ändern. Die Namen der anderen Fehlerklassen können auf XML Daten angewendet werden. Im Folgenden werde ich auf die einzelnen Fehlerklassen in Bezug auf XML Daten ausführlicher eingehen.

2.2 Fehlerhafte Textknoten und Attributwerte

Die Werte von Attributen relationaler Daten können Schreibfehler, unvollständige Werte, NULL-Werte, Dummy-Eintragungen und veraltete Daten enthalten. Während Schreibfehler, unvollständige Werte und veraltete Daten in derselben Art und Weise auch in XML Daten auftreten können, gibt es Unterschiede bei den NULL-Werten und Dummy-Eintragungen.

In XML Daten gibt es keine Typisierung. So existiert auch kein NULL-Wert; es gibt nur verschiedene Möglichkeiten, was als NULL-Wert gesehen und damit als Fehler interpretiert werden könnte. Während bei relationalen Daten für ein Attribut in einer Extension immer ein Attributwert auftreten muss (selbst wenn z.B. bei Nichtkenntnis von Daten eben ein NULL-Wert verwendet wird), besteht in XML Daten die Möglichkeit, ein Attribut oder ein Element einfach auszulassen, sofern dies nicht durch eine zugrunde liegende Beschreibung der XML Daten verboten wird - zum Beispiel durch eine Document Type Definition (DTD) oder ein XML Schema. Dabei fällt sofort die Wichtigkeit dieser Beschreibungen auf, mit denen Fehler dieser Art reduziert werden können. Bestehen gar keine Vorgaben, wird es unter Umständen extrem schwer fallen, einen Fehler als solchen zu erkennen und herauszufiltern. Mit einer DTD können für XML Daten Elemente und Attribute, ihre Anzahl und auch ihr Inhalt mit gewissen Einschränkungen vordefiniert werden, und mit der verbesserten Variante, dem XML Schema, können weitere Einschränkungen, unter anderem Muster für die Syntax von Zeichenketten, gewählt werden. Ansonsten besteht neben dem Weglassen von Elementen und Attributen auch noch die Möglichkeit, deren Inhalt leer zu lassen oder bestimmte subjektiv definierte Dummy-Werte zu verwenden.

Dummy-Eintragungen können prinzipiell in XML Daten ebenso auftreten, wie es in relationalen Daten möglich ist. Ein Dummy-Wert wird jedoch meist nur dann verwendet werden, wenn eine Beschreibung für die Daten vorliegt und ein Datum eingetragen werden muss. Eine Vermeidung von Dummy-Eintragungen ist nur dann möglich, wenn es in der Beschreibung eine begrenzte Anzahl vordefinierter Inhalte für ein Element oder Attribut gibt (z.B. nur die Möglichkeiten "ja" und "nein").

2.3 Duplikate

Duplikate entstehen meist durch die Integration verschiedener heterogener Datenquellen, können jedoch auch unabhängig von einer Integration in einzelnen Datenquellen auftreten. Ein Beispiel dafür wäre das Anlegen eines neuen Datensatzes, egal ob in relationalen oder geschachtelten XML Daten, mit anderer Schreibweise eines oder mehrerer Textobjekte, obwohl dieser Datensatz semantisch bereits vorhanden ist. Der vorhandene Datensatz wird vielleicht aufgrund seiner anderen Syntax nicht gefunden und kann somit nicht ergänzt werden.

Die Fehlerklasse der Duplikate in relationalen Daten lässt sich folgendermassen untergliedern:

- semantisch identische Einträge,

- semantisch sich ergänzende Einträge und
- mehrfache Einträge mit einer Mischung aus wahren und falschen Fakten.

Diese Unterteilung trifft natürlich ebenso auf geschachtelte XML Daten zu. Allerdings gibt es unabhängig vom Dateninhalt der Duplikate bedeutende Unterschiede in der Art der Darstellung der Daten. Während bei relationalen Daten zusammengehörende Duplikate typischerweise in ein und derselben Relation vorkommen und jedes Duplikat dieselbe Anzahl an Datenwerten besitzen muss (auch wenn es NULL-Werte sind), sind Duplikate in geschachtelten XML Daten weitaus komplexer. In den semantisch identischen oder sich (teilweise) ergänzenden Einträgen können bei XML Daten

- mehr oder weniger Elemente oder Attribute auftreten,
- Elemente und Attribute syntaktisch verschieden aber semantisch gleich sein oder
- trotz semantischer Identität oder teilweiser Ergänzung die Repräsentationen der XML Daten teilweise oder völlig verschieden sein.

Man kann hieraus erahnen, wie schwierig und umfassend die Implementierung von Data Cleansing Algorithmen für die Integration von XML Datenquellen ist. Wenn Beschreibungen der XML Daten etwa in Form von DTDs oder XML Schemas vorhanden sind, ist es möglich, ähnlich wie bei relationalen Daten vor der Integration der Daten eine Schemaintegration vorzunehmen. Fehlen jedoch Beschreibungen teilweise oder ganz, kann in manchen Fällen sicher nur manuelle Unterstützung bei der Entscheidungsfindung helfen.

Im Mittelpunkt bei der Entwicklung des Tools zur Erzeugung schmutziger XML Daten, das im dritten Kapitel noch ausführlich vorgestellt wird, stand die Erzeugung von Duplikaten. Das Resultat der Ausführung dieses Tools enthält Duplikate, bei denen sich die grundlegende Syntax und Schachtelung nicht von den Originalen unterscheidet. Damit können Benchmarks mit den aktuell verfügbaren Algorithmen zum Data Cleansing durchgeführt werden.

2.4 Inkonsistenzen

Bei relationalen Daten kann die Fehlerklasse der Inkonsistenzen in Inkonsistenzen innerhalb eines Tupels und Inkonsistenzen zwischen Tupeln eingeteilt werden. Bei Inkonsistenzen innerhalb eines Tupels handelt es sich um semantische Unstimmigkeiten in diesem Tupel – so können bei einer Person das Alter und das Geburtsdatum im Wiederspruch zueinander stehen. Inkonsistenzen zwischen Tupeln können in Inkonsistenzen aufgrund unterschiedlicher Repräsentationen der Daten und in Inkonsistenzen aufgrund von Integritätsfehlern unterteilt werden.

Unterschiedliche Repräsentationen können bei der Zusammenführung von Daten aus mehreren Datenbanken auftreten, wobei für semantisch gleiche Relationen syntaktisch mehr oder weniger stark abweichende Schematas in den

unterschiedlichen Datenbanken vorliegen[1]. Ein Beispiel dafür wären Versandhändler für Bücher: Während bei dem einen das Attribut „lieferbar" immer durch „j" oder „n" repräsentiert wird, hat der andere Händler bei der Erstellung des Datenbank-Schemas dafür die Zahlen 1 und 0 gewählt.

Integritätsprobleme treten in relationalen Datenbanken meist zwischen verschiedenen Relationen auf. So können vorhandene Daten bestimmte Regeln verletzen oder Verweise von Fremdschlüsseln auf andere Relationen fehlen, so dass bei einem Join über einen dieser Fremdschlüssel die Kardinalität der Ergebnismenge von Tupeln geringer ist als erwartet.

Die eben aufgezeigten Inkonsistenzen relationaler Daten sind ebenfalls auf geschachtelte XML Daten anwendbar. Wie jedoch weiter oben bereits angedeutet, können XML Daten im Gegensatz zu relationalen Daten auch ohne ein Schema, das die Daten beschreibt und die Komplexität der Daten begrenzt, vorliegen. Die XML Daten sind dann wohlgeformt, werden aber bei der Erstellung nicht validiert, sodass eine Gültigkeit dieser Daten nicht festgestellt werden kann. Wenn den XML Daten ein XML Schema zugrunde liegt, dann kann die Menge an Inkonsistenzen zwischen Tupeln, wenn als Inkonsistenz eine unterschiedliche Repräsentation der Daten angenommen wird, geringer sein, als wenn für die Daten nur eine DTD oder gar keine Beschreibung vorhanden ist.

