# Masterarbeit

zur Erlangung des akademischen Grades Master of Arts (M.A.)

an der Fakultät für Linguistik und Literaturwissenschaft

- Universität Bielefeld -

# Repräsentation, Verarbeitung und Visualisierung multipler Hierarchien mit XStandoff und XSLT

von

Daniel Jettka

| | |
|---|---|
| Gutachter: | Maik Stührenberg, Universität Bielefeld |
| | Prof. Dr. Dieter Metzing, Universität Bielefeld |
| Abgabedatum: | 08.03.2011 |

# Inhaltsverzeichnis

# 1 Einleitung

Die Annotation sprachlicher Daten, welche derzeit zumeist mit Hilfe der eXtensible Markup Language (XML) realisiert wird, stößt mit zunehmender Komplexität bzw. der Einbeziehung einer steigenden Anzahl von Hierarchien, bspw. in Form linguistischer Beschreibungsebenen, unweigerlich an ihre Grenzen. Durch das zugrunde liegende strikt hierarchische Baummodell ist XML in seiner Standard-Verwendung nicht in der Lage, bestimmte strukturelle Phänomene, wie z.B. Überlappungen von Annotationselementen oder diskontinuierliche Einheiten, zu erfassen. In diesem Zusammenhang ist eine Reihe von Modellen und Formaten entstanden, die versuchen die Beschränkungen von XML in unterschiedlicher Weise zu behandeln. Eines dieser Formate ist das in Folge des Forschungsprojekts „Sekundäre Informationsstrukturierung und vergleichende Diskursanalyse“ der DFG-Forschergruppe 437 „Texttechnologische Informationsmodellierung“ entstandene XStandoff, welches eine Kombination aus Standoff-Ansatz und multiplen Dokumenten (bzw. multi-rooted trees) darstellt, und als Meta-Auszeichnungssprache für multiple Hierarchien angesehen werden kann. Mit Hilfe von XSLT 2.0-Stylesheets wird die Repräsentation und Verarbeitung von XStandoff-Instanzen erleichtert und eine exemplarische Visualisierung in Form von interaktiven SVG-Dateien ermöglicht.

Die vorliegende Masterarbeit ist durch einen einleitenden theoretischen und einen daran anschließenden praktischen Teil gekennzeichnet. Im theoretisch geprägten zweiten Kapitel wird zunächst das o.g. grundlegende Problem von XML bei der Repräsentation multipler Hierarchien ausführlich dargelegt. Darüber hinaus werden vorhandene Repräsentationsformate skizziert und es erfolgt eine Einordnung der Ansätze mit Blick auf die jeweiligen Vor- und Nachteile.

Darauf aufbauend führt das dritte Kapitel in die Entwicklung und den derzeitigen Stand des Repräsentationsformats XStandoff ein. Hier wird u.a. deutlich, welche Strategien in XStandoff angewendet werden, um bspw. Überlappungskonflikte und diskontinuierliche Einheiten in angemessener Weise zu repräsentieren.

Die Werkzeuge, die für die weitergehende, z.T. recht komplexe Verarbeitung von XStandoff-Instanzen zur Verfügung stehen, werden im vierten Kapitel vorgestellt. In den einzelnen Abschnitten erfolgt für jedes vorhandene XSLT-Stylesheet eine Erläuterung der grundlegenden Funktion. Die genaue technische Umsetzung kann aufgrund der Komplexität leider nur in Ansätzen erfolgen. Umfassende Quellcode-Dokumentationen, sowie die Stylesheets selbst und Beispiele für Ergebnisse ihrer Anwendung sind jedoch auf der beiliegenden CD einzusehen.

Zum Abschluss der Arbeit befasst sich das fünfte Kapitel mit einer kurzen Zusammenfassung und gibt einen Ausblick auf mögliche weitere Entwicklungen.

# 2 Multiple Hierarchien

## 2.1 Annotation sprachlicher Daten

Die Untersuchung komplexer, sprachlicher Zusammenhänge basiert i.d.R. auf einer Vielzahl von Informationen aus verschiedenen konzeptuellen Domänen, wie z.B. den klassischen linguistischen Beschreibungsebenen der Phonologie, Morphologie, Syntax, Semantik, Pragmatik oder auch textuellen Analyseebenen wie der logischen Dokumentstruktur, der rhetorischen Struktur, der Diskursstruktur, etc. Durch die zunehmende Verbreitung texttechnologischer und korpuslinguistischer Methoden, im Speziellen der Annotation von Sprachdaten, sowie einer steigenden Anzahl betrachteter Analyseebenen, besteht seit längerer Zeit der Bedarf, die Informationsvielfalt in geeigneter Weise zu repräsentieren (vgl. Witt et al., 2005: 103; Goecke et al., 2010: 1).

Den vorherrschenden Annotationsformaten für linguistische Daten liegen zumeist XML[1]-basierte Auszeichnungssprachen zugrunde (vgl. Schmidt, 2010: 339). Da XML (und der Vorgänger SGML) „evolved from the 'presentational' markup contained in early digital documents intended for printing“ (Schmidt, 2010: 339 in Anlehnung an Goldfarb, 1996, 1997), orientiert sich die Grundkonzeption von XML an der Prämisse einer hierarchischen Struktur von Texten. Die Hierarchie-These wurde von DeRose et al. (1990) explizit formuliert, indem sie Texten eine „Ordered Hierarchy of Content Objects“ (OHCO; DeRose et al., 1990) als Strukturmodell zugrunde legten. Allerdings wurde die These aufgrund vielfacher Gegenbeispiele mehrfach rela-

1 Extensible Markup Language (vgl. http://www.w3.org/XML/)

tiviert und kann mittlerweile nur noch in eingeschränkter Form als zutreffend bezeichnet werden: „text is composed of 'meaning related features' that are 'often hierarchical'" (Schmidt, 2010: 342). Als möglichen Grund für die ursprüngliche Annahme, dass Texte eine OHCO konstituieren, stellt Witt (2001: 41) fest, dass „vermutet werden [kann], dass sich die Hierarchie-These nur entwickeln konnte, da ihr ein sehr enger Textbegriff zugrunde liegt".

Unter dem Eindruck der OHCO-These für auszuzeichnende Texte und aus Gründen der besseren computationellen Verarbeitbarkeit (vgl. Marinelli et al., 2008: 1; Schmidt, 2010: 344), ist es nicht verwunderlich, dass XML auf einem strikt hierarchischen Baummodell basiert:

> „XML-based markup requires that the identified features of a document are organized hierarchically as a single tree, whereby each fragment of the content of the document is contained in one and only one XML element, each of which is contained in one and only one parent element all the way up to the single root element at the top" (Marinelli et al., 2008: 1)

Zwar können mit Hilfe von Attributen und der Nutzung des XML-inhärenten ID/IDREF-Prinzips auch Modelle realisiert werden, welche über strikte Hierarchien hinausgehen (vgl. Abschnitt 2.3.1, S. 12 oder auch Abschnitt 3.2, S. 21); die meist genutzte und ursprünglich intendierte Auszeichnung von Inhalten erfolgt jedoch durch Elemente. Vor diesem Hintergrund kann die „limitation of SGML-based markup languages to tree structured annotations" (Witt, 2007: 1) allerdings zu bedeutenden Einschränkungen mit Blick auf die XML-basierte Repräsentation mehrerer Analyseebenen (multipler Hierarchien) für ein Sprachdatum führen. Die spezifischen Probleme für hierarchische Annotationsformate werden im folgenden Abschnitt erläutert.

## 2.2 Probleme der Repräsentation

Das wohl am häufigsten und umfassendsten untersuchte Problem im Kontext der Repräsentation multipler Hierarchien sind überlappende Auszeichnungen in Form von Annotationen aus verschiedenen Hierarchien. Das Grundproblem skizziert Schmidt (2010: 344) folgendermaßen:

> „The origin of the overlap problem is simply that humanists are trying to represent what they all agree are non-hierarchical structures using a container whose primary structure is a tree."

So sind Überlappungen wie die folgende z.B. nicht durch die Standard-XML-Notation zu erfassen (z.B. Marinelli et al., 2008: 1):

```
1   <doc><b>John <i>likes</b> Mary</i></doc>
```

In dieser Pseudo-XML-Repräsentation (nicht wohlgeformtes XML) sind die Elemente <b> und <i> nicht hierarchisch ineinander eingebettet, da sich die von ihnen annotierten Textinhalte an der Stelle des Strings „likes" überschneiden. Die Kontexte linguistischer Analysen, in denen solche Arten von Überlappungen auftreten können sind vielfach thematisiert worden:

- Überlappungen von Erzählstruktur und Dokumentstruktur: die Rede eines Charakters beginnt in der Mitte eines Paragraphen und erstreckt sich über mehrere folgende (vgl. Witt, 2007: 1; Marinelli et al., 2008: 2; Burnard & Bauman, 2010: 577; DeRose, 2004: 1)
- Überlappungen von lyrischer und syntaktischer Struktur: die formale Struktur von Versen (Strophen, Zeilen) kann sich mit Satz- oder Phrasengrenzen überschneiden (vgl. Witt, 2007: 1; Marinelli et al., 2008: 2; Burnard & Bauman, 2010: 577)

- Überlappungen der physischen und der formalen oder logischen Struktur von Texten: Auszeichnungen für Seiten, Spalten und Zeilen können sich mit denen von Kapiteln, Paragraphen, Akten und Szenen überschneiden (vgl. Witt, 2007: 1; Burnard & Bauman, 2010: 577)

Eine weitere Art von Überlappung ist diejenige von Primärdaten, also den reinen sprachlichen Daten ohne hinzugefügte Daten wie Annotationen. Diese kann sich z.B. bei der Auszeichnung verschiedener Versionen von Texten (vgl. Marinelli et al., 2008: 2) oder sich überlappenden Sprecherbeiträgen in gesprochener Sprache (vgl. Wörner, 2009: 44) ergeben. Überlappende Primärdaten stellen ein schwerwiegendes Problem dar, welches allerdings nicht im Fokus der vorliegenden Arbeit liegt. Vielmehr geht es im Folgenden um die spezifischen Probleme der Repräsentation multipler Hierarchien von Annotationen.

Eine technische Kategorisierung überlappender Annotationen lässt sich anhand der von DeRose (2004: 11) aufgestellten Übersicht möglicher Relationen zwischen zwei Elementen `<a>` und `<b>` aus unterschiedlichen Hierarchien durchführen (vgl. auch Witt, 2005: 77):

B starts before A

1: <a>.....</a>
...

2: <a>.....</a>
.....

3: <a>.....</a>
.....

4: <a>.....</a>
.....

5: <a>.....</a>
.....

B starts where A starts

6: <a>.....</a>
...

7: <a>.....</a>
.....

8: <a>.....</a>
.....

B starts within A

9: <a>.....</a>
...

10: <a>.....</a>
..... 

11: <a>.....</a>
.....

B starts where A ends

12: <a>....</a>
....

B starts after A

13: <a>....</a>
....

*Abbildung 1: Elementrelationen (DeRose, 2004: 11)*

Zusätzlich zu obiger Aufstellung möglicher Elementrelationen, die eine leicht modifizierte Version der Aufstellung von Durusau & O'Donnell (2002: 2f.) darstellt, berücksichtigt DeRose (2004: 12) Relationen, in denen eines der Elemente leer ist:

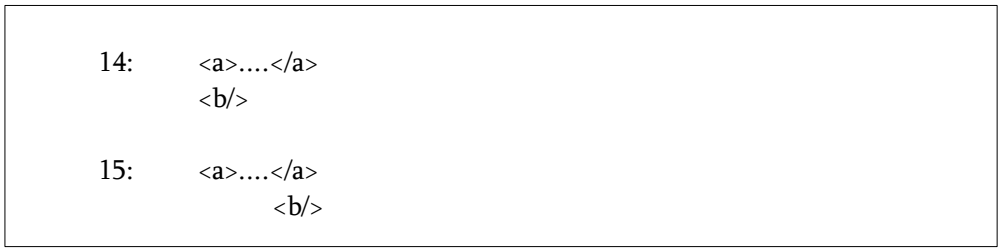



*Abbildung 2: Elementrelationen mit leerem Element (DeRose, 2004: 12)*

In Bezug auf die verschiedenen Konstellationen ist anzumerken, dass die Relationen 1, 5, 9 und 13 aus Abbildung 1 aus technischer Sicht unproblematisch sind, da sie die „well-understood basic tree-relations“ (ebd.), im Fall von 5 und 9 im Speziellen Inklusionsrelationen, darstellen. Die beteiligten Hierarchien könnten demnach in ein gemeinsames XML-Dokument integriert werden, obwohl dieses möglicherweise zu nicht intendierten Dominanz-Beziehungen der beteiligten Elemente führen könnte (vgl. Durusau & O'Donnell, 2002: 3). Auch die Identitätsrelation 7 und diejenigen, in denen nur die Start- oder Endpositionen der Elemente übereinstimmen (2, 4, 6, 8, 10, 12, 14 und 15) sind in herkömmlichem XML abbildbar, indem man die konkurrierenden Element-Tags auf Basis hierarchischer Gesichtspunkte ineinander einbettet (vgl. hierzu auch Abschnitt 4.2.4, S. 53). Auf diese Weise können sogenannte „spurious overlaps“ (Sperberg-McQueen & Huitfeld, 2004: 158), also „the (usually unintentional) use of overlapping markup where the structure of the document can be captured adequately without overlap“ (ebd.), verhindert werden.

Die verbleibenden Relationen 3 und 11, welche als die „klassischen Überlappungen“ (Durusau & O'Donnell, 2002: 3; Marinelli et al., 2008: 4) angesehen werden können, stellen hingegen eine besondere Herausforderung für hierarchisch organisierte, XML-basierte Auszeichnungssprachen dar. Mögliche Lösungsansätze für dieses Pro-

blem werden in Abschnitt 2.3 sowie im späteren Verlauf der Arbeit in Form des Repräsentationsformats XStandoff (Kapitel 3 und 4, ab S. 20) vorgestellt.

Ein zusätzliches Problem stellen sogenannte self-overlaps dar. Hierbei handelt es sich typischerweise um überlappende Elemente, die einer gemeinsamen Hierarchie zugeordnet würden und mit gleichem Namen versehen sind (vgl. z.B. Marinelli et al., 2008: 4):

```
1   <comment>John <comment>likes</comment> Mary</comment>
```

In diesem Fall ist im Nachhinein nicht feststellbar, ob es sich ursprünglich um eine Überlappung der comment-Elemente handelt oder ob die Einbettung eines Elements intendiert ist (ebd.).

Eine weitere Schwierigkeit für hierarchische Modellierungsansätze von Annotationen sind diskontinuierliche Einheiten. Diese werden normalerweise nicht als Teil des Überlappungsproblems betrachtet (vgl. DeRose, 2004: 2), da sie eine andere Art von Struktur repräsentieren. Beispiele für diskontinuierliche Einheiten sind z.B. diskontinuierliche Verbstrukturen im Deutschen (vgl. Pianta & Bentivogli, 2004), oder auch eingebettete Texteinheiten, die zu einer Unterbrechung anderer Einheiten führen (vgl. Sperberg-McQueen & Huitfeld, 2008):

```
1   <p>
2   Alice was beginning to get very
3   tired of sitting by her sister on the bank,
4   and of having nothing to do: once or twice
5   she had peeped into the book her sister
6   was reading, but it had no pictures or
7   conversations in it,
8   <q>and what is the use of a book,</q>
9   thought Alice
10  <q>without pictures or conversation?</q>
11  </p>
```

Die Annotation von Paragraphen (<p>) und indirekter Rede (<q>) an sich kann im oben stehenden Beispiel zwar problemlos erfolgen, jedoch kann z.B. nicht festgehal-

ten werden, dass der Paragraph nur einen Redebeitrag von Alice enthält, und die beiden q-Elemente daher eine Einheit bilden sollten (ebd.).

## 2.3 Repräsentationsformate

Zur Lösung der oben genannten Probleme im Rahmen der Repräsentation multipler Hierarchien wurde eine Reihe von Formaten entwickelt, die sich typologisch in drei Kategorien einteilen lassen:

1. Formate, die verschiedene Hilfskonstruktionen nutzen, um eine Repräsentation mehrerer Hierarchien in einer gemeinsamen Inline-XML-Annotation zu ermöglichen (siehe folgender Abschnitt)
2. Standoff-Ansätze, die in XML realisiert werden, allerdings die verschiedenen Hierarchien getrennt voneinander halten und per Referenzierungsmechanismen Bezüge zwischen Hierarchien untereinander und/oder den Primärdaten herstellen (siehe Abschnitt 2.3.2, S. 14)
3. Auszeichnungssprachen, die nicht dem (bezogen auf Elemente; vgl. S. 4) strikt hierarchischen Baummodell von XML folgen, sondern ausdrucksmächtigere Modelle instantiieren (siehe Abschnitt 2.3.3, S. 16)

### 2.3.1 Annotation durch Standard-Inline-XML

Milestones

Der Ansatz, multiple Hierarchien durch den Einsatz von Milestones in einer gemeinsamen XML-Struktur zu repräsentieren, ist eine der Strategien, die von den Guidelines der TEI (Text Encoding Initiative[2]) vorgeschlagen werden. Der Begriff Milestones wird in dieser Arbeit als Sammelbegriff für die verschiedenen Realisie-

2 http://www.tei-c.org/index.xml

rungsmöglichkeiten des Ansatzes des „Boundary Marking with Empty Elements“ (Burnard & Bauman, 2010: 579ff) verstanden. Das generelle Prinzip besteht darin, klassische Überlappungen durch den Einsatz von „segment boundary elements“ (ebd.) aufzulösen, indem die Start- und End-Tags betroffener Elemente der einen Hierarchie durch leere Elemente ersetzt werden und auf diese Weise eine hierarchische Strukturierung ermöglicht wird.