Ebenfalls eine große Rolle spiel die Herkunft der XML Daten. So können weder in DTDs noch in XML Schema Regeln definiert werden, die Integritätsprobleme vermeiden. Werden die Daten jedoch über eine grafische Oberfläche eingegeben, dann kann diese bestimmte Integritätsregeln überprüfen. So kann sichergestellt werden, dass XML Datenquellen, die für eine Datenintegration verwendet werden, keine Verletzungen der Integrität aufweisen und dass die Menge der Integritätsfehler der zusammengeführten Daten möglichst gering ist.

Einen wesentlichen Unterschied zwischen relationalen und XML Daten gibt es in der Verwendung von Schlüsselattributen. Während in einer Relation oft ein Attribut als Schlüsselwert dieser Relation gewählt wird, kann eine Menge von XML Daten nur einmalig auftretende Werte besitzen, wenn für diese Menge eine Beschreibung durch eine DTD oder ein XML Schema vorhanden ist. Dabei fällt auf, dass es einen Unterschied zwischen „Schlüsselwerten" in relationalen und „einmalig auftretenden Werten" in XML Daten gibt. In XML Daten können Datenwerte den Typ ID besitzen, der angibt, dass dieser Datenwert an dieser Stelle nur ein Mal im Dokument auftreten darf. In XML Daten können aber mehrere verschiedene Daten, also, um von relationalen Daten zu abstrahieren, im Prinzip mehrere Relationen auftreten. Das hat Vor- und Nachteile: Ein Vorteil ist, dass Fremdschlüssel in XML Daten durch Validierung mit einer Beschreibungsdatei direkt festgelegt und überprüft werden können (ein Datenwert vom Typ IDREF zeigt auf einen Datenwert mit dem Typen ID), sodass auf diese Art und Weise keine Integritätsprobleme zwischen Tupeln entstehen können. Fremdschlüssel über mehrere XML Datenmengen mit unterschiedlichen Beschreibungsdateien können hingegen nicht auf diese Weise überprüft werden.

[1] Eine Möglichkeit zur Vermeidung von Inkonsistenzen dieser Art bietet beispielsweise das Schema Mapping, bei dem bereits vor der eigentlichen Integration der Daten Schemata verschiedener Datenbanken zu einem einheitlichen Schema zusammengeführt werden.

Als Nachteil ist zu erwähnen, dass die Verwendung einfacher numerischer Schlüssel, wie sie in relationalen Daten häufig benutzt werden, in XML Daten, die viele unterschiedliche Informationen enthalten, kaum mehr möglich ist.

2.5 Fehlende Datensätze

Fehlende Datensätze beinhalten bei relationalen Daten das Auslassen von Informationen im Sinne von kompletten Tupeln und können entweder gezielt bei der Erstellung der Daten ausgelassen worden oder auch unerwünscht weggefallen sein.

Tupel relationaler Daten können bei dem Vorgang der Datenintegration wegfallen (durch falsche Bedingungen oder Fehler bei der Datenintegration). Ähnlich können auch Teile von XML Daten ausgelassen werden, wobei hier nicht von Tupeln, sondern eher von Teilstrukturen des Dokumentes oder von Elementen die Rede sein kann, die komplexere Datenstrukturen enthalten und nicht nur einen Datenwert (das wäre dann ein fehlerhafter Attributwert, siehe Abschnitt 2.1.1).

Beim gezielten Auslassen von Daten gibt es keinen Unterschied – so, wie in relationalen Daten Tupel komplett ausgelassen werden können, ist es möglich, in XML Daten (abhängig vom Vorhandensein einer Beschreibung für diese Daten in Form von DTD oder XML Schema) Datenstrukturen gezielt auszulassen.

Kapitel 3

Die DirtyXMLFactory

3.1 Überblick

Während die Integration relationaler Daten schon seit einiger Zeit mit immer ausgereifteren Methoden durchgeführt wird, ist im Vergleich dazu die Integration geschachtelter XML Daten weit weniger ausgereift. Das liegt unter anderem daran, dass sich der XML Standard erst in den letzten Jahren zum einheitlichen Informationsaustausch durchsetzen konnte, während relationale Datenbanksysteme bereits seit den 70er Jahren existieren. Daten in XML Form bieten jedoch gegenüber relationalen Daten viele Vorteile (zum Beispiel standardisierter Informationsaustausch), und eine steigende Anzahl von Diensten bietet den Zugriff auf ihre (unter anderem relationalen) Daten über eine XML Schnittstelle an. Aus diesem Grund ist die Integration geschachtelter XML Daten zu einem hochaktuellen Thema geworden und es werden bereits Algorithmen entwickelt, die Data Cleansing von integrierten XML Daten vornehmen[1]. Um diese Algorithmen eingehend zu testen und Vergleiche ansstellen zu können, werden Benchmarks benötigt. Die DirtyXMLFactory erzeugt aus sauberen XML Daten und einer Menge von Fehlern verschmutzte XML Daten. Bei dieser "Verschmutzung" handelt es sich zunächst um Duplikate, deren Informationsinhalt auf verschiedene Arten verändert und mit Fehlern versehen werden können.

Als Anregung für die Erzeugung der schmutzigen XML Daten diente mir das Tool dbgen [9], mit dem Duplikate von Tupeln relationaler Daten erstellt und verschmutzt werden können.

Im Folgenden werde ich auf wichtige Einzelheiten des Tools, auf seine Parameter und Algorithmen, auf Abläufe im Programm und Erweiterungsmöglichkeiten näher eingehen. Im Anhang B liefern Benchmarks ein paar Informationen über das Ausführungsverhalten des Tools.

[1] z.B. am Lehrstuhl für Informationsintegration des Instituts für Informatik der Humboldt Universität Berlin, http://www.informatik.hu-berlin.de/mac

Anmerkung zu „sauberen" und „schmutzigen" XML Daten

Bereits im Überblick wurden die Begriffe „saubere" und „schmutzige" XML Daten verwendet. Im weiteren Verlauf dieses Kapitels werden diese Begriffe noch häufiger auftauchen. Mit „sauberen" XML Daten sind immer XML Daten gemeint, die in der Regel keine Datenfehler im Sinne von Kapitel 2 besitzen und mit Fehlern versehen werden können. Im Gegensatz dazu sind unter „schmutzigen" XML Daten diejenigen zu verstehen, die gewisse Datenfehler besitzen. Die hier verwendeten Datenfehler beschränken sich dabei auf fehlerhafte Textknoten und Attributwerte, auf Duplikate und teilweise auch auf fehlende Datensätze.

3.2 Umsetzung des Tools

Die DirtyXMLFactory wurde in der Programmiersprache Java implementiert. Abbildung 3.1 zeigt einen Überblick über die Paketstruktur der Anwendung[2]:

Abbildung 3.1: Die Paketstruktur der DirtyXMLFactory

Im Paket de.hu.mac.dirtyxml befindet sich die Hauptklasse des Programms, die DirtyXMLFactory, und eine Exception für bestimmte Fehlersituationen dieses Tools. Im Unterpaket algos befinden sich das Interface ChangeAlgo, das von

[2]Diese Abbildung wurde dem Package Explorer von Eclipse [5] entnommen – der Entwicklungsumgebung, mit der dieses Tool entwickelt wurde.

allen Algorithmen implementiert wird (und von zukünftigen implementiert werden muss), und bereits einige einfache Algorithmen zum Verändern von Zeichen (darauf wird im weiteren Verlauf dieses Kapitels noch genauer Bezug genommen). Für die Generierung der schmutzigen XML Daten werden natürlich Parameter benötigt – alles dafür notwendige befindet sich im Unterpaket params. Darunter ist auch das XML Schema DirtyXMLParameters.xsd zur Validierung von Parametern in XML Form.