Zu diesem Zweck können bspw. neue Elementtypen zur Markierung von Elementgrenzen eingesetzt werden, wie z.B. die von der TEI zur Annotation einiger einschlägiger textueller Merkmale vorgeschlagenen <lb>, <pb>, <cb>, <handShift> oder die generischen Elemente <milestone> und <anchor> (vgl. ebd.). Informationen zu den ursprünglichen Elementtypen und der Funktion des jeweiligen ersetzten Tags (Start- oder End-Tag) werden in diesem Fall in Attributen der leeren Elemente aufgeführt (vgl. Witt, 2007: 3). Für das auf Seite 5 genannte Überlappungsbeispiel könnte eine Verwendung von Milestones wie folgt aussehen:

```
1   <doc><b>John <milestone elem="i" position="start" />likes</b>
    Mary<milestone elem="i" position="end" /></doc>
```

Der Milestone-Ansatz existiert in einer Reihe variierender Realisierungen. So könnte z.B. die Benennung der ursprünglich überlappenden Elemente erhalten bleiben. Anstatt generische Elementtypen wie <milestone> für die leeren Elemente, die als Milestones fungieren sollen, zu verwenden, wäre es möglich, die Start- und End-Tags eines überlappenden Elements durch leere Elemente mit dem entsprechenden Elementnamen, in obigem Beispiel <i>, zu ersetzen und zusätzliche Informationen in Form von Attributen hinzuzufügen. Der Einsatz von IDs für Start- und End-Markierungen würde darüber hinaus eine Referenzierung der Milestones untereinander ermöglichen, was in verschiedenen Realisierungen der Milestone-Methode vorgesehen ist, so z.B. in den Formaten Trojan Milestones (vgl. DeRose, 2004: 7), sowie den beiden XML-Serialisierungen von LMNL (siehe S. 17), CLIX und ECLIX (vgl. Marinelli et al., 2008: 7ff; DeRose, 2004: 8ff; Burnard & Bauman, 2010: 580). Das

Format CLIX („Canonical LMNL In XML“; DeRose, 2004: 8) stellt in diesem Zusammenhang vermutlich die extremste Variante des Milestone-Ansatzes dar. In CLIX werden nicht nur die sich überlappenden, sondern alle vorhandenen Annotationselemente als Paare leerer Elemente serialisiert, die durch den Einsatz der Attribute @clix:sID (Start) und @clix:eID (Ende) markiert werden (vgl. Marinelli et al., 2008: 8). Dieses führt zu einer vollkommen flachen Hierarchie von Elementen, und das hierarchische Baummodell von XML wird nahezu vollständig umgangen.

Obwohl mit Hilfe von Milestones z.B. Überlappungen zwischen Hierarchien auf relativ einfache Weise aufgelöst werden können, sind mit der Methode einige Nachteile verbunden. So können bspw. diskontinuierliche Einheiten nicht erfasst werden und es kann zur Entstehung nicht intendierter Dominanzrelationen zwischen Elementen der einzelnen Hierarchien kommen (vgl. ebd.). Des weiteren bedarf es für den Einsatz bzw. die Auswertung von Milestones zusätzlicher Software, um die ursprüngliche hierarchische Struktur der einzelnen Hierarchien zu rekonstruieren, und es muss zwingend eine Entscheidung getroffen werden, welche Elementtypen durch Milestones ersetzt werden sollen („one component must be made primary and the other secondary“; DeRose, 2004: 4). Auch wird durch die strukturelle Veränderung der Annotation der Einsatz evtl. vorhandener Dokumentgrammatiken für die einzelnen Hierarchien (bspw. in Form von DTDs oder XML Schemata) ausgeschlossen.

Fragmentierung

Eine weitere Strategie zur Repräsentation multipler Hierarchien in einer Inline-XML-Notation ist die Fragmentierung sich überlappender Elemente in sogenannte „partial elements“ (Burnard & Bauman, 2010: 582). Durch die Verwendung zusätzlicher Attribute können Elemente dementsprechend, mit dem Ziel, die hierarchische XML-Struktur zu gewährleisten, in mehrere Elemente gesplittet werden (vgl. ebd.; Marinelli et al., 2008: 9f.):

```
1   <doc><b>John <i xml:id="i1" next="#i2">likes</i></b>
    <i xml:id="i2" prev="#i1"> Mary</i></doc>
```

Durch die gegenseitige Referenzierung der Elemente mit Attributen wie @next und @prev (unter Verwendung des von XML vorgesehenen ID/IDREF-Mechanismus) ist es möglich, „virtual joins“ (Burnard & Bauman, 2010: 583) von fragmentierten Elementen zu erstellen, und diese u.U. in einer Art Standoff-Annotation (s.u.) später im XML-Dokument explizit aufzuführen:

```
1   <join result="i" scope="doc" targets="#i1 #i2" />
```

Eine Kombination aus Milestone- und Fragmentierungsmethode sieht Nassourou (2010) in dem Format „Selective AUgmented Fragmentation“ (SAUF) vor. Hier wird der gesamte Textinhalt des fragmentierten Elements in das erste Fragment übernommen und das zweite Fragment zu einem Milestone mit Attribut-Angaben zum ursprünglich umfassten Text (@startPos und @Length) gemacht.

Die bereits für den Milestone-Ansatz genannten Nachteilen gelten nichtsdestotrotz im Großen und Ganzen auch für die Fragmentierung und SAUF, obwohl diese Strategien im Gegensatz zu Milestones durchaus geeignet sind, diskontinuierliche Einheiten zu erfassen.

Multiple Dokumente

Die dritte Methode zur Inline-Repräsentation multipler Hierarchien durch die Verwendung von Standard-XML unterscheidet sich konzeptionell deutlich von den beiden anderen. Hierbei ist jedoch fraglich, ob sie tatsächlich den Inline-XML-Ansätzen zuzuordnen ist. Da das Prinzip allerdings noch weniger als Standoff-Annotation angesehen werden kann, wird es an dieser Stelle aufgeführt. Der Ansatz ist unter verschiedenen Bezeichnungen und in unterschiedlichen Varianten in der Literatur zu finden: „Multiple Encodings of the Same Information“ (Burnard &

Bauman, 2010: 578) sehen bspw. die Annotation multipler Hierarchien auf Basis eines Primärdatums in physisch getrennten XML-Strukturen vor:

```
1   <doc><b>John likes</b> Mary</doc>
```

```
1   <doc>John <i>likes Mary</i></doc>
```

Der Ansatz der „Twin documents“ (Marinelli et al., 2008: 10) ist dem o.g. sehr ähnlich, legt jedoch im Gegensatz zu den TEI Guidelines (Burnard & Bauman, 2010: 578f.) explizit fest, dass nicht nur die Primärdaten in den getrennten XML-Strukturen redundant aufgeführt werden, sondern auch Elementtypen, die in allen Hierarchien gleichermaßen vorhanden sind und somit nicht Teil von Überlappungen sein können („sacred nodes“ im Sinne von Marinelli et al., 2008: 5). Als Erweiterung des Ansatzes stellen Stegmann & Witt (2009) eine Repräsentation multipler XML-Annotationsdokumente mit Hilfe von TEI Feature Structures (vgl. Burnard & Bauman, 2010: 515ff) vor.

Der größte Nachteil der Strategie multipler Dokumente liegt in der redundanten Datenhaltung. Änderungen an Primärdaten oder im Fall der Twin documents an den mehrfach vorhandenen Annotationen sind u.U. recht aufwändig und bedürfen äußerster Vorsicht, um die Entstehung von Inkonsistenzen zu vermeiden.

### 2.3.2 Standoff-XML-Annotationen

Anders als bei klassischen Inline-XML-Repräsentationen, bei denen alle Hierarchien durch direkte Annotationen des Primärdatums ineinander eingebettet werden, findet bei Standoff-Annotationen eine physische Trennung zwischen einer sogenannten „Link Base“ (diese kann das Primärdatum oder eine Inline-XML-Annotation des Primärdatums sein) und den einzelnen Hierarchien statt (vgl. Burnard & Bauman,

2010: 586). Die Elemente der möglicherweise in getrennten Dokumenten gespeicherten Hierarchien haben nach dem Standoff-Ansatz keinen textuellen Inhalt, sondern referenzieren die gewünschten Teile des Primärdatums, bzw. in der anderen Variante die Elemente der Link Base. In folgendem Beispiel wird als Link Base diese Inline-XML-Annotation des Primärdatums angenommen (like.xml):

```
1   <doc xml:id="doc1">John likes Mary</doc>
```

Durch die Nutzung von XML-inhärenten Verlinkungs- und Referenzierungs-Mechanismen (XLink, XPointer, XInclude) lassen sich Teile der Link Base auch aus separaten XML-Dokumenten referenzieren, wie diese simplifizierte Darstellung veranschaulicht:

```
1   <b><xinclude href="like.xml"
     xpointer="string-range(id('doc1'), 0, 10)" /></b>
2   <i><xinclude href="like.xml"
     xpointer="string-range(id('doc1'), 6, 15)" /></i>
```

Die Verwendung von Referenzen auf das Primärdatum oder eine vorhandene Annotation macht es möglich, beliebige Inhalte, also z.B. auch überlappende oder diskontinuierliche Einheiten, zu erfassen. Allgemeiner ausgedrückt können durch Standoff-Ansätze graphen-basierte Strukturen modelliert werden. Als Varianten des Ansatzes sind u.a. die o.g. „virtual joins“ von fragmentierten Elementen (Burnard & Bauman, 2010: 583ff), „Bottom-Up Virtual Hierarchies“ (BUVH, Durusau & O'Donnell, 2002: 4ff), das „Graph Annotation Format“ (GrAF, Ide & Suderman, 2007) und das „Potsdamer Austauschformat für linguistische Annotation“ (PAULA, Chiarcos et al., 2008) anzusehen.

Die Trennung von Primärdaten und Annotationen kann als ein großer Vorteil von Standoff-Ansätzen angesehen werden, da einerseits eine nachhaltige Datenhaltung ermöglicht wird (Primärdaten stehen unverändert für zukünftige Analysen bereit) und andererseits eine verteilte Bearbeitung und Erstellung von Annotationen möglich ist (vgl. Bański, 2010; Goecke et al., 2010: 6).

Ein gewisser Nachteil bezüglich der Nachhaltigkeit von Standoff-Annotationen besteht hingegen möglicherweise darin, dass sie wie die zuvor genannten Repräsentationsmethoden zusätzliche, mehr oder weniger spezialisierte Software-Unterstützung benötigen, um Relationen zwischen den verschiedenen Hierarchien herzustellen (vgl. Rehm et al., 2010). Eine mögliche Lösung für dieses Problem sehen Rehm et al. (2010) in der Erstellung multipler Annotationen, die jeweils die Primärdaten enthalten, und der hieraus resultierenden „sustainability through redundancy" (ebd.). Dieses wiederum hat allerdings den Nachteil, dass Änderungen an den Primärdaten, welche zugegebenermaßen die absolute Ausnahme sein sollten, großen Arbeitsaufwand hervorrufen. Das gilt in ähnlicher Weise für Standoff-Ansätze im Allgemeinen, bei denen Änderungen an den Primärdaten (bzw. der Link Base) weitreichenden Aktualisierungsbedarf für die Referenzen aus den separaten Hierarchien nach sich ziehen können.

### 2.3.3 Nicht-XML-Formate

Neben den XML-basierten Ansätzen zur Repräsentation multipler Hierarchien existiert eine Reihe von Formaten, die die hierarchischen Beschränkungen, und somit die Probleme überlappender Hierarchien in anderer Weise behandeln, wie etwa durch die Integration paralleler Baumstrukturen in einem Dokument oder durch die Definition von Auszeichnungssprachen, die das Baummodell bspw. durch Zulassung multipler Elternknoten erweitern. Letzterem Ansatz liegt folgende Annahme zugrunde: „Overlap can be represented by graphs that are very like trees, but in which nodes may have multiple parents" (Sperberg-McQueen & Huitfeld, 2004: 151). Beide Strategien haben jedoch gemeinsam, dass sie nicht durch die Standard-XML-Notation realisierbar sind, was z.T. schwerwiegende Nachteile mit sich bringt (s.u.).

Unter die erste Kategorie von Formaten, also jene, die konzeptionell parallele Baumstrukturen (Hierarchien) in einer Annotation vereinen und in jeweils unterschied-

licher Weise markieren[3], fallen z.B. das SGML-Feature CONCUR (ISO, 1986; vgl. auch Sperberg-McQueen & Huitfeld, 2004: 145f.) und das sich notationell an XML annähernde MuLaX/XCONCUR (Hilbert et al., 2005). Das generelle Prinzip dieser Ansätze besteht darin, multiple Hierarchien in einem Dokument zu repräsentieren, überlappende Elemente aus verschiedenen Hierarchien zuzulassen, aber sie entsprechend zu kennzeichnen. Die syntaktische Realisierung von Element-Tags und Hierarchie-Zuweisungen variiert zwischen den verschiedenen Ansätzen.

Auch der während des Forschungsprojekts Sekimo („Sekundäre Informationsstrukturierung und vergleichende Diskursanalyse")[4] entwickelte Ansatz der Repräsentation multipler Hierarchien in einer Prolog-Faktenbasis, ist zumindest konzeptuell der ersten Kategorie von Nicht-XML-Formaten zuzuordnen. Hier werden XML-Bestandteile (Elemente, Attribute und Textdaten) aus vorhandenen Annotationshierarchien in Prolog-Prädikate überführt und in einer gemeinsamen Faktenbasis repräsentiert (siehe Abschnitt 3.1, S. 20).

Demgegenüber treffen Ansätze der zweiten Kategorie, also Auszeichnungssprachen mit einem erweiterten Strukturmodell (z.B. multiple Elternknoten), keine Unterscheidung zugunsten vorhandener Hierarchien, sondern ermöglichen die unbeschränkte nicht-hierarchische Einbettung von Elementen. Beispiele hierfür sind die „Layered Markup and Annotation Language" (LMNL; Tennison, 2002), das „Multi-Element Code System" (MECS; z.B. Sperberg-McQueen & Huitfeld, 2004: 150f.), TexMECS (Huitfeld & Sperberg-McQueen, 2003) und die „Freestyle Markup Language" (FML; Pondorf & Witt, 2010).

Da Nicht-XML-Formate allerdings keinen Gebrauch von vorhandenen XML-Technologien oder -Werkzeugen machen können (Marinelli et al., 2008: 5), sind sie zum aktuellen Zeitpunkt wenn überhaupt nur eingeschränkt in korpuslinguistischen

3 Diese Markierung ähnelt häufig der Verwendung von XML Namensräumen (http://www.w3.org/TR/xml-names/) (vgl. Goecke et al., 2010)

4 Projekt A2 der verteilten DFG-Forschergruppe 437 „Texttechnologische Informationsmodellierung" (Förderungszeitraum 2002-2008): http://www.text-technology.de/Sekimo

Forschungsprojekten einsetzbar. Ein weiterer Schwachpunkt ist, dass sie häufig keine nativen Lösungen für diskontinuierliche Einheiten oder self-overlaps vorsehen.

## 2.4 Zwischenfazit

In den vorangegangenen Abschnitten wurde deutlich, dass die Repräsentation multipler Hierarchien vor Herausforderungen steht, die durch eine Reihe von Formaten adressiert werden, aber nur z.T. gelöst werden können, da alle Ansätze mit bestimmten Nachteilen behaftet sind. DeRose (2004: 3) hat eine Reihe von Kriterien zusammengestellt, anhand derer sich die Adäquatheit der verschiedenen Lösungsansätze einschätzen lässt:

1. Menschliche Lesbarkeit
2. Wartbarkeit
3. Verfügbare Implementationen
4. XML-Kompatibilität
5. Unkompliziertheit der Validierung
6. Validierung über Hierarchien hinweg
7. Unkompliziertheit der Formatierung (z.B. durch CSS oder XSL)
8. Unkompliziertheit der Extraktion verschiedener Ansichten
9. Unkompliziertheit der Extraktion hierarchischer Teilmengen
10. Kontinuität des Textinhalts

Bei genauerer Betrachtung wird deutlich, dass keiner der vorgestellten Ansätze alle diese Kriterien vollständig erfüllt. Eine treffende Zusammenfassung des aktuellen Forschungsstands zur Repräsentation multipler, potentiell überlappender Hierarchien liefern Burnard & Bauman (2010):

> „no current solution combines all the desirable attributes of formal simplicity, capacity to represent all occurring or imaginable kinds of structures, suitability for formal or mechanical validation. The representation of non-hierarchical information is thus necessarily a matter of trade-offs among various sets of advantages and disadvantages“ (Burnard & Bauman, 2010: 577)

In den folgenden Kapiteln wird das Format XStandoff inkl. zugehöriger Werkzeuge vorgestellt. XStandoff kann zwar, wie die anderen Ansätze, nicht den Anspruch erheben, alle Kriterien für ein adäquates Repräsentationsformat in optimaler Weise zu erfüllen, aber dennoch versucht es, durch die Kombination der Vorteile unterschiedlicher XML-basierter Ansätze, die Nachteile auf ein Minimum zu reduzieren.

# 3 XStandoff

## 3.1 Entwicklung

Die Grundlagen für die Entwicklung von XStandoff wurden während der ersten Phase des Sekimo-Forschungsprojekts (s.o.) in Form einer Prolog-Faktenbasis zur Repräsentation und Analyse multipler Hierarchien gelegt. Aufbauend auf Arbeiten von Sperberg-McQueen et al. (2000, 2002) werden in diesem Ansatz die Bestandteile vorhandener primärdaten-identischer XML-Annotationen (aus getrennten Dokumenten) in drei verschiedene Arten von Prolog-Prädikaten übersetzt (vgl. Goecke et al., 2010: 7ff, Witt et al., 2005: 107ff)[5]:

Predicates for XML elements:
`node(`*Layer, StartPosition, EndPosition, PositionDocumentTree, ElementName*`).`

Predicates for XML attributes:
`attr(`*Layer, StartPosition, EndPosition, PositionDocumentTree, AttributeName, AttributeValue*`).`

Predicates for PCDATA:
`pcdata_node(`*StartPosition, EndPosition, Character*`).`

*Abbildung 3: Prädikate für XML-Bestandteile in der Prolog-Faktenbasis (Goecke et al., 2010: 8)*

Durch die Integration von XML-Bestandteilen mehrerer Hierarchien (gekennzeichnet durch *Layer*) in die Prolog-Faktenbasis können Informationen zu den Relationen zwischen Elementtypen (aus einer oder verschiedenen Hierarchien) oder auch zwischen den Hierarchien als Ganzes inferiert werden (vgl. Goecke et al., 2010: 7ff). Auf Basis der Informationen zu vorhandenen Inklusions-, Identitäts- und Überlap-

5 Das zuständige Python-Skript XML2PROLOG inkl. Dokumentation ist verfügbar unter http://www.text-technology.de/Sekimo/Resources

pungsrelationen kann eine Unifikation der Layer[6] innerhalb der Prolog-Faktenbasis sowie die Überführung in eine gemeinsame Inline-XML-Repräsentation[7] durchgeführt werden (das XStandoff-Pendant zu dieser Operation ist das XSLT-Stylesheet XSF2inline.xsl, siehe S. 51).

Im weiteren Verlauf des Sekimo-Projekts wurde auf Basis des Grundkonzepts der Prolog-Faktenbasis das generische, XML-basierte Repräsentationsformat für multiple Hierarchien SGF (Sekimo Generic Format) entwickelt. Da XStandoff eine direkte Nachfolgeversion von SGF ist (die Umbenennung erfolgte beim Übergang von der SGF-Version 1.0 nach 1.1), und da es große Überschneidungen zwischen den beiden Formatversionen gibt, werde ich an dieser Stelle nicht ausführlich auf SGF eingehen. Einführungen in SGF sind u.a. zu finden bei Stührenberg & Goecke (2008) und Goecke et al. (2010: 12ff). Die Umbenennung des Repräsentationsformats ist u.a. der Tatsache geschuldet, dass die Änderungen von der Version 1.0 nach 1.1 Inkompatibilitäten zwischen den Versionen verursachen können (vgl. Stührenberg & Jettka, 2009). So beinhaltet XStandoff z.B. leichte Modifikationen der zugrunde liegenden XML Schemata, die hinzugekommene Unterstützung von diskontinuierlichen Segmenten, die Einführung eines all-Namensraums für Elemente, die mit gleichem Typ und gleichen Segmentgrenzen in allen Layern enthalten sind (siehe Abschnitt 3.2, S. 29), und die mögliche Repräsentation einer XStandoff-Instanz als Inline-Variante (siehe Abschnitt 3.2, S. 28).

## 3.2 Einführung in das Format

Das formale Modell von XStandoff lässt sich zurückführen auf das Konzept der Annotation Graphs (vgl. Bird & Liberman, 1999, 2001). Ein Annotation Graph kann in einer minimalen Formalisierung als „directed graph with fielded records on the

6 Das zuständige Prolog-Programm SeMT.pl (Sekimo Merging Tool) inkl. Dokumentation ist verfügbar unter http://www.text-technology.de/Sekimo/Resources

7 Das zuständige Python-Skript PROLOG2XML inkl. Dokumentation ist verfügbar unter http://www.text-technology.de/Sekimo/Resources

arcs and optional time references on the nodes“ (Bird & Liberman, 1999: 1) definiert werden. Die zeitlichen Referenzen lassen sich hierbei im Kontext textueller Daten ohne Probleme auf das Konzept von Zeichenpositionen übertragen.

In Bezug auf die in Abschnitt 2.3 (S. 10ff) vorgestellten Repräsentationsformate für multiple Hierarchien kann XStandoff als eine Kombination aus Standoff-Ansatz und der Verwendung multipler Dokumente kategorisiert werden. Da XStandoff-Instanzen in Standard-XML realisiert werden, können die verwandten Verarbeitungs- und Abfragetechnologien wie XPath, XSLT und XQuery sowie Schemasprachen zur Definition von Dokumentgrammatiken (XML Schema, RelaxNG, etc.) genutzt werden. Die formalen Eigenschaften von XStandoff werden durch die Benennung des Formats reflektiert: X steht hierbei gleichzeitig für *eXtensible* (mit Bezug auf XML) und *eXtended* (im Sinne eines erweiterten Standoff-Ansatzes)[8].

Da XStandoff multiple Hierarchien in einem gemeinsamen XML-Dokument repräsentiert und Teile des Primärdatums durch Referenzierung als virtuelle Blattknoten der Annotationen angesehen werden können, ist das Format konzeptionell als „multi-rooted tree“ zu verstehen (vgl. Goecke et al., 2010: 7; Stührenberg & Goecke, 2008): „A multi-rooted tree consists of several trees that span over the same data leaves“ (Stegmann & Witt, 2009). Dieses gilt jedoch nur solange keine diskontinuierlichen Segmente definiert werden. In diesem Fall wäre eher eine zugrunde liegende Modellierung durch GODDAGs (General Ordered-Descendant Directed Acyclic Graphs; Sperberg-McQueen & Huitfeld, 2004) anzunehmen (vgl. Stührenberg & Jettka, 2009).

Die Grundlage jeder XStandoff-Annotation bildet das Primärdatum. Dieses kann prinzipiell sowohl in Form von Textdaten aber z.B. auch als auditives oder visuelles Sprachdatum vorliegen. Referenzen auf das Primärdatum können entweder anhand von Zeitpunkten oder wie in folgendem Beispiel anhand von Zeichenpositionen erfolgen:

8 vgl. „Overview“ des Formats auf http://www.xstandoff.net

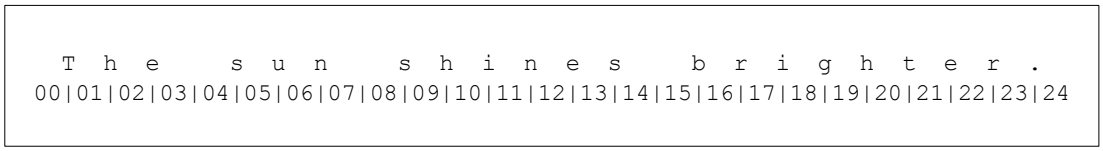



*Abbildung 4: Referenzierung eines Primärdatums durch Zeichenpositionen (http://xstandoff.net/)*

Der Basislayer von XStandoff, dessen Elemente mit der Namensraum-URL http://www.xstandoff.net/2009/xstandoff/1.1 verknüpft sind (s.u.), und für den ein XML Schema implementiert wurde[9], dient als Meta-Auszeichnungssprache bzw. Container für Standoff-Repräsentationen von zuvor als Inline-XML vorliegenden Hierarchien. Zur Unterscheidung des XStandoff-Basislayers, optionalen Metadaten und den einzelnen Annotationslayern werden unterschiedliche Namensräume verwendet (vgl. Stührenberg & Jettka, 2009).

Die Primärdaten können entweder direkt in den Basislayer integriert werden (als Textinhalt des Elements <xsf:textualContent>) oder aber sie werden durch Angabe eines Pfades zu einer externen Datei referenziert (durch <xsf:primaryDataRef uri="..."/>, welches gleichermaßen als Kindelement von <xsf:primaryData> realisiert würde). Der folgende Ausschnitt zeigt die Grundstruktur des Basislayers (Präfix xsf) einer XStandoff-Instanz:

```
1   <xsf:corpusData
2          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3          xmlns:xsf="http://www.xstandoff.net/2009/xstandoff/1.1"
4          xsi:schemaLocation="
5              http://www.xstandoff.net/2009/xstandoff/1.1
6              http://www.xstandoff.net/2009/xstandoff/1.1/xsf.xsd"
7          xsfVersion="1.1" xml:id="sun">
8      <xsf:primaryData start="0" end="24">
9         <xsf:textualContent>The sun shines brighter.</xsf:textualContent>
10     </xsf:primaryData>
11     <xsf:segmentation>
12        <xsf:segment xml:id="seg1" start="0" end="24"/>
13        <xsf:segment xml:id="seg2" start="0" end="3"/>
14        <xsf:segment xml:id="seg3" start="4" end="7"/>
15        <!-- ... -->
16     </xsf:segmentation>
17     <xsf:annotation>
18        <xsf:level xml:id="...">
19           <xsf:layer>
20              <!-- ... -->
```

9 Siehe beiliegende CD: „xsd\xsf_1.1.xsd“ oder http://www.xstandoff.net/2009/xstandoff/1.1/xsf_1.1.xsd (zur Validierung wird ein Parser benötigt, der XSD 1.1 assertions unterstützt)

```
21          </xsf:layer>
22          <!-- weitere Annotationslayer -->
23       </xsf:level>
24       <!-- weitere Level -->
25    </xsf:annotation>
26 </xsf:corpusData>
```

Das konkrete Format des Primärdatums kann bei Verwendung des Elements <xsf:primaryDataRef> mit Hilfe des Attributs @mimeType spezifiziert werden.

In einer XStandoff-Instanz können nicht nur Annotationen für ein einzelnes Primärdatum repräsentiert werden, sondern es besteht zusätzlich die Möglichkeit, ein Korpus zusammenzustellen. Hierzu wird das Wurzelelement <xsf:corpus> verwendet, welches mehrere xsf:corpusData-Elemente als Kindelemente haben kann. Alternativ können durch <xsf:corpusDataRef> auch separate XStandoff-Instanzen referenziert werden, wie dieser Ausschnitt demonstriert:

```
1  <xsf:corpusData
2         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3         xmlns:xsf="http://www.xstandoff.net/2009/xstandoff/1.1"
4         xsi:schemaLocation="
5             http://www.xstandoff.net/2009/xstandoff/1.1
6             http://www.xstandoff.net/2009/xstandoff/1.1/xsf.xsd"
7         xsfVersion="1.1" xml:id="sun">
8     <xsf:corpusDataRef xml:id="c1" uri="c1.xml"
                         mime-type="text/xml"/>
9     <xsf:corpusDataRef xml:id="c2" uri="c2.xml"
10                             mime-type="text/xml"/>
11    <!-- ... -->
12 </xsf:corpus>
```

Die Beziehung zwischen Primärdaten und Annotationen wird über die Konstruktion einer Liste von Segmenten unterhalb des Elements <xsf:segmentation> hergestellt (vgl. Listing S. 23, Zeile 11-16). Die Segmente basieren im Fall von textuellen Primärdaten auf Zeichenpositionen (vgl. Abbildung 4, S. 23). Die Erstellung einer Segmentliste sollte i.d.R. nicht manuell, sondern unter Zuhilfenahme des Stylesheets inline2XSF.xsl erfolgen (siehe Abschnitt 4.2.1, S. 35), das anhand des Primärdatums und einer korrespondierenden Inline-Annotation die Positionierung annotierter Textpassagen ermittelt und so für alle vorhandenen Annotationselemente entsprechende Segmente () mit Angaben über ihre Start- (@start) und End-

positionen (@end) bildet. Durch die Verwendung des optionalen Attributs @type für das Element  können verschiedene Segmenttypen spezifiziert werden, z.B..: char (für Zeicheninhalte), ws (für Whitespaces), pun (für Interpunktion), dur (für zeitliche Einheiten) oder seg (zur Referenzierung anderer Segmente, was die Implementation diskontinuierlicher Einheiten erlaubt; vgl. hierzu Abschnitt 2.2, S. 9).

Die einzelnen Annotationshierarchien werden in Form von leicht modifizierten Versionen ihrer Inline-Repräsentationen unterhalb von xsf:layer-Elementen aufgeführt. Diese wiederum stellen Kindelemente von xsf:level-Elementen dar. Durch die Unterscheidung von <xsf:level> und <xsf:layer> kann die Trennung von konzeptuellen und technischen Realisierungen von Annotationen sichergestellt werden. Ein (Annotations-)Level bezeichnet hierbei den konzeptuellen Gehalt einer Annotation, z.B. die zugrunde liegende linguistische Beschreibungsebene oder auch verschiedene Ansätze zur Erfassung einer einzigen Beschreibungsebene, während ein (Annotations-)Layer auf die technische Umsetzung der Annotation abzielt (vgl. Goecke et al., 2010: 2). Im Sinne des Konzepts der Annotation Graphs (s.o.) könnte ein Layer demzufolge als ein Pfad im Annotation Graph aufgefasst werden, der das komplette Primärdatum umspannt (vgl. ebd.). Die Annahme einer n:m-Beziehung (mit n, m > 0) zwischen Level und Layer (Goecke et al., 2010) gilt auch in XStandoff. Zwar ist aufgrund der Realisierung in einem hierarchisch organisierten XML-Dokument zumindest aus technischer Sicht nur eine 1:m-Beziehung (Eltern-Kind-Beziehung) visualisierbar, d.h. ein <xsf:level> kann beliebig viele xsf:layer-Elemente, also verschiedene Bestandteile oder Perspektiven einer Beschreibungsebene, beinhalten. Aus konzeptueller Sicht ist es jedoch durchaus möglich, mehrere Annotationen verschiedener Beschreibungsebenen (Level) in einem Layer zu integrieren. Dieses könnte durch die Verwendung verschiedener Namensräume verdeutlicht werden, solange es nicht zu hierarchischen Konflikten innerhalb des betroffenen Layers kommt.

Die oben angesprochene Modifikation der Inline-Repräsentationen, welche unter xsf:layer-Elementen eingefügt werden, besteht aus der Ersetzung der textuellen Daten (Primärdaten) innerhalb der Annotation durch xsf:segment-Attribute, welche wiederum die IDs von xsf:segment-Elementen aus der Segmentliste referenzieren. Dieses Vorgehen macht einen entscheidenden Unterschied zu den klassischen Ansätzen multipler Dokumente oder Annotationen (vgl. Abschnitt 2.3.1, S. 13) aus, in denen Primärdaten redundant in allen Annotationslayern aufgeführt werden. Die Referenzierung von Segmenten in XStandoff nutzt den XML-inhärenten ID/IDREF-Mechanismus, um Annotationen mit Segmenten und auf diese Weise mit den Primärdaten zu verlinken (vgl. Stührenberg & Jettka, 2009). Dabei ist es durchaus möglich und auch intendiert, dass Elemente aus verschiedenen Annotationslayern das selbe Segment referenzieren, da somit für die unterschiedlichen Kombinationen von Start- und Endpositionen jeweils nur ein Segment benötigt wird:

```
1   <xsf:corpusData
2          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3          xmlns:xsf="http://www.xstandoff.net/2009/xstandoff/1.1"
4          xsi:schemaLocation="
5              http://www.xstandoff.net/2009/xstandoff/1.1
6              http://www.xstandoff.net/2009/xstandoff/1.1/xsf.xsd"
7          xsfVersion="1.1" xml:id="sun">
8      <xsf:primaryData start="0" end="24">
9         <xsf:textualContent>The sun shines brighter.</xsf:textualContent>
10     </xsf:primaryData>
11     <xsf:segmentation>
12        <xsf:segment xml:id="seg1" start="0" end="24"/>
13        <xsf:segment xml:id="seg2" start="0" end="3"/>
14        <xsf:segment xml:id="seg3" start="4" end="7"/>
15        <xsf:segment xml:id="seg4" start="8" end="14"/>
16        <xsf:segment xml:id="seg5" start="8" end="13"/>
17        <xsf:segment xml:id="seg6" start="13" end="14"/>
18        <xsf:segment xml:id="seg7" start="15" end="21"/>
19        <xsf:segment xml:id="seg8" start="15" end="20"/>
20        <xsf:segment xml:id="seg9" start="20" end="23"/>
21        <xsf:segment xml:id="seg10" start="21" end="23"/>
22     </xsf:segmentation>
23     <xsf:annotation>
24        <xsf:level xml:id="sun-morph">
25           <xsf:layer priority="0"
26              xmlns:morph="http://www.xstandoff.net/morph"
27              xsi:schemaLocation="
28                 http://www.xstandoff.net/morph morph.xsd">
29              <morph:morphemes xsf:segment="seg1">
30                 <morph:m xsf:segment="seg2"/>
31                 <morph:m xsf:segment="seg3"/>
32                 <morph:m xsf:segment="seg5"/>
33                 <morph:m xsf:segment="seg6"/>
34                 <morph:m xsf:segment="seg7"/>
35                 <morph:m xsf:segment="seg10"/>
```

```
36              </morph:morphemes>
37           </xsf:layer>
38        </xsf:level>
39        <xsf:level xml:id="sun-syll">
40           <xsf:layer priority="0"
41              xmlns:syll="http://www.xstandoff.net/syll"
42              xsi:schemaLocation="
43                  http://www.xstandoff.net/syll syll.xsd">
44              <syll:syllables xsf:segment="seg1">
45                 <syll:s xsf:segment="seg2"/>
46                 <syll:s xsf:segment="seg3"/>
47                 <syll:s xsf:segment="seg4"/>
48                 <syll:s xsf:segment="seg8"/>
49                 <syll:s xsf:segment="seg9"/>
50              </syll:syllables>
51           </xsf:layer>
52        </xsf:level>
53     </xsf:annotation>
54  </xsf:corpusData>
```

Wie im Fall der oben aufgeführten XStandoff-Instanz zu sehen ist, lassen sich durch das XStandoff-Format z.B. klassische Überlappungen (vgl. Abschnitt 2.2, S. 8) der repräsentierten Hierarchien erfassen: hier die Überlappung der Segmente seg7 (Zeichenpositionen 15-21) und seg9 (20-23). Mit Standard-Inline-XML könnten diese nicht dargestellt werden:

```
1   * <morph:m>brigh<syll:s>t</morph:m>er</syll:s>
```

Auch identitätskonfligierende Annotationen (seg2 und seg3) können mit Hilfe von XStandoff repräsentiert werden. Zusätzlich hierzu besteht, wie bereits erwähnt, die Möglichkeit, diskontinuierliche Segmente zu berücksichtigen, indem Segmente andere Segmente referenzieren (vgl. das Vorgehen für das „Alice“-Beispiel in Stührenberg & Jettka, 2009).

Zur Erfassung von Metadaten gibt es in XStandoff optionale xsf:meta- und xsf:metaRef-Elemente (letzteres für externe Daten), die als zusätzliche Kindelemente von <xsf:corpus>, <xsf:corpusData>, <xsf:primaryData>, <xsf:resource>, <xsf:annotation>, <xsf:level>, <xsf:layer> und <xsf:log> verwendet werden können. Das optionale xsf:log-Element kann darüber hinaus als Kindelement von <xsf:level> spezifiziert werden, um Informationen zu Bearbeitungsständen und

Änderungen festzuhalten, was eine verteilte Arbeit an XStandoff-Instanzen unterstützt. Für detaillierte Informationen zu den einzelnen Elementtypen kann das XML Schema des XStandoff-Basislayers konsultiert werden[10].