3.2.1 Die Benutzerschnittstelle des Programms

Wichtigster Bestandteil des Tools ist die Klasse DirtyXMLFactory, die mit den Methoden aus Tabelle 3.1 eine Schnittstelle zur Erzeugung der verschmutzten XML Daten anbietet.

Methode	Funktion
public void readXMLFile(String)	Liest die saubere XML Datei ein, deren Inhalt als Grundlage für die Erzeugung verschmutzter XML Daten dienen soll.
public void setParameters(String), public void setParameters(DirtyXMLParameters)	Setzt die für die Verschmutzung notwendigen Parameter. Auf die Verwendung der Parameter wird weiter unten noch näher eingegangen.
public void generateDirtyXML()	Generiert, nachdem die saubere XML Datei eingelesen und die Parameter gesetzt wurden, die verschmutzten XML Daten.
public String getDirtyXML()	Gibt die erzeugten schmutzigen XML Daten als eine Zeichenkette zurück.
public void saveDirtyXML(String)	Speichert die erzeugten verschmutzten XML Daten in eine Datei mit dem angegebenen Dateinamen.

Tabelle 3.1: User Interface der DirtyXMLFactory

Nach dem Erzeugen einer Instanz der DirtyXMLFactory muss zunächst mittels der Methode readXMLFile eine XML Datei angegeben werden, die die Originaldaten enthält. Anschliessend werden mit einer der beiden Varianten von setParameters die Parameter gesetzt, die für die Verschmutzung der Daten nötig sind. Der eigentliche Prozess zur Generierung der schmutzigen XML Daten wird durch den Aufruf der Methode generateDirtyXML ausgelöst. Nach Beendigung dieser Aufgabe hält die erzeugte Instanz der DirtyXMLFactory die Daten im Speicher, die der Benutzer nun durch den Aufruf einer der Methoden getDirtyXML oder saveDirtyXML erhalten kann.

3.2.2 Die Parameter zum Erzeugen der schmutzigen XML Daten

Die Parameter sind das Kernstück für die Generierung der schmutzigen XML Daten. Ohne gesetzte Parameter gibt es keinen Unterschied zwischen den erzeugten Daten und den Originaldaten.

Eine Menge von Parametern wird durch eine Instanz der Klasse DirtyXML-Parameters repräsentiert. Darin wird festgelegt, von welchen Elementen Duplikate erstellt werden sollen, mit welcher Wahrscheinlichkeit die Duplikate erstellt werden, bei welchen Attributen Veränderungen des Textinhaltes auftreten sollen und vieles mehr.

Mit der Methode setParameters der Klasse DirtyXMLFactory werden die Parameter dem Programm übergeben. Das kann auf zwei verschiedene Arten geschehen:

1. Mit der Methode setParameters(DirtyXMLParameters parameters) oder
2. durch Aufruf der Methode setParameters(String XMLFile).

Die erste Variante übergibt dem Programm direkt eine Instanz der Klasse DirtyXMLParameters. Der Autor der Parameter muss Java-Kenntnisse besitzen, um Parameter auf diese Art und Weise zu erzeugen.

Mit der zweiten Variante wird dem Programm der Standort einer XML Datei im System angegeben, in der die Parameter enthalten sind. Im Programm wird dann die XML-Datei geparst und daraus eine Instanz der Klasse `DirtyXMLParameters` erzeugt. Hierbei muss der Autor der Parameter keine JAVA-, sondern XML Kenntnisse besitzen, um die Parameter zu setzen. Auf diese Weise erstellte Parameter sind übersichtlicher, besser wartbar und es ist so möglich, das Tool von der Kommandozeile aus zu benutzen, da die saubere XML Datei, die Parameter-XML-Datei und die Ausgabedatei als Kommandozeilenparameter gesetzt werden können.

Auf die Erstellung der Parameter mit der jeweiligen Methode wird im Abschnitt 3.3 näher eingegangen; zunächst wollen wir uns mit dem Aufbau der Parameter beschäftigen.

3.2.2.1 `DirtyXMLParameters`

Wie bereits erwähnt, handelt es sich hierbei um die Hauptklasse für die Parameter. Eine Instanz dieser Klasse enthält eine gewisse Menge definierter Parameter. Abbildung 3.2 zeigt eine Übersicht der Parameter, die in DirtyXMLParameters gesetzt werden können.

Methoden zum Setzen der Parameter:

setValid4Desc(boolean value) Um unterschiedliche Parameter für gleichnamige Elemente in ungleichen Hierarchieebenen zuzulassen, können die Parameter genauso hierarchisch strukturiert werden, wie die Elemente in der sauberen

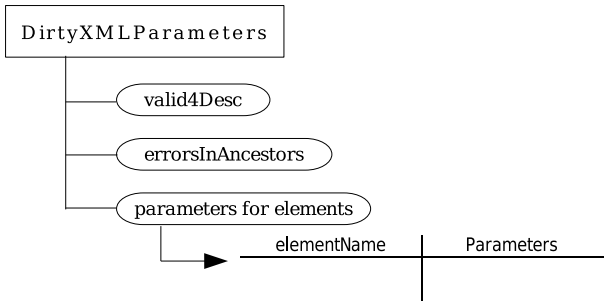

Abbildung 3.2: Die Hauptklasse der Parameter - DirtyXMLParameters

XML Datei vorhanden sind. Zur Veranschaulichung soll das folgende Beispiel dienen. Eine XML Datei habe folgenden Aufbau:

```
<?xml version="1.0"?>
<personen>
  <person ID="57">
    <name>
      <vorname>Angelika</vorname>
      <nachname>Bierhoff</nachname>
    </name>
  </person>
        <!-- weitere "person" Elemente -->
</personen>
```

Dann definiert man direkt in der Instanz der Klasse DirtyXMLParameters Parameter für das Element personen; darin kann man dann Parameter für die person-Elemente definieren, darin dann Parameter für name und wiederum darin Parameter für die Elemente vorname und nachname. Auf diese Weise wird verhindert, dass zum Beispiel die Parameter für vorname auf ein gleichnamiges Element angewendet werden, wenn es ein Kind des Elementes person ist.

Mit der Übergabe des Wertes **true** für den Parameter valid4Desc (unabgekürzt gleich „valid for descendants") ist es aber auch möglich, direkt in der Instanz der Klasse DirtyXMLParameters Parameter für bestimmte Elemente zu setzen, die kein Wurzelelement des XML Dokumentes sind. So können beispielsweise nur Parameter für die Elemente vorname und nachname gesetzt werden, die dann auf alle diese Elemente angewendet werden, egal, in welcher Hierarchieebene sie sich befinden. Wird der Wert **false** übergeben, müssen die direkt in DirtyXMLParameters definierten Elemente im Quelldokument als Wurzelelemente auftreten (wobei in der Regel ein XML-Dokument nur ein Wurzelelement enthalten darf).

setErrorsInAncestors(boolean value) Wird hier der Wert **true** übergeben, werden die Fehlerparameter auch auf die Elemente angewendet, von denen Du-

plikate erstellt werden. Ansonsten bleiben die Originale erhalten und Verschmutzungen kommen nur in den erzeugten Duplikaten zur Geltung.

setElementParameters(String name, ElementParameters params) Hiermit werden die Parameter für ein bestimmtes Element gesetzt, von dem Duplikate erzeugt werden sollen. Die ElementParameters werden im folgenden Abschnitt näher erörtert.