Inline XStandoff

Neben der Standard-XStandoff-Repräsentation sieht das Format eine Inline-Variante vor, in der die einzelnen zuvor getrennt voneinander gespeicherten Hierarchien in einen gemeinsamen Layer integriert werden können. Zu diesem Zweck wird ein Wurzelelement <xsf:inline> definiert, in das alle Elemente aus den einzelnen Annotationslayern eingebettet werden. Ein entsprechender XML-Ausschnitt für die obige XStandoff-Instanz ist auf Seite 56 zu sehen.

Da die Einbettung multipler Hierarchien in eine Inline-Repräsentation, wie im vorherigen Kapitel deutlich geworden ist, durchaus mit einigen Schwierigkeiten versehen ist, steht für die Erstellung der Inline-Variante von XStandoff das Stylesheets XSF2inline.xsl zur Verfügung (siehe Abschnitt 4.2.4, S. 51). Zur Behandlung von klassischen Überlappungen werden milestone-Elemente (vgl. Abschnitt 2.3.1, S. 10) eingefügt, um eine hierarchische Struktur der XML-Notation zu gewährleisten.

Da die Inline-XStandoff-Variante allerdings die Nachteile der o.g. Inline-XML-Repräsentationsformate für multiple Hierarchien (z.B. eingeschränkte Lesbarkeit und fehlende Behandlung diskontinuierlicher Einheiten) beinhaltet und die entstehende Instanz darüber hinaus aufgrund der neuen Strukturierung (Einbettung) der Elemente aus den verschiedenen Hierarchien nicht mit Hilfe von evtl. vorhandenen Dokumentgrammatiken der einzelnen Annotationslayer validiert werden kann, ist sie eher als zusätzliche Orientierungshilfe oder zu Demonstrationszwecken geeignet.

10 Siehe beiliegende CD: „xsd\xsf_1.1.xsd“ oder http://www.xstandoff.net/2009/xstandoff/1.1/xsf_1.1.xsd

Der all-Layer

Eine leichte Verbesserung der Lesbarkeit von XStandoff-Instanzen (sowohl Standard als auch Inline) ist durch die Implementation eines all-Layers zu erreichen. Durch die Angabe des Stylesheet-Parameters all-layer=1 bei der automatisierten Erstellung und Zusammenführung von XStandoff-Instanzen durch inline2XSF.xsl (siehe S. 35) und mergeXSF.xsl (siehe S. 46) kann ein zusätzlicher Annotationslayer konstruiert werden, welcher diejenigen Elemente in sich vereint, die in allen vorhandenen Layern mit gleichen Segmentreferenzen und lokalem Namen zu finden sind („sacred nodes“ nach Marinelli et al., 2008: 5). Ein Beispiel für eine sinnvolle Anwendung ist bei Stührenberg & Jettka (2009) zu finden: hier beinhaltet eine XStandoff-Instanz sechs verschiedene Layer, die als oberste alles umspannende Elemente jeweils die folgende Struktur aufweisen:

```
1   <*:text xsf:segment="seg1">
2      <*:body xsf:segment="seg1">
3         <!-- ... weitere Annotationen ... -->
4      </*:body>
5   </*:text>
```

(mit * = dem jeweiligen Namensraum-Präfix)

Evtl. vorhandene Attribute der betroffenen Elemente werden unter Beibehaltung ihres Namensraums in den all-Layer übernommen. Durch die generelle Möglichkeit, Annotationselemente in einen eigenen Layer mit dem Namensraum-Präfix all (URL: http://www.xstandoff.net/2009/all) auszulagern, kann nicht nur die Lesbarkeit, v.a. der Inline-XStandoff-Variante, verbessert, sondern auch die Gesamtgröße der XStandoff-Datei u.U. deutlich reduziert werden. Da der semantische Gehalt von Annotationen allerdings nicht automatisch inferiert werden kann, wird es dem Benutzer überlassen, einen derartigen Layer anlegen zu lassen.

Weitere Informationen zum all-Layer und zu den Problemen, die während seiner Erstellung zu lösen sind, finden sich im Abschnitt 4.2.4 „Erstellung von Inline-XStandoff: XSF2inline.xsl“ auf Seite 51.

Trotz der offensichtlichen Vorzüge des Repräsentationsformats XStandoff (wie z.B. die Verwendung von XML, Möglichkeiten der Validierung, nicht redundante Datenhaltung) gibt es durchaus Nachteile, die speziell dem Prinzip der Standoff-Annotation geschuldet sind. Diese betreffen v.a. die leicht eingeschränkte Lesbarkeit durch die erforderlichen Elemente des XStandoff-Basislayers und die manuelle Bearbeitung der Primärdaten und Annotationen. Im Allgemeinen ist eine manuelle Erstellung von XStandoff-Instanzen aufgrund der Komplexität des Formats (bspw. die intensive Nutzung des ID/IDREF-Mechanismus) nur eingeschränkt realisierbar. Daher werden weitere Werkzeuge zur Erstellung und Bearbeitung von XStandoff benötigt, die in Form von XSLT-Stylesheets vorliegen und im folgenden Kapitel vorgestellt werden.

# 4 Verarbeitung von XStandoff mit XSLT

Im Rahmen der Entwicklung des XStandoff-Formats sind XSLT 2.0-Stylesheets[11] entstanden, die die automatische Erstellung, Weiterverarbeitung und Visualisierung von XStandoff-Instanzen ermöglichen. Die nachfolgend vorgestellten Stylesheets, ihre Quellcode-Dokumentationen sowie Beispiele für XStandoff-Instanzen und -Visualisierungen sind auf der beiliegenden CD sowie auf http://www.jettka.de/xsf/ einzusehen.

Mit Ausnahme des Stylesheets inline2XSF.xsl, welches produkt-spezifische Erweiterungen des Saxon XSLT-Prozessors nutzt (vgl. Abschnitt 4.2.1, S. 35), können die XSLT-Transformationen prinzipiell mit jedem beliebigen XSLT-Prozessor, der die XSLT-Version 2.0 unterstützt, durchgeführt werden. Trotzdem wird die Verwendung des Saxon XSLT-Prozessors empfohlen, da die Stylesheets mit seiner Hilfe entwickelt und erfolgreich getestet wurden[12].

Die XSLT-Stylesheets des XStandoff Toolkits (vgl. Stührenberg & Jettka, 2009), welche die rechnergestützte Erstellung und Verarbeitung von XStandoff-Instanzen ermöglichen, wurden im Rahmen der Masterarbeit z.T. erweitert bzw. optimiert. Des weiteren ist nun ein Stylesheet zur Visualisierung von XStandoff-Instanzen verfügbar, sodass z.Z. mit Hilfe der nachfolgend erläuterten fünf XSLT-Stylesheets die folgende Aufgaben erledigt werden können:

- die Erstellung von XStandoff-Instanzen aus einer XML-Inline-Annotation

11 Eine umfassende Einführung in die Extensible Stylesheet Language Transformations (XSLT) 2.0 ist bei Kay (2007, 2008) zu finden

12 Saxon-B 9.1.0.8 sowie zu Testzwecken der schema-aware Prozessor Saxon-EE 9.3.0.4

- die Zusammenführung mehrerer XStandoff-Instanzen, welche auf dem gleichen Primärdatum basieren, in eine gemeinsame Instanz
- die Entfernung und Extraktion einzelner Annotationslayer aus einer XStandoff-Instanz
- die Konvertierung einer XStandoff-Instanz in eine XStandoff-Inline-Annotation
- die Visualisierung einer XStandoff-Instanz in einer SVG-Datei

Bevor in Abschnitt 4.2 die vorhandenen Stylesheets im Detail vorgestellt werden, möchte ich zunächst auf die umfassenden Dokumentationen des Quellcodes hinweisen und in diesem Zusammenhang einen kleinen Exkurs zu zwei unterschiedlichen Methoden der Quellcode-Dokumentation für XSLT-Stylesheets vornehmen.

## 4.1 Dokumentation von XSLT-Stylesheets

Gerade komplexe Programme bedürfen einer umfassenden und detaillierten Dokumentation, um Nutzern und kooperierenden Personen die Möglichkeit zu geben, sich innerhalb des Quelltexts orientieren zu können. Für die Dokumentation von XSLT-Stylesheets stehen in diesem Zusammenhang verschiedene Ansätze zur Verfügung. Sie verfolgen zumeist das gleiche Prinzip: die Dokumentation der einzelnen Stylesheet-Bestandteile (Templates, Funktionen, Parameter, Variablen, etc.) innerhalb des Quellcodes mit Hilfe eines mehr oder weniger restringierten Inventars von speziellen XML-Elementen.

Die dokumentierten Stylesheets können so selbst als Eingabedateien für XSLT-Transformationen dienen, welche umfassende Dokumentationen z.B. im HTML-Format erzeugen. Hierbei zeigt sich, dass es sehr von Vorteil ist, dass XSLT-Stylesheets selbst in XML verfasst werden. Zwei Ansätze, die im Zusammenhang der

Dokumentation der Stylesheets für XStandoff genutzt wurden, möchte ich im Folgenden vorstellen.

XSLTDoc

Das Dokumentationswerkzeug XSLTDoc ist frei verfügbar[13] und besteht aus einem Dokumentationsformat und XSLT-Stylesheets, welche als Eingabedateien dokumentierte Stylesheets erwarten und sie in HTML-Dateien transformieren.

Nachfolgend ist ein Beispiel für einen Dokumentationsblock im XSLTDoc-Format zu sehen:

```
1   <xd:doc xmlns:xd="http://www.pnp-software.com/XSLTdoc">
2     <xd:short>Kurzbeschr. mit <code>HTML Tags</code>.</xd:short>
3     <xd:detail>
4       Hier folgen <b>weitere Details</b>, die in einem separaten
5       Abschnitt der Dokumentation aufgeführt werden.
6     <xd:detail>
7   </xd:doc>
8   <xsl:...>
```

Die XML-Elemente, welche für die Dokumentation vorgesehen sind, sind mit dem Namensraum-Präfix xd, und somit mit der Namensraum-URL http://www.pnp-software.com/XSLTdoc verknüpft. Auf diese Weise kann die Dokumentation direkt im Quelltext erfolgen, ohne Einfluss auf die übrigen Inhalte zu nehmen. Bei einer XSLT-Transformation durch das dokumentierte Stylesheet (z.B. der Überführung einer XML-Inline-Annotation nach XStandoff; inline2XSF.xsl) werden die Elemente aus dem xd-Namensraum sowie deren Kindelemente nicht beachtet. Innerhalb der Dokumentationselemente kann XHTML verwendet werden, was einen großen Einfluss des Programmierers auf die Formatierung der später zu erzeugenden Dokumentation ermöglicht.

Mittels einer Konfigurationsdatei[14] können mehrere Stylesheets in eine gemeinsame Dokumentation integriert werden. Die komplette, mit Hilfe von XSLTDoc erzeugte

13 http://www.pnp-software.com/XSLTdoc/

14 Siehe beiliegende CD („Dokumentation\XSFdocConfig.xml“) oder auch http://www.jettka.de/xsf/doc/XSFdocConfig.xml

Dokumentation der XStandoff-Stylesheets ist auf der beiliegenden CD in dem Ordner „Dokumentation\XSLTDoc“ zu finden.

Das XSLTDoc-Format wurde aufgrund der freien Verfügbarkeit der zugehörigen Stylesheets für die Dokumentation der XStandoff-Stylesheets verwendet, da die Dokumentation auf diese Weise bei Bedarf von anderen Personen ohne großen Aufwand erweitert werden könnte. Im Anhang (S. 76f.) sind zwei Beispielausschnitte einer Dokumentation zu sehen, die mit Hilfe von XSLTDoc generiert wurde.

Dokumentation mit <oXygen/>

Eine Alternative zur Dokumentation von XSLT-Stylesheets mit XSLTDoc ist das im Oxygen XML Editor[15] integrierte Dokumentationswerkzeug. Die Dokumentation der einzelnen Stylesheet-Bestandteile erfolgt hier in ähnlicher Weise wie bei XSLTDoc, allerdings basiert das Oxygen-Format auf einer Kombination eigener Elementtyp-Definitionen, einer Teilmenge von Docbook 5-Elementen und einer Teilmenge von DITA-Elementen (vgl. SyncRO Soft Ltd., o.A.: 175). Genau wie bei XSLTDoc kann zur Dokumentation XHTML genutzt werden. Allerdings ist die Oxygen Dokumentation etwas ausgereifter gestaltet als XSLTDoc, da durch den Einsatz von JavaScript einzelne Abschnitte und Bestandteile der resultierenden Dokumentation in HTML weg und wieder hinzu geschaltet werden können. Der Einsatz der Oxygen Dokumentation ist jedoch nur möglich, sofern man über eine Lizenz des Editors verfügt.

Die Generierung einer Dokumentation im HTML-Format, welche durch die Anwendung von Oxygen beiliegenden Stylesheets erfolgt, kann direkt im Editor über das Menü „Werkzeuge > Generiere Dokumentation > XSLT Stylesheet Dokumentation...“ ausgeführt werden. Darüber hinaus steht eine Batch-Datei namens stylesheetDocumentation.bat im Oxygen Installationsverzeichnis zur Verfügung, mit deren Hilfe man eine Dokumentation von der Kommandozeile aus gene-

15 http://www.oxygenxml.com/

rieren kann (vgl. SyncRO Soft Ltd., o.A.: 179). Diese Möglichkeit hat sich als sehr nützlich erwiesen, da durch sie unter zu Hilfenahme eines weiteren von mir erstellten XSLT-Stylesheets[16] und durch eine leichte Modifikation der Batch-Datei (zwischengeschaltete XSLT-Transformation des XSLTDoc-Formats in das Oxygen-Format durch o.g. Stylesheet) die Oxygen Dokumentation auch auf Basis des XSLTDoc-Formats mit relativ geringem Aufwand zu generieren war. Die mit Hilfe von Oxygen erzeugten Dokumentationen der nachfolgend erläuterten XSLT-Stylesheets sind in dem Ordner „Dokumentation\OxygenDoc“ auf der beiliegenden CD einzusehen. Ein Beispielausschnitt ist im Anhang auf Seite 78 aufgeführt.

## 4.2 Stylesheets für XStandoff

### 4.2.1 Erstellung von XStandoff-Instanzen: inline2XSF.xsl

Die Hauptaufgabe des Stylesheets inline2XSF.xsl[17] ist die Überführung einer Inline-XML-Annotation in eine XStandoff-Instanz. Als Referenz für die Ermittlung von Segmentgrenzen dient ein Primärdatum in Form einer txt-Datei, welche die unannotierten reinen Textdaten enthält. Das Vorgehen bei der Transformation kann anhand der folgenden Ausgangs-Inline-Annotation (phr-role.xml) demonstriert werden:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <phr:s xmlns:phr="http://www.xstandoff.net/phrase"
           xmlns:role="http://www.xstandoff.net/gram-role">
3      <phr:np>
4         <phr:pron>
5            <role:subj>This</role:subj>
6         </phr:pron>
7      </phr:np>
8      <phr:vp>
9         <phr:v>is</phr:v>
10        <phr:np>
11           <role:obj>
```

16 Siehe CD: „xsl\XSLTDoc2OxygenDoc.xsl“ oder auch http://www.jettka.de/xsf/doc/XSLTDoc2OxygenDoc.xsl

17 Dieses und alle nachfolgend in der Masterarbeit vorgestellten Stylesheets sind auf der beiliegenden CD im Ordner „xsl“ zu finden.