3.2.2.2 ElementParameters

Eine Instanz dieser Klasse enthält bestimmte Parameter für die Duplizierung und Verschmutzung von Elementen, die dann mittels setElementParameters in den DirtyXMLParameters oder in den ElementParameters einem bestimmten XML Element zugewiesen werden. Abbildung 3.3 veranschaulicht die Parameter für Elemente.

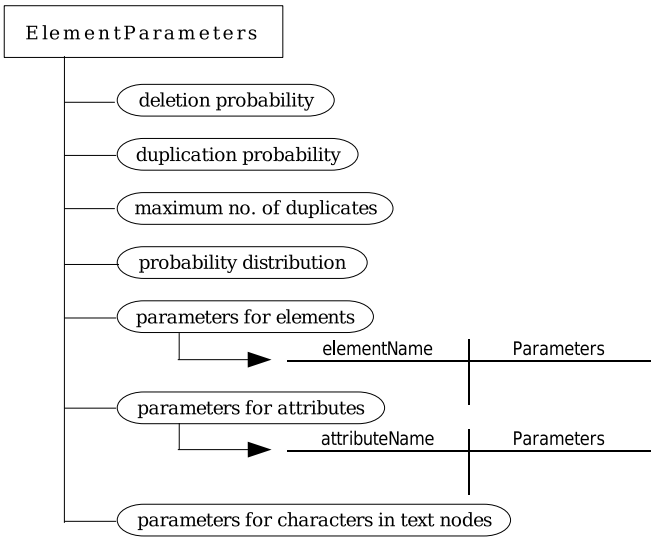

Abbildung 3.3: Parameter für XML Elemente

Methoden zum Setzen der Parameter:

setDeletionProbability(float prob) Setzt die Wahrscheinlichkeit, dass dieses Element gelöscht wird. Für alle Parameter mit Wahrscheinlichkeiten gilt, dass diese als Prozentwert zwischen 0 und 100 angegeben werden müssen.

setDuplicationProbability(float prob) Damit wird die Wahrscheinlichkeit gesetzt, mit der von einem Element mit diesen Parametern Duplikate erstellt werden.

setMaximumDuplicates(int max) Legt die maximale Anzahl zu erzeugender Duplikate fest.

setProbabilityDistributionType(short type) Setzt den Typ der Wahrscheinlichkeitsverteilung, mit der Duplikate erzeugt werden. Momentan werden die Gleich- und die Normalverteilung unterstützt. Als Typen sollten die statischen Konstanten der Klasse ProbabilityDistribution verwendet werden.

setElementParameters(String name, ElementParameters params) Setzt Parameter für ein Kindelement des Elementes, für das diese ElementParameters definiert werden.

setAttributeParameters(String name, CharacterParameters params) Setzt Parameter für ein Attribut des Elementes, für das diese ElementParameters definiert werden. Da es sich beim Inhalt eines Attributes im groben Sinne um eine Zeichenkette handelt[3], werden für ein Attribut CharacterParameters festgelegt, die für die Verschmutzung von Zeichenketten verantwortlich sind. Wenn für ein Attribut keine Parameter gesetzt werden, ist es nach Duplizierung in den Duplikaten genauso vorhanden wie in den Originalen.

setCharacterParameters(CharacterParameter params) Mit dieser Methode werden Parameter für Textknoten, die in Elementen mit diesen Parametern auftauchen können, gesetzt. Werden diese Parameter weggelassen, wird der Text so im Duplikat erscheinen wie im Original.

3.2.2.3 CharacterParameters

Mit einer Instanz dieser Klasse ist es möglich, Parameter für die Verschmutzung von Zeichenketten zu definieren, die sowohl auf Attributwerte als auch auf Textknoten innerhalb von Elementen angewendet werden können. Der Abbildung 3.4 kann der Aufbau der CharacterParameters entnommen werden.

Die Methoden dieser Parameter-Klasse:

setDeleteProbability(float prob) Legt die Wahrscheinlichkeit fest, mit der diese Zeichenkette komplett gelöscht wird. Wenn der Fall eintritt, dass eine Zeichenkette gelöscht werden soll, dann

- wird bei Attributen das Attribut vollständig entfernt,
- werden Elemente, die als Kindelemente nur Textknoten besitzen, anschließend leer sein.

[3] In Verbindung mit einem XML Schema können Werte von Attributen und Inhalte von Elementen auch typisiert sein. Die saubere XML Datei wird jedoch (bisher) nicht in Verbindung mit eventuell existierenden XML Schemas untersucht und verschmutzt.

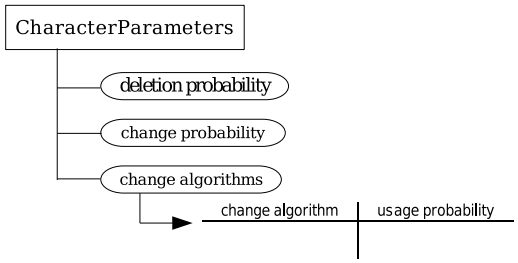

Abbildung 3.4: Parameter zum Verschmutzen von Zeichenketten

setChangeProbability(float prob) Spezifiziert, mit welcher Wahrscheinlichkeit die angegebenen Algorithmen, die die Verschmutzung der Zeichen durchführen, auf die Zeichenkette angewendet werden sollen.

addChangeAlgo(ChangeAlgo algo, float prob) Setzt einen Algorithmus für die Veränderung der Zeichenkette und die Wahrscheinlichkeit, mit der dieser Algorithmus benutzt wird. Es können beliebig viele Algorithmen angegeben werden; es sollte jedoch beachtet werden, dass sich alle Wahrscheinlichkeiten zu 100% aufaddieren. Abschnitt 3.2.3 beschäftigt sich mit Algorithmen und deren Erstellung.

3.2.3 Algorithmen zum Verschmutzen von Zeichenketten

Die Veränderung oder Verunreinigung von Zeichenketten ist ein wesentlicher Bestandteil bei der Erzeugung schmutziger XML Daten. Bei Duplikaten mit gleichem Textinhalt wie die Originale wäre es wesentlich einfacher, diese herauszufiltern, als wenn der Wert von Attributen oder der Textinhalt von Elementen sich in gewisser Weise vom Original unterscheidet. Die Algorithmen zum Data Cleansing müssen in der Lage sein, mit einer bestimmten Wahrscheinlichkeit fehlerhafte Daten herauszufiltern und Duplikate mit sich ergänzenden Daten zusammenzuführen.

Fehlerhafte Daten können auf viele unterschiedliche Arten entstehen. Das Erscheinungsbild der Fehler hängt dabei oft vom Typ der Daten ab und auch davon, wie diese Daten in einen Datenspeicher gelangt sind. Ein Beispiel: Werden etwa in einem Erfassungssystem von Personen die Nachnamen der Personen von Hand eingegeben, so werden diese Namen sicherlich andere Fehler enthalten als vielleicht die Postleitzahl der Adresse, die aus einer Liste mit Postleitzahlen zur Auswahl stand. Bei den Nachnamen können z.B. Buchstabendreher auftauchen oder Buchstaben ausgelassen werden, während eine Postleitzahl einfach falsch aus der Liste ausgewählt worden sein kann.

Aus diesem Grund ist bei der DirtyXMLFactory die Anzahl der Algorithmen unbegrenzt erweiterbar. Bei der Entwicklung eines Algorithmus' ist nur das folgende Interface zu implementieren:

DeleteChar	InsertChar	SwapChars
includeFirstChar	includeFirstChar	includeFirstChar
includeLastChar	includeLastChar	includeLastChar
	includeLower	minSwaps
	includeUpper	maxSwaps
	includeDigits	

Tabelle 3.2: Parameter der Algorithmen DeleteChar, InsertChar und SwapChars

```
public interface ChangeAlgo {
  public String applyAlgorithm(String s);
}
```

Die Methode applyAlgorithm bekommt eine Zeichenkette übergeben, führt damit irgendwelche Umwandlungen durch und gibt diese „verschmutzte" Zeichenkette dann zurück. Ein neu implementierter Algorithmus kann sofort in den Parametern verwendet werden. In der ChangeAlgo implementierenden Klasse wird man dann in der Regel Parameter setzen wollen, die das Verhalten des Algorithmus beeinflussen.