```
12              <phr:det>a</phr:det>
13              <phr:n>sentence</phr:n>
14           </role:obj>
15        </phr:np>
16     </phr:vp>.
17  </phr:s>
```

Die Textdatei mit den Primärdaten (phr-role.txt) hat folgenden Inhalt:

```
1   This is a sentence.
```

Diese Informationen sollen nun genutzt werden, um eine XStandoff-Instanz zu erstellen, welche für jeden vorhandenen Namensraum in der Ausgangsannotation (hier gekennzeichnet durch die Präfixe phr und role) einen Annotationslayer anlegt[18], in dem die jeweils zugehörigen Elemente inkl. Referenzen zu ihrer Position in Bezug auf das Primärdatum aufgeführt werden.

Sollte kein Primärdatum vorhanden sein, müsste dieses zuvor manuell angelegt werden. Alternativ besteht die Möglichkeit, mit Hilfe des Stylesheets und der Angabe des Stylesheet-Parameters primary-data=create (create wird dann anstelle des relativen Pfades zur txt-Datei angegeben; s.u.) automatisch eine Primärdatendatei auf Basis der textuellen Inhalte der Ausgangsdatei (phr-role.xml) erstellen zu lassen. Dieses Vorgehen birgt jedoch einige Schwierigkeiten, wie die äußerst komplexe automatische Detektion von Whitespaces (Zeilenumbrüche können z.B. nicht sinnvoll inferiert werden). Das kann dazu führen, dass nicht in jedem Fall ein optimales bzw. gewünschtes Ergebnis zu erzielen ist. Aus diesem Grund sollte die Methode der automatischen Erstellung von Primärdaten nur in Ausnahmefällen und mit eingehender, späterer Prüfung der inferierten Primärdaten gewählt werden.

Die angestrebte XStandoff-Instanz wird durch die Ausführung der XSLT-Transformation mit Hilfe des Stylesheets inline2XSF.xsl erstellt, deren Aufruf, unter der Annahme dass alle Dateien im selben Verzeichnis liegen, folgendermaßen aussehen könnte:

18 Der u.U. vorhandene default- oder leere Namensraum (xmlns="") wird als separater Namensraum behandelt

```
saxon -o phr-role-xsf.xml phr-role.xml inline2XSF.xsl primary-data=phr-role.txt
```

Bei dem Aufruf der Transformation steht eine Reihe weiterer optionaler Parameter zur Verfügung, mit deren Hilfe der Benutzer das genaue Vorgehen des Stylesheets beeinflussen kann[19]. So besteht z.B. die Möglichkeit, mit Hilfe eines XPath-Ausdrucks, der als Wert des Parameters $virtual-root spezifiziert wird, einen Knoten der Ausgangsannotation anzugeben, der als Wurzelelement des Annotationslayers angesehen werden soll. Auf diese Weise können Probleme aufgrund zusätzlicher in der Ausgangsannotation vorhandener textueller Daten (z.B. Metadaten, die nicht im Primärdatum auftauchen) vermieden werden. Die für die obige Inline-Annotation in der Standardeinstellung resultierende XStandoff-Instanz (phr-role-xsf.xml) hat folgendes Erscheinungsbild:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsf:corpusData xsfVersion="1.1" xml:id="phr-role"
      xmlns:xsf="http://www.xstandoff.net/2009/xstandoff/1.1">
   <xsf:primaryData start="0" end="19">
      <xsf:primaryDataRef uri="phr-role.txt"/>
   </xsf:primaryData>
   <xsf:segmentation>
      <xsf:segment xml:id="seg1" start="0" end="19"/>
      <xsf:segment xml:id="seg2" start="0" end="4"/>
      <xsf:segment xml:id="seg3" start="5" end="18"/>
      <xsf:segment xml:id="seg4" start="5" end="7"/>
      <xsf:segment xml:id="seg5" start="8" end="18"/>
      <xsf:segment xml:id="seg6" start="8" end="9"/>
      <xsf:segment xml:id="seg7" start="10" end="18"/>
   </xsf:segmentation>
   <xsf:annotation>
      <xsf:level xml:id="phr-role-level1">
         <xsf:layer priority="0"
            xmlns:phr="http://www.xstandoff.net/phrase">
            <phr:s xsf:segment="seg1">
               <phr:np xsf:segment="seg2">
                  <phr:pron xsf:segment="seg2"/>
               </phr:np>
               <phr:vp xsf:segment="seg3">
                  <phr:v xsf:segment="seg4"/>
                  <phr:np xsf:segment="seg5">
                     <phr:det xsf:segment="seg6"/>
                     <phr:n xsf:segment="seg7"/>
                  </phr:np>
               </phr:vp>
            </phr:s>
         </xsf:layer>
      </xsf:level>
      <xsf:level xml:id="phr-role-level2">
         <xsf:layer priority="0"
            xmlns:role="http://www.xstandoff.net/gram-role">
            <role:subj xsf:segment="seg2"/>
```

19 Siehe Dokumentation des Stylesheets (beiliegende CD: „Dokumentation\OxygenDoc\inline2XSF.html“ oder auch http://www.jettka.de/xsf/doc/OxygenDoc/inline2XSF.html)

```
38              <role:obj xsf:segment="seg5"/>
39           </xsf:layer>
40        </xsf:level>
41     </xsf:annotation>
42  </xsf:corpusData>
```

Die Überführung einer Inline-Annotation (z.B. phr-role.xml) nach XStandoff (z.B. phr-role-xsf.xml) kann konzeptuell in drei Teilprobleme zerlegt werden, auf die ich im Folgenden etwas ausführlicher eingehen werde:

1. Überprüfung der Primärdatenidentität von Ausgangs-Inline-Annotation und Primärdatum.
2. Erstellung einer Segmentliste (<xsf:segmentation>). Diese entspricht der Zuordnung der Annotationselemente zu Zeichenpositionen.
3. Erstellung (leicht modifizierte Kopie) der einzelnen Annotationslayer (unterhalb von xsf:layer-Elementen).

Überprüfung der Primärdatenidentität

Die Primärdatenidentität von Primärdatum (txt-Datei) und XML-Annotation ist eine Grundvoraussetzung für die Ermittlung von Segmentgrenzen (Positionierung von Start- und End-Tags) für die einzelnen vorhandenen Elemente der Annotation. Aufgrund der Zulässigkeit zusätzlicher Whitespaces in XML-Dateien (Einrückungen, Zeilenumbrüche) sind in dem Stylesheet inline2XSF.xsl drei verschiedene Abstufungen für die Strenge der Primärdatenidentitätsprüfung vorgesehen. Diese kann der Benutzer bei Bedarf durch die Angabe des Stylesheet-Parameters pd-check=(lax|middle|strict) selbst festlegen. Per Default ist die Strenge auf middle eingestellt.

Ein Minimum an Primärdatenidentität wird durch den Wert lax überprüft. Hierbei müssen alle Nicht-Whitespace-Zeichen aus dem Primärdatum in gleicher Reihenfolge in den konkatenierten Textknoten der XML-Annotation enthalten sein. Für Whitespaces, die im Primärdatum, aber nicht in der XML-Datei, vorhanden sind,

werden in jedem Fall, so auch bei den anderen beiden Methoden, Warnungen ausgegeben.

Der Default-Wert middle besagt, dass alle Zeichen aus dem Primärdatum, also auch Whitespaces, in gleicher Reihenfolge in den Textknoten der XML-Datei zu finden sein müssen, wobei allerdings auch weitere Whitespaces erlaubt sind.

Bei der Methode strict hingegen sind zusätzliche Whitespaces nur in Textknoten, die Teil eines gemischten Inhaltsmodells sind (es gibt benachbarte Elementknoten), und an den Rändern von terminalen Textknoten (nicht Teil eines gemischten Inhaltsmodells) erlaubt. Sollte die Primärdatenidentitätsprüfung nicht erfolgreich abgeschlossen werden, bricht die XSLT-Transformation ab und gibt Hinweise auf die jeweiligen Gründe. Nach erfolgreicher Prüfung der Primärdatenidentität wird die Erstellung einer XStandoff-Instanz mit der Lösung des nächsten Teilproblems fortgesetzt.

Erstellung und Zuordnung von Segmenten

Die Zusammenstellung einer Liste von Segmenten, die im späteren Verlauf als Referenzpunkte für die Positionsangaben der Elemente aus den verschiedenen Annotationslayern dienen, erfolgt aus Gründen der Komplexitätsreduktion in zwei Phasen. In der ersten Phase werden die Zeichen, die sich in terminalen Textknoten (nicht mixedContent) in der Ausgangsannotation befinden, durch Angaben zu ihrer Positionierung in Bezug auf das Primärdatum ersetzt. Hierbei wird mittels Iteration jedes einzelne Zeichen des Primärdatums mit dem Inhalt der terminalen Textknoten der XML-Datei verglichen und ein Zwischenergebnis erstellt, das die benötigten Zeichenpositionen in Form von leeren Elementen (<c p="..."/>) enthält. Einige Zeichenpositionen (z.B. 1 bis 3) sind für die spätere Segmentierung unerheblich, da in ihrer Nachbarschaft keine Elementtags vorhanden sind. Daher werden diese im Zwischenergebnis nicht berücksichtigt:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <phr:s xmlns:phr="http://www.xstandoff.net/phrase"
3          xmlns:role="http://www.xstandoff.net/gram-role"
4          xmlns="http://www.xstandoff.net/2009/xstandoff/1.1">
5      <phr:np>
6         <phr:pron>
7            <role:subj>
8               <c p="0"/>
9               <c p="4"/>
10           </role:subj>
11        </phr:pron>
12     </phr:np>
13     <phr:vp>
14        <phr:v>
15           <c p="5"/>
16           <c p="7"/>
17        </phr:v>
18        <phr:np>
19           <role:obj>
20              <phr:det>
21                 <c p="8"/>
22                 <c p="9"/>
23              </phr:det>
24              <phr:n>
25                 <c p="10"/>
26                 <c p="18"/>
27              </phr:n>
28           </role:obj>
29        </phr:np>
30     </phr:vp>.
31  </phr:s>
```

In der zweiten Phase werden die Zeichenpositionen für Zeichen von Textknoten, die sich in einem gemischten Inhaltsmodell (mixedContent) befinden, ermittelt und festgehalten. Betroffen sind hiervon im Fall der Ausgangsannotation phr-role.xml (siehe S. 35) und dem zugehörigen Primärdatum phr-role.txt die Leerzeichen an den Positionen 4-5 (Zeile 11/12), 7-8 (Zeile 20/21), 9-10 (Zeile 28/29) sowie der Punkt an der Position 18-19 (Zeile 37/38):

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <phr:s xmlns:phr="http://www.xstandoff.net/phrase"
3          xmlns:role="http://www.xstandoff.net/gram-role"
4          xmlns="http://www.xstandoff.net/2009/xstandoff/1.1">
5      <phr:np>
6         <phr:pron>
7            <role:subj>
8               <c p="0"/>
9               <c p="4"/>
10           </role:subj>
11           <c p="4"/>
12           <c p="5"/>
13        </phr:pron>
14     </phr:np>
15     <phr:vp>
16        <phr:v>
```

```
17          <c p="5"/>
18          <c p="7"/>
19       </phr:v>
20       <c p="7"/>
21       <c p="8"/>
22       <phr:np>
23          <role:obj>
24             <phr:det>
25                <c p="8"/>
26                <c p="9"/>
27             </phr:det>
28             <c p="9"/>
29             <c p="10"/>
30             <phr:n>
31                <c p="10"/>
32                <c p="18"/>
33             </phr:n>
34          </role:obj>
35       </phr:np>
36    </phr:vp>
37    <c p="18"/>
38    <c p="19"/>
39 </phr:s>
```

Auf Basis dieses Zwischenergebnisses, das während der Transformation in der globalen Variable $segmentation-base gespeichert wird[20], kann nun auf der einen Seite mit relativ wenig Aufwand durch folgende XSLT-Anweisung eine Liste von Elementen der Form  (vgl. Listing auf Seite 37, Zeile 8-15) erstellt werden:

```
1  <xsl:for-each select="
2    distinct-values($segmentation-base//*/concat(
3     descendant::xsf:c[1]/@p, '-', descendant::xsf:c[last()]/@p)
4     [. != '-']">
5     <xsf:segment xml:id="{concat('seg', position())}"
6      start="{tokenize(., '-')[1]}" end="{tokenize(., '-')[2]}"/>
7  </xsl:for-each>
```

Auf der anderen Seite dient das Zwischenergebnis der späteren Kopie der Elemente in die einzelnen Annotationslayer, wie im Folgenden deutlich wird.

20 Zum Zweck der Reduktion des Speicherbedarfs wurde die globale Variable durch die Saxon-Erweiterung saxon:assignable zu einer überschreibbaren Variable gemacht. Sie Zwischenergebnisse der einzelnen Phasen werden so jeweils überschrieben. Dieses Vorgehen ist standardmäßig in XSLT 2.0 aufgrund möglicherweise entstehender Side-Effects nicht vorgesehen, führt aber zu einer deutlichen Steigerung der Transformationsgeschwindigkeit.

Erstellung der einzelnen Annotationslayer

Anhand der im temporären Zwischenergebnis vorhandenen c-Elemente, die die Start-Position eines Elements (XPath-Ausdruck: descendant::c[1]/@p) sowie seine End-Position (descendant::c[last()]/@p) repräsentieren, kann die zugehörige Segment-ID aus der Segmentliste für alle Elemente der Ausgangsannotation ermittelt und als Referenz in das Attribut @xsf:segment übernommen werden. Durch das rekursive Durchlaufen des Zwischenergebnisses für jeden einzelnen vorhandenen Namensraum (außer xsf) und das Kopieren der zugehörigen Elemente, können auf diese Weise die Annotationslayer (unterhalb von <xsf:layer>) in die XStandoff-Instanz übernommen werden (vgl. XML-Listing auf Seite 37, Zeilen 19-33 und 36-40).

Abschließend möchte ich noch auf zwei spezielle Schwierigkeiten hinweisen, die bei der Transformation einer Inline-Annotation nach XStandoff entstehen: die Behandlung leerer Elemente und ein Problem im Zusammenhang mit Whitespaces.

Leere Elemente

Die Berücksichtigung evtl. in der Ausgangsannotation vorhandener leerer Elemente wurde aus dem oben erläuterten Vorgehen zur Erstellung einer XStandoff-Instanz zunächst ausgeblendet, da hier einige Besonderheiten zu beachten sind. Für das Stylesheet inline2XSF.xsl kann der Nutzer selbst bestimmen, ob und wenn ja welche leeren Elemente bei der Transformation berücksichtigt werden. Dieses geschieht durch den Stylesheet-Parameter include-empty-elements, mit dessen Hilfe man Namensraum-Präfixe spezifizieren kann, deren zugehörige leere Elemente in die Layer aufgenommen werden sollen. Standardmäßig ist der Wert des Parameters auf #all eingestellt, was bedeutet, dass alle leeren Elemente berücksichtigt werden.

Die besondere Schwierigkeit im Zusammenhang mit leeren Elementen liegt in ihrer Zuordnung zu bestimmten Positionen. Da sie keinen Textinhalt haben, steht zwar

von vornherein fest, dass ihre Start- und Endpositionen gleich sind; jedoch kann es problematisch sein, ihre Position mit Bezug auf das Primärdatum korrekt zu ermitteln, wenn sie in unmittelbarer Nachbarschaft zu einer Position stehen, mit der ein Whitespace referenziert wird. Dieses Problem verdeutlicht das folgende Beispiel:

```
1   <s xmlns="http://www.xstandoff.net/phrase"
2      xmlns:phon="http://www.xstandoff.net/phon">
3      <np>
4         <pron>This</pron>
5      </np>
6      <phon:pause/>
7      <vp>
8         <v>is</v>
9         <np>
10           <det>a</det>
11           <n>sentence</n>
12        </np>
13     </vp>.
14  </s>
```

Das leere Element <phon:pause/> (dessen Semantik in diesem Zusammenhang keine Rolle spielt) in Zeile 6 ist nicht eindeutig einer bestimmten Zeichenposition des Primärdatums zuzuordnen. Die beiden alternativen Positionen sind 4 und 5, da das Leerzeichen aus dem Primärdatum zwischen This und is vor oder nach dem leeren Element lokalisiert werden könnte. Um die XSLT-Transformation nicht scheitern zu lassen, wird dem leeren Element die jeweils letzte mögliche Position (in diesem Fall 5) zugewiesen. Dieses Vorgehen hat allerdings keine empirische Motivation und zeigt eine grundsätzliche Schwierigkeit der Standoff-Annotation leerer Elemente auf.

Eine mögliche Lösung für das Problem liegt in der expliziten Annotation von Whitespaces. Wäre das Leerzeichen an der Position 4-5 mit einem Element wie <ws> </ws> annotiert, welches vorzugsweise in einem separaten Namensraum stehen sollte, so wäre eindeutig feststellbar, welcher Position das leere Element <phon:pause/> zuzuordnen ist. Durch die Vergabe eines eigenen Namensraums für Elemente vom Typ ws würden diese während der Transformation in einem eigenen Annotationslayer (<xsf:layer>) gespeichert und könnten bei Bedarf im Nachhinein entfernt werden (vgl. extractXSFcontent.xsl, S. 49). Nichtsdestotrotz stellt dieses

Vorgehen, welches mit erheblichem zusätzlichem Aufwand verbunden sein kann, nur ein Hilfskonstrukt dar.

Ein weiteres Problem ist die Reihenfolge leerer Elemente aus verschiedenen Namensräumen, die später in verschiedenen Annotationslayern untergebracht werden. Nimmt man an, dass in der Ausgangsannotation zwei direkt benachbarte leere Elemente aus unterschiedlichen Namensräumen zu finden sind, die bei der inline2XSF-Transformation der gleichen Position zugeordnet werden, so geht die Information zu ihrer Reihenfolge durch die Speicherung in verschiedenen Layern verloren. Diese ist nicht ohne weiteres (z.B. die erneute Konsultation der Inline-Annotation) wiederherzustellen.

Das Whitespace-Problem

Ein weiteres Problem bei der Transformation von Inline-Annotationen nach XStandoff besteht in der korrekten Zuordnung von Zeichenpositionen zu Whitespaces (Leerzeichen, Tabulatoren, Zeilenumbrüche).

Das Stylesheet inline2XSF.xsl steht vor der Herausforderung, jedem Zeichen bzw. jeder Zeichenposition aus dem Primärdatum ein Zeichen aus den Textknoten der korrespondierenden XML-Eingabedatei zuzuordnen. Bei Nicht-Whitespace-Zeichen stellt dieses kein Problem dar, da diese eindeutig identifiziert werden können. Da allerdings, wie bereits erwähnt, durch die Annotation der Primärdaten aufgrund von Einrückungen und zusätzlichen Zeilenumbrüchen mehr Whitespaces in der Eingabedatei vorhanden sein können als im Primärdatum, ist die Zuordnung von Whitespaces eine sehr komplexe Aufgabe. Im Extremfall kann es sein, dass die Zeichenpositionen für Whitespaces aus dem Primärdatum nicht auf Basis der in der XML-Eingabe vorkommenden Zeichen extrahiert werden können. So ist z.B. an der Position, die dem Leerzeichen zwischen „This“ und „is“ entsprechen würde, in der folgenden möglichen XML-Eingabedatei kein Leerzeichen vorhanden (zwischen den Tags </np> und <vp> in Zeile 4):

```
<s xmlns="http://www.xstandoff.net/phrase">
   <np>
      <pron>This</pron>
   </np><vp>
      <v>is</v>
      <np>
         <det>a</det>
         <n>sentence</n>
      </np>
   </vp>.</s>
```

Dieses ist in den Einstellungen lax und middle der Primärdatenidentitätsprüfung durchaus erlaubt, kann aber in der aktuellen Version des Stylesheets inline2XSF.xsl zu unerwünschten Segmentierungen führen. So würden in der Standardeinstellung dem ersten np-Element die Segmentgrenzen 0...5 zugewiesen, wünschenswert wäre aber 0...4, da die Annotation der Nominalphrase eigentlich nur die Zeichenkette „This“ umfassen sollte. Da das Stylesheet versucht, die wahrscheinlichste Position des Leerzeichens aus dem Primärdatum („This is a sentence.“, Position 4...5) anhand von tatsächlich vorhandenen Zeichen in der XML-Eingabe zu extrahieren, ermittelt es dementsprechend eine Positionierung zwischen den beiden Tags </pron> und </np>. Tritt dieser Fall ein, gibt das Stylesheet während der Transformation folgende Warnung aus:

```
*** WARNING: whitespace from primary data not present in XML!
*** missing: '&amp;#32;' at position: [4]
```

Befänden sich zwischen den Tags </pron> und <vp> gar keine Leerzeichen, würde eine andere Strategie greifen, welche die Position des Leerzeichens außerhalb der Grenzen derjenigen Elemente annimmt, die den vorhergehenden Text („This“) und den nachfolgenden Text („is a sentence“) enthalten. Demzufolge würden in diesem Fall die Segmentgrenzen für das np-Element in folgendem Ausschnitt korrekt bestimmt, da ein virtuelles Leerzeichen zwischen </np> und <vp> angenommen werden könnte.

```
<s>
   <np><pron>This</pron></np><vp>
```

```
3          <v>is</v>
4             <!-- ... -->
5       </vp>.</s>
```

Diese Strategie kann durch den Benutzer des Stylesheets auch manuell festgelegt werden, indem der Stylesheet-Parameter heuristic-ws-search auf den Wert true() gesetzt wird. Da hierdurch jedoch kein Abgleich tatsächlich vorhandener Whitespaces erfolgt, sollten die Vor- und Nachteile dieses Vorgehens sorgfältig abgewägt werden.