Bisher stehen drei Algorithmen zur Verfügung: DeleteChar, InsertChar und SwapChars. Aus einer Zeichenkette löscht der erstgenannte Zeichen, der zweite fügt welche ein und der dritte vertauscht zwei Zeichen. Die Parameter für die drei Klassen können Tabelle 3.2 entnommen werden.

Für jeden Parameter existiert dann eine Setter-Methode, mit der dieser gesetzt werden kann. Die Namensgebung dieser Methoden ist dabei wie folgt: Das Wort „set" mit dem Namen des Parameters dahinter, wobei der erste Buchstabe des Parameters groß geschrieben wird. Für den Parameter includeFirstChar heisst die Setter-Methode zum Beispiel setIncludeFirstChar. Wenn die Parameter aus einer XML-Datei ausgelesen werden, dann ist darauf zu achten, dass den Methoden immer ein String übergeben werden kann, denn für einen Parameter aus einer XML Datei wird nur der Methodenname im Programm zusammengesetzt und mit dem Wert des Parameters als String aufgerufen.

Um schliesslich eine parametrisierte Instanz eines Algorithmus' zur Verschmutzung von XML Daten benutzen zu können, kann dieser in den CharacterParameters mit der Methode addChangeAlgo angegeben werden.

3.2.4 Wahrscheinlichkeitsverteilungen

Mit den Wahrscheinlichkeitsverteilungen kann fest gelegt werden, welche Anzahl von Duplikaten wie häufig auftreten soll. Bisher stehen die Gleich- und die Normalverteilung zur Verfügung, die sich aber nach Bedarf problemlos erweitern lassen.

Ein Beispiel: Wenn als Wahrscheinlichkeitsverteilung die Normalverteilung verwendet wird und immer 1 bis 10 Duplikate erstellt werden sollen, dann sollten am häufigsten 5 Duplikate, am zweithäufigsten 4 und 6 Duplikate usw. auftreten.

Nach Abbildung 3.3 besitzt die Klasse der ElementParameters den Parameter *probability distribution*. Er wird durch Aufruf der Methode setProbabilityDistributionType gesetzt, dem eine Konstante der Klasse ProbabilityDistribution (momenten ProbabilityDistribution.UNIFORM und ProbabilityDistribution.NORMAL) übergeben wird.

3.3 Verwendung des Programms

Nun werde ich an zwei Beispielen erklären, wie man die DirtyXMLFactory benutzt, um sich endlich seine eigenen schmutzigen XML Daten erstellen zu können. Zwei Beispiele deshalb, weil die Parameter, wie bereits am Anfang dieses Kapitels erwähnt, auf zwei verschiedene Arten erstellt werden können: Direkt in einem JAVA-Programm oder in einer XML Datei.

Doch zunächst der obligatorische Rahmen:

```
DirtyXMLFactory dxf = new DirtyXMLFactory();
String xmlFile = "persons.xml";
String dirtyXMLFile = "persons_dirty.xml";
try {
    dxf.readXMLFile(xmlFile);

    // hier die Parameter definieren und setzen
    // bzw. aus XML Parameter Datei einlesen

    dxf.generateDirtyXML();
    dxf.saveDirtyXML(dirtyXMLFile);
}
catch (Exception e) {
    System.err.println("Fehler bei der
      Ausführung des Programms.");
}
```

Dabei fehlt jetzt noch die Definition der Parameter, ohne die das Programm gar nicht erst anfängt, schmutziges XML zu erzeugen.

Anstatt mit der Methode saveDirtyXML die Daten zu speichern, kann man sie auch mit getDirtyXML als eine Zeichenkette zurückgeliefert bekommen.

Beim Einlesen der Datei und bei der Generierung der XML Daten können verschiedene Fehler auftreten, die hier zusammenfassend mit einer einzigen Exception abgefangen wurden:

- java.io.IOException, org.xml.sax.SAXException und javax.xml.parsers.ParserConfigurationException können beim Einlesen der Originaldaten oder beim Einlesen der Parameter aus einer XML Datei geworfen werden und

- de.hu.mac.DirtyXMLFactoryException folgt, wenn es Fehler bei der Generierung der XML Daten gab oder wenn versucht wurde, Daten zu speichern, die noch gar nicht erstellt wurden.

In den Abschnitten 3.3.1 und 3.3.2 werden die Kommentarzeilen aus dem obigen Quelltext mit der jeweils möglichen Parameterdefinition ersetzt.

3.3.1 Parameter in JAVA

Angenommen, wir haben die XML Datei aus Abbildung 3.5 und wollen daraus eine neue Datei mit Duplikaten und Datenfehlern erzeugen.

```
<?xml version="1.0"?>
<personen>
  <person ID="1" SVN="17954885">
    <vorname>Hans</vorname>
    <name>Schopenhauer</name>
  </person>
  <person ID="2" SVN="19885643">
    <vorname>Albert</vorname>
    <name>Hinkelstein</name>
  </person>
</personen>
```

Abbildung 3.5: Die saubere XML Datei

Das ist eine sehr einfache Quelldatei; in Abschnitt 3.4 wird kurz angesprochen, wie man mit Hilfe des Tools ToXgene komplexere Ausgangsdaten erzeugen kann.

Abbildung 3.5 zeigt eine Liste mit zwei Personen. Jede Person besitzt eine ID und eine Sozialversicherungsnummer (hier „SVN") und darüber hinaus einen Vor- und Nachnamen.

Wir wollen nun festlegen, dass das Element person nie gelöscht, aber immer, also mit einer Wahrscheinlichkeit von 100%, dupliziert wird, und das maximal zwei Duplikate erstellt werden:

```
ElementParameters personParams
  = new ElementParameters(0, 100, 2);
```

Im Konstruktor können gleich *deletion probability*, *duplication probability* und *maximum duplicates* gesetzt werden. Alternativ ist das natürlich auch mit den Methoden aus Abschnitt 3.2.2 möglich. Nun soll das Attribut ID bei den Duplikaten immer gelöscht werden. Ausserdem soll das Attribut SVN nie gelöscht und zu 75% verändert werden.

```
CharacterParameters idParams
```

```
  = new CharacterParameters(100, 0);
CharacterParameters svnParams
  = new CharacterParameters(0, 75);
```

Hier wurden ebenfalls *deletion probability* und *change probability* gleich im Konstruktor angegeben.