### 4.2.2 Zusammenführung von Instanzen: mergeXSF.xsl

Das Stylesheet mergeXSF.xsl übernimmt die Aufgabe der Zusammenführung zweier XStandoff-Instanzen, die auf dem selben Primärdatum beruhen. Dabei wird während des Aufrufs der Transformation die erste XStandoff-Datei als Eingabedatei der XSLT-Transformation spezifiziert und die zweite mit Hilfe des Stylesheet-Parameters merge-with referenziert. Dieses könnte für zwei Dateien (Phrasenstrukturannotation und Silbenannotation in separaten XStandoff-Instanzen) z.B. folgendermaßen aussehen:

```
saxon -o phr-syll-xsf.xml phr-xsf.xml mergeXSF.xsl merge-with=syll-xsf.xml
```

Die Zusammenführung der beiden Dateien phr-xsf.xml (während der Transformation in der globalen Variable $XSF1 gespeichert) und syll-xsf.xml ($XSF2) in die gemeinsame Instanz phr-syll-xsf.xml kann in zwei Teilaufgaben geteilt werden: (1) die Zusammenführung der Segmentlisten aus den beiden XStandoff-Instanzen, und (2) die Kopie der Annotationslayer inkl. einer Anpassung der Segment-Referenzen.

Zusammenführung von Segmentlisten

Da die beiden vorhandenen Segmentlisten (<xsf:segmentation>) aus $XSF1 und $XSF2 höchstwahrscheinlich unterschiedliche IDs für Segmente mit gleichen Start- und Endpositionen beinhalten, können die xsf:segment-Elemente aus den Dateien nicht einfach in die zu erstellende gemeinsame XStandoff-Instanz kopiert werden. Vielmehr müssen sie aktualisiert werden, um mehrfach verwendete ID-Werte auszuschließen. Zwar kann die Aktualisierung der Liste von $XSF1 durch Angabe des Stylesheet-Parameters keep-segments=1 verhindert werden: für diesen Fall werden nur die IDs aus $XSF2 mit neuen Werten belegt. Standardmäßig erfolgt jedoch eine Reorganisation der Segmente in der gemeinsamen Liste. Dieses kann anhand folgender Tabelle verdeutlicht werden:

| neue @xml:id | $XSF1 - @xml:id | $XSF2 - @xml:id | @start | @end |
|---|---|---|---|---|
| seg1 | seg1 | seg1 | 0 | 19 |
| seg2 | seg2 | seg2 | 0 | 4 |
| seg3 | seg3 | - | 5 | 18 |
| seg4 | seg4 | seg3 | 5 | 7 |
| seg5 | seg5 | - | 8 | 18 |
| seg6 | seg6 | seg4 | 8 | 9 |
| seg7 | seg7 | - | 10 | 18 |
| seg8 | - | seg5 | 10 | 13 |
| seg9 | - | seg6 | 13 | 18 |

*Tabelle 1: mergeXSF - Segmentindex*

Die Zusammenführung bzw. Neuorganisation der beiden Segmentlisten aus $XSF1 und $XSF2 besteht aus der Erstellung jeweils eines neuen Segments für alle unterschiedlichen Kombinationen von Start- und Endpositionen, sowie der Sortierung dieser neuen Segmente auf Basis ihrer Positionsangaben und der Vergabe neuer fortlaufender ID-Werte (seg1...n).

Der in Tabelle 1 dargestellte Index, welcher für die Transformation in einer globalen Variable bestehend aus Elementen der Form  vorliegt, kann genutzt werden, um einerseits eine neue Liste von Segmenten auszugeben (in diesem Fall werden die Attribute @XSF1ID und @XSF2ID weggelassen) und um andererseits als Referenz für die Kopie der Annotationslayer (<xsf:layer>) aus $XSF1 und $XSF2 in die resultierende gemeinsame XStandoff-Instanz zu dienen.

Die intensive Nutzung von Keys in XSLT (<xsl:key> und key(); vgl. Kay, 2008: 376ff und 812ff) ermöglicht in diesem Zusammenhang eine effiziente Verarbeitung bzw. Indizierung vorhandener Segmente und die schnelle Zugänglichkeit der neu erstellten Segmente, z.B. bei der Kopie der einzelnen Layer. Vergleichstests zwischen einer älteren Stylesheet-Version (old-mergeXSF.xsl; siehe CD im Ordner „key-test\xsl“), welche Segmente mit Hilfe von Vergleichsoperatoren in XPath-Ausdrücken paarweise (in einer doppelten for-each-Schleife) miteinander vergleicht, und der aktuellen Version (mergeXSF.xsl) zeigen, dass die Transformationsgeschwindigkeit durch Keys erheblich gesteigert werden kann. So hängt die Dauer der XSLT-Transformation mit der aktuellen Stylesheet-Version etwas weniger als linear von der Anzahl der existierenden Segmente ab (vgl. Anhang B, S. 79).

Kopie der Annotationslayer/Aktualisierung der Segment-Referenzen

Nachdem, wie im vorherigen Abschnitt dargestellt wurde, die Segmentlisten der beiden zusammenzuführenden XStandoff-Instanzen miteinander kombiniert wurden, kann nun die relativ simple Kopie der einzelnen Annotationslayer erfolgen. Zu diesem Zweck werden die Elemente unterhalb von <xsf:layer> rekursiv durchlaufen und kopiert, während ihre Segmentreferenzen @xsf:segment mit den neu ermittelten Werten belegt werden (siehe erste Spalte der Tabelle 1).

Eine leichte Modifikation des Vorgehens bei der Kopie der Layer erfolgt durch die Angabe des Stylesheet-Parameters all-layer=1. Hierdurch wird ein neuer Annota-

tionslayer angelegt, der diejenigen Elemente enthält, die in allen der bereits vorhandenen Layern mit gleichen Positionsangaben und gleichem lokalen Namen (XPath-Funktion: local-name()) existieren (vgl. Abschnitt „Der all-Layer“, S. 29). In diesem Fall werden die entsprechenden Elemente bei der Kopie der bereits vorhandenen Annotationslayer übergangen, sodass sie tatsächlich nur noch im all-Layer zu finden sind.

Die technische Umsetzung der Erstellung des all-Layers kann theoretisch auf Basis eines beliebigen Layers aus $XSF1 oder $XSF2 erfolgen (mergeXSF.xsl verwendet den ersten, der in $XSF1 zu finden ist), da alle Layer Instanzen derjenigen Elemente enthalten, welche im all-Layer gespeichert werden sollen. Durch die gleiche Strategie wie bei der Kopie der übrigen „normalen“ Layer, also die rekursive Kopie von Elementen, die die Bedingung für die Zugehörigkeit zum all-Layer erfüllen, wird auf diese Weise der zusätzliche Annotationslayer erstellt.

### 4.2.3 Extraktion/Löschung von Leveln oder Layern: extractXSFcontent.xsl

Das nachfolgend vorgestellte Stylesheet extractXSFcontent.xsl ist eine Nachfolgeversion des Stylesheets removeXSFcontent.xsl, und dient der Bearbeitung von XStandoff-Instanzen durch Extraktion bzw. Löschung einzelner Level oder Layer. Die Namensänderung bzw. Überarbeitung wurde vorgenommen, um die Möglichkeit der Extraktion, die bereits in der Vorgängerversion implizit (durch optionale Ausgabe gelöschter Inhalte in separate XStandoff-Instanzen) vorgesehen war, deutlich hervorzuheben und konzeptuell von der Löschung abzuheben.

Die gewünschte Operation des Stylesheets hängt in der vorliegenden Version von der Wahl eines der exklusiv zu verwendenden Stylesheet-Parameter remove-ID und extract-ID ab. Je nachdem welcher Parameter spezifiziert wird, werden entweder Inhalte aus der XStandoff-Instanz gelöscht oder extrahiert. Ein Beispielaufruf der

Transformation, um bspw. das xsf:level-Element mit der ID (@xml:id) „phr-role-level2“ aus der XStandoff-Instanz auf Seite 37 zu entfernen, sieht wie folgt aus:

```
saxon -o phr-xsf.xml phr-role-xsf.xml extractXSFcontent.xsl
remove-ID=phr-role-level2
```

Die beiden vorgesehenen Operationen, Extraktion und Löschung, werden im Folgenden kurz erläutert.

Löschung von Inhalten

Durch die Angabe einer ID (Wert des Attributs @xml:id) eines in der Eingabedatei vorhandenen xsf:level- oder xsf:layer-Elements kann eine XStandoff-Instanz erstellt werden, in der das entsprechend identifizierte Element entfernt wurde. Sollte kein Element oder gar mehrere mit dieser ID gefunden werden, wird eine Fehlermeldung ausgegeben.

Zwar wäre die Entfernung einzelner Level oder Layer aus einer XStandoff-Instanz theoretisch auch manuell möglich, jedoch ist z.B. die Aktualisierung von Segment-Informationen bzw. -Referenzen eine sehr komplexe Aufgabe, die bei manueller Bearbeitung die Gefahr resultierender Inkonsistenzen birgt (vgl. mergeXSF.xsl, Abschnitt 4.2.2, S. 46). Dementsprechend besteht die Löschung eines <xsf:level> oder <xsf:layer> nicht nur aus dem Entfernen dieses Elements selbst und seiner Kindelemente, sondern ggfs. auch aus dem Entfernen nicht mehr verwendeter xsf:segment-Elemente aus der Segmentliste, sowie einer neuen fortlaufenden Nummerierung der Segment-IDs () und der Anpassung von alten Segment-Referenzen (@xsf:segment) für die Elemente der verbleibenden Layer. Das Vorgehen des Stylesheets extractXSFcontent.xsl erfolgt analog zu dem für das Stylesheet mergeXSF.xsl beschriebenen (vgl. S, 46ff), d.h. für die verbleibenden Layer wird eine neue Liste mit verwendeten Segmenten erstellt.

Die Durchführung einer Aktualisierung der Segmentinformationen wird bei der Transformation standardmäßig ausgeführt, jedoch steht es dem Benutzer, genau wie

bei der Anwendung des Stylesheets mergeXSF.xsl, generell frei, die alten Segmentinformationen in die resultierende XStandoff-Instanz übernehmen zu lassen (Stylesheet-Parameter $keep-segments=1).

Extraktion von Inhalten

Die Extraktion von xsf:level- oder xsf:layer-Elementen erfolgt nach dem gleichen Prinzip wie die Löschung, allerdings wird nur das durch den Stylesheet-Parameter $extract-ID identifizierte Element in die resultierende XStandoff-Instanz übernommen. Dementsprechend kann die Extraktion eines Levels oder Layers auch als die Löschung aller anderen angesehen werden. Spätestens an dieser Stelle sollte deutlich werden, warum die beiden Operationen Löschung und Extraktion durch ein gemeinsames XSLT-Stylesheet realisiert werden. Die einzelnen Schritte decken sich mit den für die Löschung beschriebenen. Allerdings wird, wie oben erwähnt, anstatt des Parameters remove-ID zur Extraktion eines Levels oder Layers extract-ID verwendet.

### 4.2.4 Erstellung von Inline-XStandoff: XSF2inline.xsl

Das Konzept der Unifikation von XML-Dokumenten ist von Witt et al. (2005) ausführlich dargestellt worden. Große Teile des Ansatzes spiegeln sich in der Realisierung des Stylesheets XSF2inline.xsl wider, welches v.a. zu Demonstrationszwecken die verschiedenen Layer einer XStandoff-Instanz in eine gemeinsame Inline-Annotation überführen kann.

Als Ausgangsannotation kann folgende XStandoff-Instanz herangezogen werden:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <xsf:corpusData xsfVersion="1.1"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xmlns:xsf="http://www.xstandoff.net/2009/xstandoff/1.1"
5         xml:id="sun-morph-sun-syll" xsi:schemaLocation="
          http://www.xstandoff.net/2009/xstandoff/1.1
```

```
         http://www.xstandoff.net/2009/xstandoff/1.1/xsf.xsd">
6      <xsf:primaryData start="0" end="24">
7         <xsf:primaryDataRef uri="sun.txt"/>
8      </xsf:primaryData>
9      <xsf:segmentation>
10        <xsf:segment xml:id="seg1" start="0" end="24"/>
11        <xsf:segment xml:id="seg2" start="0" end="3"/>
12        <xsf:segment xml:id="seg3" start="4" end="7"/>
13        <xsf:segment xml:id="seg4" start="8" end="14"/>
14        <xsf:segment xml:id="seg5" start="8" end="13"/>
15        <xsf:segment xml:id="seg6" start="13" end="14"/>
16        <xsf:segment xml:id="seg7" start="15" end="21"/>
17        <xsf:segment xml:id="seg8" start="15" end="20"/>
18        <xsf:segment xml:id="seg9" start="20" end="23"/>
19        <xsf:segment xml:id="seg10" start="21" end="23"/>
20     </xsf:segmentation>
21     <xsf:annotation>
22        <xsf:level xml:id="sun-morph-level1">
23           <xsf:layer priority="0"
24                 xmlns:morph="http://www.xstandoff.net/morph">
25              <morph:morphemes xsf:segment="seg1">
26                 <morph:m xsf:segment="seg2"/>
27                 <morph:m xsf:segment="seg3"/>
28                 <morph:m xsf:segment="seg5"/>
29                 <morph:m xsf:segment="seg6"/>
30                 <morph:m xsf:segment="seg7"/>
31                 <morph:m xsf:segment="seg10"/>
32              </morph:morphemes>
33           </xsf:layer>
34        </xsf:level>
35        <xsf:level xml:id="sun-syll-level1">
36           <xsf:layer priority="0"
37                 xmlns:syll="http://www.xstandoff.net/syll" >
38              <syll:syllables xsf:segment="seg1">
39                 <syll:s xsf:segment="seg2"/>
40                 <syll:s xsf:segment="seg3"/>
41                 <syll:s xsf:segment="seg4"/>
42                 <syll:s xsf:segment="seg8"/>
43                 <syll:s xsf:segment="seg9"/>
44              </syll:syllables>
45           </xsf:layer>
46        </xsf:level>
47     </xsf:annotation>
48  </xsf:corpusData>
```

Diese XStandoff-Instanz enthält zwei Annotationslayer: Morphem- und Silbenannotationen für das Primärdatum „The sun shines brighter.“. Diese wurden zunächst einzeln nach XStandoff überführt (inline2XSF.xsl) und danach in eine gemeinsame Instanz transformiert (mergeXSF.xsl). Für die Überführung in eine Inline-Variante müssen zwei grundlegende Konfliktszenarien für Inline-XML-Repräsentationen multipler Hierarchien bewältigt werden (vgl. Abschnitt 2.2 „Probleme der Repräsentation“, S. 5): Identitätskonflikte und Überlappungen von Elementen aus

verschiedenen Hierarchien (xsf:layer-Elementen). Diskontinuierliche Einheiten werden von dem Stylesheet XSF2inline.xsl derzeit nicht unterstützt.

Identitätskonflikte

Das erste Problem besteht in der Bestimmung der Einbettungsreihenfolge von Elementen mit gleichen Segmentinformationen. Dieses Problem betrifft Elemente aus verschiedenen Layern, deren Start- und/oder Endpositionen übereinstimmen (vgl. die Abbildungen zu Elementrelationen, S. 7 und 8, Relationen 2, 4, 6, 7, 8, 10, 12, 14 und 15).

Die Einbettungsentscheidung für Elementrelationen, bei denen entweder nur die Startpositionen oder nur die Endpositionen übereinstimmen, ist relativ einfach zu bewerkstelligen, indem man dasjenige Element als oberes (das Elternelement) realisiert, welches mehr Textinhalt hat (vgl. Witt et al., 2005: 110f.). Auf diese Weise verhindert man die Entstehung von „spurious overlaps“ (Sperberg-McQueen & Huitfeld, 2004: 158; vgl. Abschnitt 2.2, S. 8), also Überlappungen, die ohne Änderung von Positionsinformationen in einem hierarchischen Modell aufgelöst werden können (vgl. ebd.). Die Überlappung in folgender Pseudo-XML-Notation

```
1   <a><b>John likes</a> Mary</b>
```

wird dementsprechend zu der XML-konformen Annotation

```
1   <b><a>John likes</a> Mary</b>
```

Demgegenüber bedarf die Entscheidung bei gleichen Start- und Endpositionen einer komplexeren Strategie, denn es lässt sich z.B. nicht ohne weiteres feststellen, welche der folgenden Möglichkeiten für die Einbettung der Elemente mit der Segmentreferenz @xsf:segment=“seg2“ aus der obigen XStandoff-Instanz vorzuziehen ist:

```
1   <morph:m><syll:s>This</syll:s></morph:m>
```

```
1   <syll:s><morph:m>This</morph:m></syll:s>
```

Der Unifikationsalgorithmus von Witt et al. (2005) bietet für den Fall eines Identitätskonflikts drei verschiedene Lösungsansätze: (1) die externe Spezifikation einer generellen Priorität der beiden Layer, als einbettend oder eingebettet realisiert zu werden, (2) nur eines der beiden Elemente wird in das Resultat aufgenommen, und (3) eines der Elemente wird berücksichtigt und die Attribute des anderen Elements werden dem aufgenommenen hinzugefügt (vgl. Witt et al., 2005: 112f.). Im Rahmen des Stylesheets XSF2inline.xsl wurde die erste der genannten Möglichkeiten implementiert. Sie bedient sich der Angaben des Attributs @priority, welches für xsf:layer-Elemente spezifiziert werden kann (Default-Wert: 0, je niedriger der Wert, desto tiefer werden Elemente des Layers eingebettet). Die Nutzung der Prioritätsangabe ist zwar nicht standardmäßig vorgesehen, kann jedoch durch Angabe des Stylesheet-Parameters sort-by=priority festgelegt werden.

Zusätzlich zur Möglichkeit einer manuellen Prioritätsangabe für Layer wurde zur Überführung einer Standard-XStandoff-Instanz in die Inline-Variante ein alternatives Vorgehen implementiert, welches auf einer statistischen Methode beruht (Standard-Realisierung des Stylesheet-Parameters: sort-by=measure). Zu diesem Zweck wurde eine Funktion (doc:inclusion-measures()) definiert, die vor Beginn des Aufbaus der Inline-Repräsentation für alle möglichen Kombinationen von zwei Elementtypen aus den verschiedenen Annotationslayern die statistische Verteilung von Inklusionsrelationen anhand ihrer Start- und Endpositionen ermittelt. Auf diese Weise kann bspw. für die beiden denkbaren Inklusionsrelationen von <morph:m> und <syll:s> (s.o.) folgendes Bild gezeichnet werden:

```
1   <measure parent="morph:m" child="syll:s" value="0.5"/>
2   <measure parent="syll:s" child="morph:m" value="0.8"/>
```

Diese Angaben beinhalten die Information, dass für 50% aller morph:m-Elemente (@value="0.5") syll:s-Elemente existieren, die identische oder einbettende Positionsinformationen (vgl. Abbildungen zu Elementrelationen, S. 7, Relationen: 6, 7, 9, 10, 14, 15) haben. Hingegen sind für 80% aller syll:s-Elemente morph:m-Elemente zu finden, die theoretisch als ihre Kindelemente serialisiert werden könnten. Daraus lässt sich folgern, dass aus statistischer Sicht <morph:m> in <syll:s> eingebettet werden sollte.

Für den Fall, dass weder durch die statistische Methode noch durch die Auswertung der priority-Attribute eine Präferenz zur Einbettung inferiert werden kann, wird dennoch eine Einbettungsentscheidung getroffen, indem die Reihenfolge der Layer, in denen die betroffenen Elemente vorkommen, als Entscheidungshilfe genutzt wird: Elemente aus in der Ausgangs-XStandoff-Instanz weiter oben stehenden Layern betten weiter unten stehende ein.

Klassische Überlappungen

Das zweite Problem bei der Erstellung einer XStandoff-Inline-Annotation bezieht sich auf die Behandlung von klassischen Überlappungen (Abbildung 1, S. 7, Relationen 3 und 11). Hier verwendet das Stylesheet XSF2inline.xsl in Analogie zu Witt et al. (2005: 111) Milestones (vgl. Abschnitt 2.3.1, S. 10). Durch diese wird die Darstellung einer XStandoff-Instanz in Form einer XML-Inline-Repräsentation ermöglicht.

Die Basis der Serialisierung der Inline-XStandoff-Variante bildet in diesem Zusammenhang eine temporäre, modifizierte Form der Segmentliste, die auf der einen Seite um Informationen zu den textuellen Inhalten (<text>, siehe Zeile 3 und 5, Rest ausgelassen) erweitert wurde. Andererseits wurden diejenigen Segmente, die Teil einer klassischen Überlappung sind, durch Milestones ersetzt (siehe Zeile 13 und 18):

```
1   <xsf:segment xml:id="seg1" start="0" end="24"/>
2   <xsf:segment xml:id="seg2" start="0" end="3"/>
```

```
3   <text xml:id="text0" start="0" end="1"/>
4   <!-- Textinhalt -->
5   <text xml:id="text3" start="3" end="4"/>
6   <xsf:segment xml:id="seg3" start="4" end="7"/>
7   <!-- ... -->
8   <xsf:segment xml:id="seg4" start="8" end="14"/>
9   <xsf:segment xml:id="seg5" start="8" end="13"/>
10  <!-- Textinhalt -->
11  <xsf:segment xml:id="seg6" start="13" end="14"/>
12  <!-- ... -->
13  <milestone xml:id="seg7~1" start="15" end="15" x="26"
              xsf:charPos="15" xsf:type="start"/>
14  <xsf:segment xml:id="seg8" start="15" end="20"/>
15  <!-- ... -->
16  <xsf:segment xml:id="seg9" start="20" end="23"/>
17  <!-- ... -->
18  <milestone xml:id="seg7~2" start="21" end="21" x="26"
              xsf:charPos="21" xsf:type="end"/>
19  <xsf:segment xml:id="seg10" start="21" end="23"/>
20  <!-- ... -->
```

Da auf diese Weise Überlappungskonflikte durch Milestones aufgelöst werden[21] und Identitätskonflikte, wie oben beschrieben, durch verschiedene Möglichkeiten (statistisch oder durch @priority) behandelt werden, kann abschließend durch Iteration der Segmente und durch den rekursiven Aufbau der XML-Struktur eine Inline-Repräsentation der XStandoff-Instanz erstellt werden:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <xsf:inline
          xmlns:xsf="http://www.xstandoff.net/2009/xstandoff/1.1"
          xmlns:morph="http://www.xstandoff.net/morph"
          xmlns:syll="http://www.xstandoff.net/syll">
3      <morph:morphemes xsf:segment="seg1">
4         <syll:syllables xsf:segment="seg1">
5            <syll:s xsf:segment="seg2">
6               <morph:m xsf:segment="seg2">The</morph:m>
7            </syll:s>
8            <syll:s xsf:segment="seg3">
9               <morph:m xsf:segment="seg3">sun</morph:m>
10           </syll:s>
11           <syll:s xsf:segment="seg4">
12              <morph:m xsf:segment="seg5">shine</morph:m>
13              <morph:m xsf:segment="seg6">s</morph:m>
14           </syll:s>
15           <xsf:milestone xsf:unit="morph:m" xsf:n="1"
                            xsf:segment="seg7~1" xsf:charPos="15"
                            xsf:type="start"/>
16           <syll:s xsf:segment="seg8">brigh</syll:s>
17           <syll:s xsf:segment="seg9">t
18              <xsf:milestone xsf:unit="morph:m" xsf:n="2"
                               xsf:segment="seg7~2" xsf:charPos="21"
                               xsf:type="end"/>
19              <morph:m xsf:segment="seg10">er</morph:m>
20           </syll:s>.
```

21 Es wird jeweils das erste Element, das Teil einer Überlappung ist, durch Milestones ersetzt

```
21        </syll:syllables>
22      </morph:morphemes>
23  </xsf:inline>
```

Besondere Beachtung ist hierbei den milestone-Elementen in den Zeilen 15 und 18, sowie der Einbettung von <morph:m> in <syll:s>, welche statistisch ermittelt wurde, zu schenken. Die Tatsache, dass <syll:syllables> in <morph:morphemes> eingebettet wurde, ist dem Umstand geschuldet, dass dieser Identitätskonflikt weder statistisch noch durch die angegebenen priority-Attribute der <xsf:layer> (jeweils Wert 0) aufgelöst werden konnte und somit die Reihenfolge der Layer in der XStandoff-Instanz entscheidend war (s.o.).

Weitere Beispiele für Inline-XStandoff-Repräsentationen sind auf der beiliegenden CD im Ordner „inline“ oder auf http://www.jettka.de/xsf zu finden

### 4.2.5 Visualisierung von XStandoff-Instanzen: XSF2SVG.xsl

Aufgrund der Komplexität des nachfolgenden Ansatzes, möchte ich in diesem Abschnitt weniger auf die technische Umsetzung des XSLT-Stylesheets XSF2SVG.xsl eingehen, sondern vielmehr die Grundkonzeption der Visualisierung multipler Hierarchien im Allgemeinen und derjenigen von XStandoff-Instanzen im Speziellen thematisieren.

Bei der Visualisierung von XStandoff-Instanzen kommt ein relativ einfaches Prinzip der Darstellung von Baumstrukturen zur Anwendung. Dieses ist möglich, da die einzelnen Annotationslayer in XStandoff keine Überlappungen enthalten können, und somit separat als Bäume betrachtet werden können (multi-rooted trees). Eine Darstellungsmethode für getrennte Annotationsschichten, die große Ähnlichkeit mit der im Rahmen dieser Arbeit verfolgten Strategie aufweist, ist z.B. bei Witt (2005) zu finden:


*Abbildung 5: Visualisierung von Annotationsschichten (Witt, 2005: 76)*

Hier sind zwei farbig voneinander abgehobene Annotationsschichten zu sehen, die sich auf ein gemeinsames Primärdatum (hier einen Text) beziehen. Durch die Länge der einzelnen horizontal angeordneten Segmente, welche die Annotationselemente repräsentieren, wird der jeweils umspannte textuelle Inhalt, welcher unterhalb der Balken dargestellt wird, verdeutlicht. Die oben aufgeführte Darstellungsform hat allerdings gewisse Nachteile: so hängt die Breite, und somit auch die Übersichtlichkeit, entscheidend von der Länge des zugrunde liegenden Texts ab. Darüber hinaus können evtl. im Primärdatum vorhandene Zeilenumbrüche nicht berücksichtigt werden.

Durch eine vertikale Anordnung der Annotationsschichten könnten diese Nachteile umgangen werden. Das Konzept kann anhand der folgenden Annotationen für Silben und Morpheme in dem Satz „The sun shines brighter.“ verdeutlicht werden (die korrespondierende XStandoff-Instanz wurde auf Seite 51 eingeführt):

```
1   <morphemes>
2      <morpheme>The</morpheme>
3      <morpheme>sun</morpheme>
4      <morpheme>shine</morpheme>
5      <morpheme>s</morpheme>
6      <morpheme>bright</morpheme>
7      <morpheme>er</morpheme>.
8   </morphemes>
```

```
<syllables>
   <syllable>The</syllable>
   <syllable>sun</syllable>
   <syllable>shines</syllable>
   <syllable>brigh</syllable>
   <syllable>ter</syllable>.
</syllables>
```

Da eine Überlappung der Annotationen an der Stelle des Wortes „brighter“ existiert, können diese nicht ohne Weiteres in eine gemeinsame Baumstruktur integriert werden. Nach der Darstellungsmethode in Witt (2005: 76) könnte man die Annotationen folgendermaßen visualisieren (um die vorhandenen Baumstrukturen zu verdeutlichen, wurden sie zusätzlich in Knoten- und Kantendarstellung eingefügt):

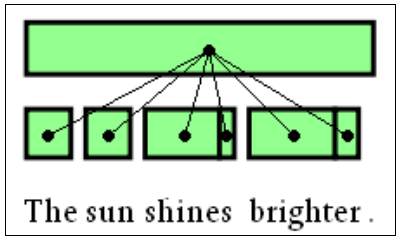



*Abbildung 6: Horizontale Darstellung, Morphemannotation*

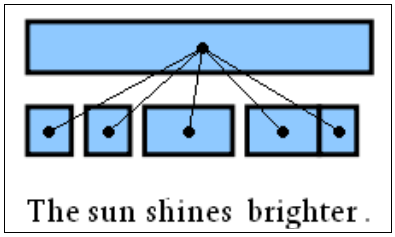



*Abbildung 7: Horizontale Darstellung, Silbenannotation*

Um die o.g. Nachteile der horizontalen Anordnung der Elementrepräsentationen zu vermeiden, könnte man die obigen Darstellungen um 90° nach rechts drehen, horizontal spiegeln, sowie die Laufrichtung des Textes anpassen und erhielte folgende Ansicht:

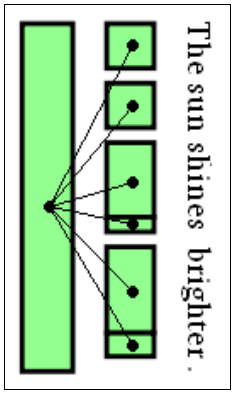



*Abbildung 8: Vertikale Darstellung, Morphemannotation*

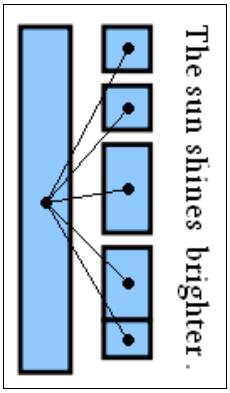



*Abbildung 9: Vertikale Darstellung, Silbenannotation*

Um nun die Lesbarkeit des Textes sowie die Berücksichtigung von Zeilenumbrüchen im Text zu gewährleisten, bedarf es weiterer Schritte wie sie z.B. Piez (2010) realisiert hat. Die graphische Darstellung von XStandoff-Instanzen durch das XSLT-Stylesheet `XSF2SVG.xsl` ist maßgeblich von dem nachfolgend vorgestellten Ansatz von Piez (2010) inspiriert. Dieser hat auf Basis von LMNL-Markup (vgl. Abschnitt 2.3.3, S. 17) eine Visualisierung multipler Hierarchien mit Hilfe von interaktivem SVG (Scalable Vector Graphics)[22] implementiert:

22 vgl. http://www.w3.org/TR/SVG11/

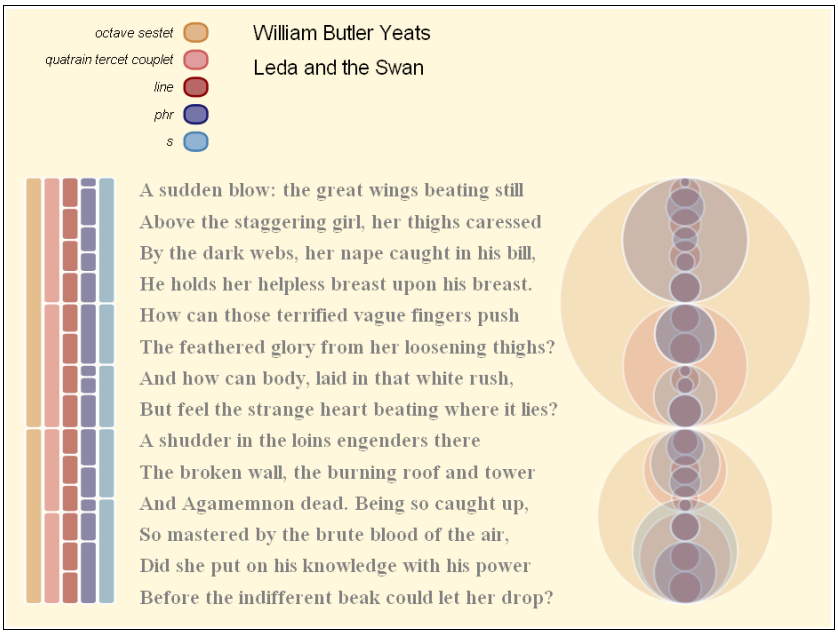



*Abbildung 10: Markup-Visualisierung durch Piez (2010)*

Die vorhandenen Annotationsschichten und beinhalteten Elementtypen, welche oben links in der Grafik aufgelistet sind, und per Mausklick einzeln ausgeblendet werden können, werden durch zwei verschiedene Darstellungen, als Balken (links) einerseits und als Kreise (rechts) andererseits, mit dem annotierten Text (in der Mitte) in Beziehung gesetzt. Durch interaktive Mouseover-Effekte wird es dem Benutzer ermöglicht, korrespondierende Textteile und Annotationen hervorzuheben (vgl. http://www.piez.org/wendell/papers/dh2010/clix-sonnets/ledaandswan-map.svg). Überlappungen der verschiedenen Annotationsschichten sind in dieser Darstellungsform anhand der Balken (z.B. roter Balken/erstes Segment und lila Balken/ zweites Segment) sowie anhand sich überschneidender Kreise nachzuvollziehen.

Obwohl die obige Art der Visualisierung, wie Piez (2010) betont, v.a. Demonstrationszwecken dient, sollten die Nachteile, welche sie mit sich bringt, kurz benannt werden, da sie direkte Auswirkungen auf die Implementation der Visualisierung von XStandoff-Instanzen hatten:

1. die Annotationsschichten in Piez' Beispielen enthalten nur Elemente, die ausreichend lange Textpassagen auszeichnen. Auf diese Weise entstehen

keine Probleme bei der Darstellung der einzelnen korrespondierenden Balkensegmente. Wollte man bspw. eine Annotationsschicht für die einzelnen Token aus dem Text in obige Grafik integrieren, würden die Balkensegmente und Kreise so klein, dass sie nicht mehr sinnvoll dargestellt werden könnten.

2. die Kreisdarstellung ist nur praktikabel, solange keine Annotationen vorhanden sind, die sehr große Textteile umspannen, da ansonsten der Umfang der Kreise zu groß würde.

3. da die einzelnen Annotationsschichten jeweils den kompletten Text lückenlos abdecken, entsteht der Eindruck, dass die Visualisierungsform sehr übersichtlich ist. Dieses ändert sich jedoch sobald Annotationsschichten eingeführt werden, deren Elemente bestimmte Teile des Textes nicht erfassen. Dieser Umstand wird im späteren Verlauf der Erläuterungen zum Visualisierungsansatz in dieser Arbeit deutlicher.