Damit das Attribut SVN verschmutzt werden kann, benötigen wir Algorithmen, die die Verschmutzung durchführen. Wir benutzen SwapChars zum Vertauschen zweier Zeichen; das erste und das letzte Zeichen der Zeichenkette sollen für den Algorithmus mit verwendet und es sollen ein bis zwei Vertauschungen durchgeführt werden.

```
SwapChars swap1 = new SwapChars();
swap1.setIncludeFirstChar("true");
swap1.setIncludeLastChar("true");
swap1.setMinSwaps("1");
swap1.setMaxSwaps("2");
```

Schliesslich muss noch festgelegt werden, dass swap1 auf SVN angewendet werden soll (und wie hoch die Wahrscheinlichkeit ist, dass dieser Algorithmus gewählt wird):

```
svnParams.addChangeAlgo(swap1, 100);
```

Bleibt noch zu sagen, dass ID und SVN Attribute des Elementes person sind:

```
personParams.setAttributeParameters("ID", idParams);
personParams.setAttributeParameters("SVN", svnParams);
```

Ähnlich werden jetzt Parameter für das Element vorname festgelegt, nur mit anderen Wahrscheinlichkeiten und anderen Algorithmen. Aus Platzgründen legen wir keine Parameter für den Nachnamen fest. Damit bleibt in den Duplikaten dieses Element so erhalten, wie es im Original vorkommt.

```
DeleteChar del1 = new DeleteChar();
del1.setIncludeFirstChar("false");
del1.setIncludeLastChar("true");

InsertChar ins1 = new InsertChar();
ins1.setIncludeFirstChar("false");
ins1.setIncludeLastChar("true");
ins1.setIncludeDigits("false");
ins1.setIncludeLower("true");
ins1.setIncludeUpper("false");

ElementParameters vornameParams
  = new ElementParameters(50, 60, 4);
```

```
CharacterParameters vornameText
  = new CharacterParameters(30, 80);
vornameText.addChangeAlgo(del1, 50);
vornameText.addChangeAlgo(ins1, 50);
vornameParams.setCharacterParameters(vornameText);
```

Entsprechend der Hierarchie in der Original-XML Datei müssen zum Schluß noch die ElementParameters ineinander verschachtelt und diese Parameter dann gesetzt werden:

```
personParams.setElementParameters("vorname", vornameParams);

DirtyXMLParameters params =
  new DirtyXMLParameters(true, false);
params.setValid4Desc(true);
params.setErrorsInAncestors(false);
params.setElementParameters("person", personParams);
dxf.setParameters(params);
```

Da keine Parameter für das Wurzelelement personen festgelegt wurden, wird der Parameter valid4Desc auf **true** gesetzt; ausserdem sollen die Original-Elemente nicht verschmutzt werden.

3.3.2 Parameter in XML

Auf ähnliche Weise, wie die Parameter in Java erzeugt werden können, ist es möglich, sie in eine XML Datei zu legen. Die XML Datei in Anhang A enthält die gleichen Parameter, die wir eben in Java definiert haben, in einer XML Datei. Dabei sollte zur Validierung das XML Schema DirtyXMLParameters.xsd verwendet werden, dass in das Wurzelelement der Parameter-XML-Datei wie folgt eingefügt wird:

```
<dirtyXMLparameters
    xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
    xs:noNamespaceSchemaLocation="DirtyXMLParameters.xsd"
    valid4Desc="true"
    errorsInAncestors="false">

  <!-- ... Parameter-Definitionen -->

</dirtyXMLparameters>
```

Beim Standort der Schema-Datei im Dateisystem muss gegebenenfalls noch der Pfad hinzugefügt werden, in dem sich dieses Schema befindet. Die Werte der beiden Parameter valid4Desc und errorsInAncestors können natürlich vom Beispiel abweichende Werte haben.

Die Klasse ParameterFileReader im Paket de.hu.mac.dirtyxml.params (siehe Abbildung 3.1) ist dafür zuständig, Parameter aus einer XML Datei in eine Instanz der DirtyXMLParameters umzuwandeln.

Der Inhalt der Parameterdatei ist aus Gründen der Übersichtlichkeit in zwei Teile aufgegliedert: Zum einen die Definition der Algorithmen zur Veränderung von Zeichenketten und zum anderen die Parameter für die Elemente, die dupliziert werden sollen. Da der Aufbau der Parameter in XML ansonsten sehr ähnlich dem in Java ist, gehe ich an dieser Stelle nicht weiter darauf ein und verweise auf das Beispiel in Anhang A und auf das zugrunde liegende Schema DirtyXMLParameters.xsd. Informationen über XML Schema gibt es unter [3] und in vielen Fachbüchern zum Thema.

3.3.3 Benutzung des Programms aus der Kommandozeile

Liegen die Parameter als XML Datei vor, so kann das Tool auch von der Kommandozeile aus gestartet werden. Der Parameter für den Programmaufruf sind dabei wie folgt:

1. Name der sauberen XML Datei,
2. Name der Parameter XML Datei,
3. Name der schmutzigen XML Datei.

3.4 Abläufe im Programm

In diesem Abschnitt wird auf einige Abläufe im Programm näher eingegangen. Dabei konzentriere ich mich hier auf die wichtigsten Prozesse: Die Generierung der schmutzigen XML Daten und das Einlesen von Parametern aus einer XML Datei.

3.4.1 Erzeugung der schmutzigen XML Daten

Wenn die Quelldatei eingelesen und die Parameter gesetzt wurden, wird der Prozess zur Generierung der schmutzigen XML Daten mit der Methode generateDirtyXML gestartet. Danach wird überprüft, ob das Wurzelelement des XML Dokumentes in den Parametern vorkommt. Ist dies der Fall, dann werden die Parameter auf dieses Element angewendet. Dieser Vorgang wird rekursiv auf die Kinder dieses Elementes bis zum letzten Element in der Hierarchie durchgeführt.

Werden Parameter auf ein Element angewendet, so bedeutet das, das der Algorithmus zunächst mittels Zufallszahlen überprüft, ob dieses Element gelöscht wird; dann, ob es dupliziert wird, und falls ja, wie viele Duplikate entstehen sollen. Wenn Duplikate erzeugt wurden, dann werden an jedem von ihnen folgende Schritte ausgeführt:

1. Fehlerparameter auf die Attribute anwenden,

2. Fehlerparameter auf die Kindelemente des Duplikats anwenden (also auch Fehlerparameter auf Textknoten in diesem Element),

3. Duplikat mit verschmutzten Daten in eine Liste von Duplikaten aufnehmen.

Sollen die Fehlerparameter auch auf die Originale angewendet werden, so durchlaufen die Originale anschliessend ebenfalls die ersten beiden Schritte.

Attribute und Textknoten, also der Textinhalt von Elementen, werden wie folgt mit Fehlern versehen: Zuerst wird überprüft, ob das Attribut oder der Textknoten gelöscht wird; danach, ob sein Datenwert geändert wird, und falls ja, werden die gesetzten Fehlerparameter auf die Zeichenkette des Attributes oder den Textinhalt des Elementes angewendet.

Um Zeichenketten zu verändern, geht das Programm wie folgt vor: Für Zeichenketten können in den Parametern Algorithmen angegeben werden, die die Zeichen verändern. Dabei wird zu jedem Algorithmus eine Wahrscheinlichkeit angegeben, mit welcher er verwendet wird, um eine Veränderung der Zeichenkette durchzuführen. Die Summe der Wahrscheinlichkeiten aller Algorithmen für die Veränderung einer Zeichenkette muss dabei 100% betragen. Im Programm werden nun im Prinzip alle Algorithmen hintereinander aufgelistet und ein Algorithmus wird durch eine Zufallszahl zwischen 0 und 100 gewählt. Ein Beispiel hierzu sei in Abbildung 3.6 veranschaulicht.

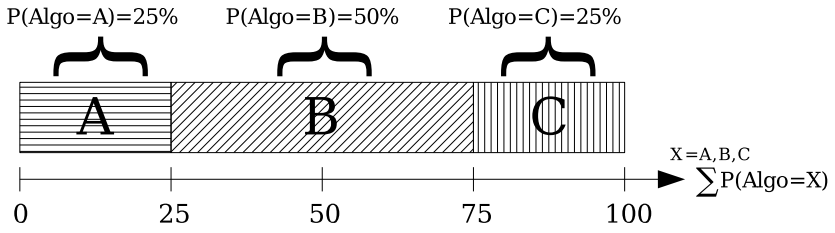

Abbildung 3.6: Beispiel zur Verwendung der Algorithmen

In diesem Beispiel wurden zur Verschmutzung der Zeichenketten eines Attributs oder eines Elements drei Algorithmen A, B und C mit den Wahrscheinlichkeiten 25%, 50% und 25% angegeben. Das Programm listet nun diese Algorithmen wie in Abbildung 3.6 auf und wählt eine Zufallszahl zwischen 0 und 100. Die vom Zufallsgenerator erzeugte Zahl liegt dann irgendwo auf der unteren Geraden, die die Summe aller Wahrscheinlichkeiten darstellt. Der Algorithmus, der dann darüber liegt, wo die Zufallszahl eingeordnet werden kann, wird für die Verschmutzung der Zeichenkette verwendet, über deren Verschmutzung gerade entschieden wird.

Wenn am Ende alle Original-Elemente geparst und einige Duplikate erstellt wurden, liegt im Programm eine Liste von Duplikaten vor, die dann noch mit dem Originaldokument zusammengeführt werden müssen, um das schmutzige XML Dokument zu erhalten. Die Originalelemente (das sind die, von denen die

Duplikate erstellt wurden) sind im resultierenden schmutzigen XML Dokument immer mit enthalten, können aber selbst auch durch Setzen des entsprechenden Parameters verschmutzt werden. Für die Duplikate existiert in der DirtyXMLFactory eine innere Klasse mit dem Namen DupNode, die sich den verschmutzten duplizierten Knoten und seinen Elternknoten merkt. Die Liste der Duplikate wird der Reihe nach durchgegangen und die Duplikate werden nacheinander in die Struktur des Originaldokumentes eingefügt.

3.4.2 Einlesen der Parameter aus einer XML Datei

Für die Umwandlung dieser Parameter in eine Instanz der DirtyXMLParameters ist die Klasse ParameterFileReader im Package params zuständig.

Zunächst wird die XML Datei als DOM eingelesen. Dann werden alle gefundenen Algorithmen abgearbeitet. Für jedes algo Element wird eine Instanz des Algorithmus erzeugt, mit den gefundenen Parametern (Elemente parameter) versehen und die jeweilige Instanz wird dann zusammen mit dem angegebenen Referenznamen des Algorithmus' (Attribut name) in einer Hashtable abgelegt. Damit kann dann jede Instanz später bei der Erstellung von CharacterParameter - Instanzen referenziert werden.

Bei der Erstellung einer Instanz eines Algorithmus wird die Reflection API von Java benutzt und im Detail wird wie folgt vorgegangen:

1. Erstellung des Namens der Algorithmus-Klasse aus dem Wert des Attributs baseAlgo; der erste Buchstabe des Wertes wird dafür in einen Großbuchstaben umgewandelt.

2. Erzeugung des Klassenobjektes und einer Instanz der Algorithmus-Klasse aus diesem Namen.

3. Für jeden Parameter:

   (a) Erzeugen des Methodennamens, mit der dieser Parameter im aktuellen Algorithmus gesetzt wird. Dabei wird das Wort „set" mit dem Namen des Parameters (Attribut name) verknüpft, von dessen Namen das erste Zeichen in einen Großbuchstaben umgewandelt wird. Der erzeugte Methodenname ist die jeweilige Setter-Methode für diesen Parameter.

   (b) Erhalt einer Instanz dieser Methode von der Algorithmus-Klasse und Aufruf der Methode mit dem Wert des Parameters (Attribut value) als String (deshalb sollten die Setter-Methoden der Algorithmus-Klassen immer in der Lage sein, String-Werte als Parameter zu akzeptieren).

4. Einfügen des Algorithmus' in die Hashtable.

Anschliessend werden die Parameter für die Elemente eingelesen, für die Duplikate erzeugt werden sollen. Das ist ein einfacher Vorgang: Eine Instanz der

ElementParameters wird erzeugt, die Parameter für die Algorithmen und für den Textinhalt werden gesetzt, und anschließend werden rekursiv die Parameter für die Kindelemente eingelesen. Wird eine Instanz der CharacterParameters erzeugt und sind für die Veränderung der Zeichen Algorithmen angegeben worden, dann werden die Instanzen der Algorithmen aus der oben genannten Hashtable geholt.

3.5 Saubere XML Daten mit ToXgene

Mit der DirtyXMLFactory gibt es nun ein Tool, mit dem man schmutzige XML Daten generieren kann. Dazu werden saubere Daten benötigt, die als Input zur Generierung der Daten fungieren. Mit Absicht wurde darauf verzichtet, schmutzige XML Daten nur aus einer Menge von Parametern „von Grund auf" zu generieren – denn es gibt bereits Tools, die in der Lage sind, massenhaft „normale" XML Daten zu erschaffen.

Eines dieser Tools ist ToXgene, das an der University of Toronto entwickelt wurde. Die zugrunde liegende Programmiersprache ist Java. Für die Generierung der XML Daten muss ein um spezielle Anmerkungen erweitertes XML Schema erstellt werden, das sowohl die Struktur als auch den Inhalt der Daten, die generiert werden sollen, beschreibt.

ToXgene und eine ausführliche Anleitung zur Verwendung dieses Tools stehen auf der Website des Tools [4] zum Download bereit.

3.6 Verbesserungs- und Erweiterungsmöglichkeiten

In der Regel durchlaufen sowohl komplexe Anwendungen als auch relativ einfache Tools den Software Life Cycle – und so ist natürlich auch die DirtyXMLFactory keineswegs perfekt und mir sind bereits einige Dinge aufgefallen, die eine Veränderung benötigen würden. Der für eine Studienarbeit relativ kurze Zeitraum von zwei Monaten ist darüber hinaus auch kaum ausreichend, um sowohl ein Tool zu entwickeln als auch es komplexen Tests zu unterziehen. Jedoch wurde die Entwicklung mit größtmöglicher Sorgfalt vorgenommen und das Tool kann durchaus als „einsatzbereit" bezeichnet werden.

Das saubere XML Dokument wird momentan als Document Object Model (DOM) [6] eingelesen und das schmutzige XML Dokument wird ebenfalls als DOM erzeugt. Der Speicherverbrauch von DOM Bäumen ist sehr groß, und die XML Dokumente werden vollständig im Speicher gehalten. Es kann dabei Probleme mit sehr großen Dokumenten geben oder wenn sehr viele Duplikate erzeugt werden sollen. Deshalb ist eine Umstellung auf die reine Verwendung von SAX (Simple API for XML) [7] oder aber auf eine Mischung zwischen SAX und DOM, nämlich SAXDOMIX [8] anzuraten.

Die Verwendung von reinem SAX wäre sicherlich komplizierter, dafür könnte man selbst für platzsparende Speicherstrukturen sorgen und außerdem nur die

„beachtenswerten" Elemente – d.h. diejenigen, für die Parameter existieren – in den Speicher aufnehmen und bearbeiten. SAXDOMIX hingegen liefert für vorher definierte Elemente „kleine" DOM Bäume, sozusagen Teilbäume vom gesamten DOM des Dokuments. So könnten die sauberen XML Daten Stück für Stück abgearbeitet und die Duplikate gleich in eine Datei geschrieben werden, ohne sie im Speicher zu halten.

Bei der Speicherung der XML Daten wird im Moment noch der Umweg über JDOM gemacht, da mit DOM selbst keine Möglichkeit zur Serialisierung des Baumes im XML Format vorliegt. Diese Umwandlung kostet natürlich wiederum viel Speicherkapazität und Rechenleistung. Sinnvoller ist es, die Serialisierung des DOM selbst vorzunehmen. Dann könnte auch gleich eine „human-readable" Formatierung der resultierenden XML Daten erfolgen.

Ich persönlich finde es auch etwas umständlich, sowohl eine Parameter-Datei für ToXgene als auch eine für die DirtyXMLFactory zu schreiben. Auch wenn diese Variante zweier Dateien vielleicht sauberer ist, wäre eine Möglichkeit zur Vereinigung beider Parameter-Dateien denkbar. Man könnte ein ToXgene Schema noch mit weiteren Parametern für die Erzeugung schmutziger XML Daten versehen, im Programm dann die Parameter trennen und einen Teil an ToXgene zur Erzeugung der sauberen und danach den anderen an die DirtyXMLFactory zur Erzeugung der schmutzigen XML Daten übergeben.

Weiterhin ist es natürlich immer möglich, mehr Wahrscheinlichkeitsverteilungen und Algorithmen zu implementieren. Über Algorithmen wurde bereits in Abschnitt 3.2.3 gesprochen. Wahrscheinlichkeitsverteilungen können in der Klasse ProbabilityDistribution ergänzt werden, indem man sich einfach an den bereits vorhandenen Verteilungen (bisher Gleich- und Normalverteilung) orientiert.

Letztendlich könnte die Verschmutzung von Zeichenketten auch typisiert gestaltet werden. Das bedeutet, dass den XML Daten ein XML Schema zugrunde liegt, in dem für den Wert von jedem Attribut und dem Textinhalt jedes Elements ein Datentyp angegeben werden muss. Damit wäre es dann zum Beispiel möglich, bestimmte Berechnungen mit Zahlenwerten vorzunehmen.

Anhang A

Parameter in XML

Die folgende Datei ist die ungekürzte Variante der Parameter aus Abschnitt 3.3.1 in Form einer XML Datei.

```
<?xml version="1.0"?>
<dirtyXMLparameters
xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="DirtyXMLParameters.xsd"
valid4Desc="true"
errorsInAncestors="false">
  <algo name="swap1" baseAlgo="swapChars">
    <parameter name="includeFirstChar" value="true"/>
    <parameter name="includeLastChar" value="true"/>
    <parameter name="minSwaps" value="1"/>
    <parameter name="maxSwaps" value="2"/>
  </algo>
  <algo name="del1" baseAlgo="deleteChar">
    <parameter name="includeFirstChar" value="false"/>
    <parameter name="includeLastChar" value="true"/>
  </algo>
  <algo name="ins1" baseAlgo="insertChar">
    <parameter name="includeFirstChar" value="false"/>
    <parameter name="includeLastChar" value="true"/>
    <parameter name="includeUpper" value="false"/>
    <parameter name="includeLower" value="true"/>
    <parameter name="includeDigits" value="false"/>
  </algo>
  <dupElement name="person" delProb="0" dupProb="100"
              maxDup="2">
    <attribute name="ID">
      <chars delProb="100" changeProb="0"/>
    </attribute>
```

```
      <attribute name="SVN">
        <chars delProb="0" changeProb="75">
          <changeAlgo algoName="swap1" useProb="100"/>
        </chars>
      </attribute>
      <dupElement name="vorname" delProb="50" dupProb="60"
                  maxDup="4">
        <chars delProb="30" changeProb="80">
          <changeAlgo algoName="del1" useProb="50"/>
          <changeAlgo algoName="ins1" useProb="50"/>
        </chars>
      </dupElement>
    </dupElement>
  </dirtyXMLparameters>
```

Anhang B

Benchmark des Tools

An dieser Stelle soll ein kurzer Einblick in das Ausführungsverhalten des Tools gegeben werden. Weitere, ausführlichere Benchmarks werden im weiteren Entwicklungsverlauf folgen. Die sauberen XML Daten werden mit ToXgene und dem bei ToXgene mitgelieferten Template zur Generierung eines Buchkataloges (catalog.tsl im Verzeichnis samples/catalog) erzeugt.

Die mit diesem Template erzeugten XML Dateien haben den Aufbau wie in Abbildung B.1.

```
<catalog>
  <book isbn="0732633599" genres="Adventure,Children">
    <title>thinly close escapades</title>
    <author>Lucio Sigal</author>
    <author>Raquel Reeken</author>
    <author>Shantanu Strouf</author>
    <author>Nevana Koshors</author>
      <!-- evtl. weitere Autoren -->
    <price currency="CDN">270.00</price>
  </book>
          <!-- weitere book elemente -->
</catalog>
```

Abbildung B.1: Mit ToXgene erzeugte XML Datei

Die Parameter, die zur Erzeugung der schmutzigen XML Daten verwendet wurden, sind der Abbildung B.2 zu entnehmen. Die Parameter der Algorithmen werden an dieser Stelle nicht extra aufgeführt.

Die Katalogdaten wurden mit 500, 1000, 2000 und 3000 Buchelementen erzeugt. Bereits bei der Ausführung der DirtyXMLFactory mit 2000 Buchelementen brach das Programm beim Speichern der Daten mit einem OutOfMemoryError ab – ein Beleg dafür, dass unbedingt eine Änderung der Speicherungsfunktionalität programmiert werden muss. Die Zeit, die für die Generierung benötigt

```
<dupElement name="book" delProb="0"
            dupProb="70" maxDup="20">
  <attribute name="isbn">
    <chars delProb="0" changeProb="75.4">
      <changeAlgo algoName="swap1" useProb="60"/>
      <changeAlgo algoName="ins2" useProb="40"/>
    </chars>
  </attribute>
  <dupElement name="title" delProb="0"
              dupProb="50" maxDup="2">
    <chars delProb="35" changeProb="28">
      <changeAlgo algoName="del1" useProb="100"/>
    </chars>
    <dupElement name="author" delProb="50"
                dupProb="60" maxDup="1">
      <chars delProb="40" changeProb="25">
        <changeAlgo algoName="ins1" useProb="100"/>
      </chars>
    </dupElement>
    <dupElement name="price" delProb="30"
                dupProb="0" maxDup="0">
      <attribute name="currency">
        <chars delProb="20" changeProb="0"/>
      </attribute>
    </dupElement>
  </dupElement>
</dupElement>
```

Abbildung B.2: Die Parameter zum Verschmutzen der Katalogdaten

wurde, konnte aber bestimmt werden. Die folgenden Tabellen stellen die Ergebnisse der Benchmarks dar. Die Programm wurde auf einem Pentium M 1.3 GHz mit 512 MB RAM durchgeführt.

Generierungszeit für 500 Elemente: 627 ms

Elementname	# Elemente	# Elemente nach Verschmutzung
catalog	1	1
book	500	4341
title	500	7217
author	2678	22610
price	500	4341

Tabelle B.1: Ergebnisse für die Verschmutzung der Datei mit 500 book-Elementen

Generierungszeit für 1000 Elemente: 1044 ms

Elementname	# Elemente	# Elemente nach Verschmutzung
catalog	1	1
book	1000	8173
title	1000	13523
author	5652	46442
price	1000	8173

Tabelle B.2: Ergebnisse für die Verschmutzung der Datei mit 1000 book-Elementen

- *Generierungszeit für 2000 Elemente: 2013 ms*
- *Generierungszeit für 3000 Elemente: 2974 ms*

Literaturverzeichnis

[1] Daniel Tonn: *Data Cleansing Verfahren für Data Warehouses*, Diplomarbeit an der Humboldt Universität Berlin, Institut für Informatik, 31. Oktober 2000

[2] Silberschatz, Korth, Sudarshan: *Database System Concepts*, Mc Graw Hill, 4th Edition (2002)

[3] XML Schema Specifications and Development. http://www.w3.org/XML/Schema#dev

[4] ToXgene: An extensible template-based data generator for XML. http://www.cs.toronto.edu/tox/toxgene/

[5] The Eclipse Project. http://www.eclipse.org

[6] The W3C Document Object Model (DOM). http://www.w3.org/DOM/

[7] Simple API for XML (SAX). http://www.saxproject.org/

[8] SAX + DOM Mix = SAXDOMIX - Free Open Source Standards-based Framework for Scalable XML Processing. http://www.devsphere.com/xml/saxdomix/

[9] The CUCS Mailing-List Generator written by Mauricio A. Hernandez. http://www.cs.utexas.edu/users/ml/riddle/data/dbgen.tar.gz