Vor dem Hintergrund dieser Einschränkungen, jedoch auch mit Blick auf die Stärken des Ansatzes von Piez (2010), wurden alle in Abbildung 10 gezeigten Bestandteile, außer der Kreisdarstellung, für die graphische Darstellung von XStandoff-Instanzen mit Hilfe von SVG implementiert und um zusätzliche Interaktionsmöglichkeiten für den Betrachter erweitert. Der Aufruf der XSLT-Transformation einer XStandoff-Instanz in die SVG-Visualisierung mit `XSF2SVG.xsl` sieht dabei folgendermaßen aus (zum Zweck der Vergleichbarkeit wurde die Annotation aus Piez' Beispiel zunächst nach XStandoff überführt[23] und dann folgendermaßen in SVG transformiert):

```
saxon -o ledaandswan-xsf.svg ledaandswan-xsf.xml XSF2SVG.xsl
```

Das Erscheinungsbild einer mit Hilfe von `XSF2SVG.xsl` visualisierten XStandoff-Instanz sieht folgendermaßen aus:

23 Siehe beiliegende CD „xsf\ledaandswan-xsf.xml“ oder http://www.jettka.de/xsf/examples/xsf/ledaandswan-xsf.xml

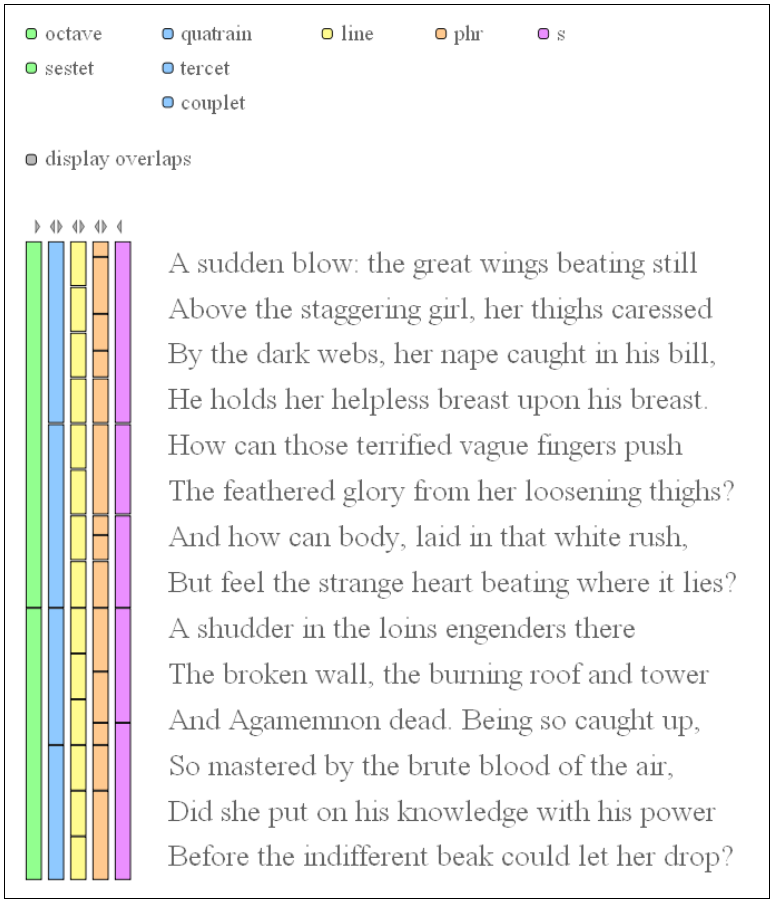



*Abbildung 11: Visualisierung von XStandoff durch XSF2SVG.xsl*

Die einzelnen Hierarchien (xsf:layer-Elemente) werden in unterschiedlichen Farben dargestellt. Mouseover-Effekte sorgen wie bei Piez (2010) dafür, dass korrespondierende Elemente und Primärdaten hervorgehoben werden können (vgl. Anhang C, S. 82, Abbildungen 16 und 17). Zusätzlich besteht die Möglichkeit, klassische Überlappungen durch Klick auf die Schaltfläche „display overlaps“ durch Farbänderung der entsprechenden Segmente anzeigen zu lassen (vgl. Anhang C, S. 82, Abbildung 18). Des weiteren kann die Anordnung der Hierarchien verändert bzw. einzelne Elementtypen ausgeblendet werden (vgl. Anhang C, S. 82, Abbildung 19)[24].

Neben den offensichtlichen Vorzügen, die diese Art der Darstellung von XStandoff-Instanzen (und damit multiplen Hierarchien) bietet, ist der Ansatz nicht mehr als

24 Die Funktionalität des Beispiels kann mit Hilfe der auf der beiliegenden CD mitgelieferten Datei „svg\ledaandswan-xsf.svg“ oder auch auf http://www.jettka.de/xsf/examples/svg/ledaandswan-xsf.svg getestet werden. Weitere Beispiele sind auf der CD im Ordner „svg“ oder auf http://www.jettka.de/xsf/ zu finden

ein Orientierungspunkt auf dem Weg zu ausgereifteren Visualisierungsmethoden. Neben dem Problem, dass leere Elemente z.Z. nicht berücksichtigt werden, kann es schnell zu unübersichtlichen Ergebnissen für Hierarchien mit Elementen, die sich auf sehr kleine Primärdatenausschnitte beziehen, kommen (z.B. Token, Morpheme, Silben). Für diese Fälle muss der Text in der Darstellung mit zusätzlichen Zeilenumbrüchen versehen werden, damit auch für wenige Positionen umfassende Segmente sichtbare Balkensegmente erstellt werden können. Dieses verdeutlicht der folgende Ausschnitt, welcher eine XStandoff-Instanz (buergschaft-xsf.xml) visualisiert, in der u.a. Annotationen für Silben (syll-Elemente im ersten Layer) und Wörter (w-Elemente im zweiten Layer) existieren[25]:

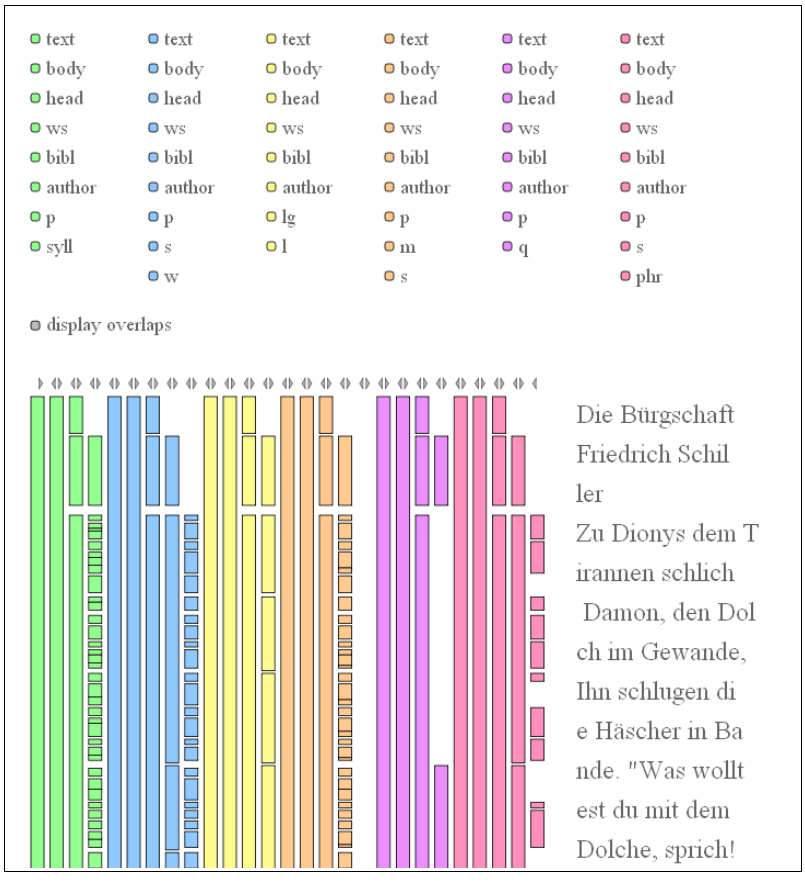



*Abbildung 12: Visualisierung von Hierarchien inkl. kleiner Segmente*

25 Die Übersichtlichkeit der Visualisierung könnte durch die Berücksichtigung des all-Layers (vgl. Abschnitt 3.2, S. 29) für die zugrunde liegende XStandoff-Instanz verbessert werden, da auf diese Weise die Elemente <text>, <body>, <head>, <bibl> und <author> aus den einzelnen Layern ausgelagert würden.

Darüber hinaus kann die Interaktivität für große XStandoff-Instanzen aufgrund einer verlangsamten Verarbeitungsgeschwindigkeit der JavaScript-basierten Mouseover-Effekte leiden. Dieses variiert allerdings stark in Abhängigkeit des zur Ansicht der SVG-Datei verwendeten Werkzeugs: die während der Entwicklung verwendeten Browser Google Chrome 9.0.597.98 und Opera 11.01 sind bspw. deutlich schneller in der Bearbeitung der Effekte als der Mozilla Firefox 3.5.16. Die oben aufgeführten sowie einige weitere Beispiele für XStandoff-Visualisierungen mit Hilfe des Stylesheets `XSF2SVG.xsl` sind auf der beiliegenden CD im Ordner „svg“ zu finden.

Aufgrund der genannten Nachteile und des Umstands, dass bestimmte Herausforderungen an die Repräsentation multipler Hierarchien (z.B. diskontinuierliche Einheiten) mit dem vorliegenden Visualisierungsansatz nicht dargestellt werden können, wird deutlich, dass die mit Hilfe von `XSF2SVG.xsl` erstellten SVG-Dateien nicht geeignet sind, multiple Hierarchien in optimaler Weise darzustellen. Aus diesem Grund gibt es bereits Vorüberlegungen zu einer Visualisierungsmethode, die den auf XML basierenden ISO-Standard X3D zur Realisierung dreidimensionaler Grafiken (vgl. u.a. Kloss, 2010; oder auch http://www.web3d.org/x3d/) nutzt, um bspw. mehrere Hierarchien als hintereinander gelegte transparente Graphenstrukturen zu visualisieren. Zwischen bzw. in diesen Hierarchien könnte dann durch interaktive Mouse-Effekte navigiert werden. Dieses sind jedoch lediglich erste Denkansätze auf dem Weg zu einer geeigneten Visualisierungsmethode.

# 5 Zusammenfassung und Ausblick

Die XML-basierte Annotation sprachlicher Daten unterliegt wichtigen hierarchischen Beschränkungen, die sich auf die Repräsentation multipler Hierarchien auswirken (vgl. Kapitel 2). So können Annotationen verschiedener Beschreibungsebenen u.U. nicht mit Standard-Inline-XML umgesetzt werden, wenn bspw. überlappende Annotationen oder diskontinuierliche Einheiten erfasst werden sollen. In Abschnitt 2.3 (S. 10) wurden verschiedene Formate vorgestellt, die unterschiedliche Strategien zur Repräsentation multipler Hierarchien verfolgen und mit jeweils eigenen spezifischen Vor- und Nachteilen verbunden sind.

Mit Bezug auf die vorhandenen Formate wurde in Kapitel 3 das ausdrucksmächtige Repräsentationsformat für multiple Hierarchien XStandoff dargestellt, welches die Erfassung von Überlappungen und diskontinuierlichen Einheiten ermöglicht. XStandoff vereint hierbei wichtige Vorteile anderer Ansätze, wie der Standoff-Annotation und der Nutzung multipler Dokumente (spezifischer: multi-rooted trees). Durch den Gebrauch von Standard-XML und XML-verwandten Technologien sowie die Realisierung einer nicht-redundanten Datenhaltung wird eine nachhaltige Nutzung des Formats begünstigt.

Da die manuelle Erstellung und Bearbeitung von XStandoff-Instanzen zwar möglich ist, u.U. jedoch recht aufwändig werden kann, wurden im Rahmen der vorliegenden Masterarbeit XSLT-Stylesheets vorgestellt, die grundlegende Operationen im Zusammenhang mit XStandoff bewältigen können (siehe Kapitel 4). Sie ermöglichen die automatisierte Erstellung von XStandoff-Instanzen auf Basis einer Inline-Annotation und dem zugehörigen Primärdatum (`inline2XSF.xsl`), die Zusammenführung

von XStandoff-Instanzen, die auf dem selben Primärdatum beruhen (mergeXSF.xsl), die Löschung oder Extraktion einzelner Level und Layer (extractXSFcontent.xsl), die Überführung einer Standard-XStandoff-Instanz in eine Inline-Repräsentation (XSF2inline.xsl) sowie die Visualisierung einer XStandoff-Instanz in Form einer interaktiven SVG-Datei (XSF2SVG.xsl).

Die Stylesheets sind hierbei allerdings nicht als finale Versionen anzusehen, da z.B. die Verwendung großer Eingabedateien Schwierigkeiten hinsichtlich der Verarbeitungsgeschwindigkeit der XSLT-Transformationen verursachen kann (speziell für inline2XSF.xsl). In diesem Zusammenhang bestünde der Bedarf einer Prüfung möglicher Optimierungen.

Des weiteren gibt es bereits Ansätze zur Erweiterung der vorhandenen Stylesheets: ergänzend zu mergeXSF.xsl steht bspw. eine schema-unterstützende (schema-aware) Version mergeXSF-sa.xsl zur Verfügung, die zwar die gleiche grundlegende Funktion erfüllt, jedoch die Validierung der einzelnen Layer während der XSLT-Transformation ermöglicht. Auf diese Weise können vorhandene Dokumentgrammatiken in den Transformationsprozess einbezogen und valide XStandoff-Instanzen garantiert werden. Auch für die anderen Stylesheets wären schema-unterstützende Versionen wünschenswert. Die in XStandoff vorgesehene Repräsentation diskontinuierlicher Einheiten wird von den Stylesheets bislang leider nicht unterstützt. Diese Erweiterungsmöglichkeit könnte jedoch ebenfalls in weiteren Arbeiten aufgegriffen werden.

Da mit den bereits vorhandenen Stylesheets (und zusätzlichen denkbaren Erweiterungen) eine ganze Reihe unterschiedlicher Programme für XStandoff existieren, wäre es sinnvoll, vereinfachte Programmabläufe zu implementieren, wie z.B. die gleichzeitige Transformation mehrerer Inline-Annotationen nach XStandoff. Bislang muss zu diesem Zweck jede Eingabedatei einzeln nach XStandoff überführt werden (inline2XSF.xsl) und danach müssen die XStandoff-Instanzen paarweise in eine gemeinsame Instanz integriert werden (mergeXSF.xsl). Für die Erstellung einer XStandoff-Instanz mit Hierarchien aus 5 verschiedenen Eingabeannotationen würde

dieses insgesamt 9 Aufrufe von XSLT-Transformationen bedeuten. Zur Vereinfachung dieses Vorgehens sind verschiedene Strategien denkbar. Ein Ansatz wäre die Erstellung von Batch-Dateien (Stapelverarbeitungsprogramme), mit deren Hilfe bspw. alle Dateien, die in einem bestimmten Verzeichnis liegen, durch einen einzigen Programmaufruf nach XStandoff transformiert werden könnten. Eine andere Methode wäre der Einsatz von XProc („XML Pipeline Language"; Walsh et al., 2010) zur Spezifikation von Operationssequenzen für beliebig viele XML-Eingabedateien. Diese Möglichkeit erscheint sehr vielversprechend, wurde aber bislang noch nicht eingehend getestet.

Ein noch weitreichenderer Lösungsansatz zur Verbesserung der Programmabläufe komplexer XStandoff-Operationen bestünde in der Entwicklung eines speziellen Frameworks für XStandoff-Instanzen (in der Art eines Editors), welches z.B. die benutzerfreundliche Bearbeitung von Primärdaten und Annotationen (Update von Segmenten) mit integrierten Funktionen wie dem Import von Layern, der Zusammenführung von Instanzen, der Analyse von Elementrelationen, sowie der manuellen Zuordnung von Whitespaces und leeren Elementen zu bestimmten Zeichenpositionen (zur Behebung der Probleme von inline2XSF.xsl) in sich vereint. Zudem könnten Import- und Export-Funktionen für andere Repräsentationsformate multipler Hierarchien implementiert werden, welche sich an dem Syntaxkonvertierungsansatz von Marinelli et al. (2008) orientieren. Dieses wäre ein weiterer wichtiger Schritt in Richtung der notwendigen „creation of an unifying framework to reason formally and programmatically about overlapping markup, a particular kind of markup used to encode data structure[s] richer than those natively encodable in XML languages" (Marinelli et al., 2008: 27).

# 6 Verzeichnisse

## 6.1 Literaturverzeichnis


Bański, P. (2010). Why TEI stand-off annotation doesn't quite work: and why you might want to use it nevertheless. In *Proceedings of Balisage: The Markup Conference 2010*. Balisage Series on Markup Technologies, Vol. 5.
→ http://www.balisage.net/Proceedings/vol5/html/Banski01/BalisageVol5-Banski01.html

Bird, S. & Liberman, M. (1999). Annotation graphs as a framework for multidimensional linguistic data analysis. In *Proceedings of the Workshop „Towards Standards and Tools for Discourse Tagging“*, Association for Computational Linguistics, S. 1-10.

Bird, S. & Liberman, M. (2001). A formal framework for linguistic annotation. In *Speech Communication*, Vol. 33, S. 23-60.

Burnard, L. & Bauman, S. (Hrsg.) (2010). *TEI P5: Guidelines for Electronic Text Encoding and Interchange.* Oxford/Providence/Charlottesville/Nancy: TEI Consortium.
→ http://www.tei-c.org/release/doc/tei-p5-doc/en/Guidelines.pdf

Chiarcos, C., Dipper, S., Götze, M., Ritz, J. & Stede, M. (2008). A Flexible Framework for Integrating Annotations from Different Tools and Tag Sets. In *Proceedings of The First International Conference on Global Interoperability for Language Resources.* Hong Kong, S. 43-50.

DeRose, S. J., Durand, D. G., Mylonas, E. & Renear, A. H. (1990). What is Text, Really? In *Journal of Computing in Higher Eduction*, Vol. 1, Nr. 2, S. 3-26.

DeRose, S. J. (2004). Markup Overlap: A Review and a Horse. In *Proceedings of Extreme Markup Languages 2004*, Montreal, Kanada.
→ http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.9959&rep=rep1&type=pdf

Durusau, P. & O'Donnell, M. (2002). Concurrent Markup for XML Documents. In *Proceedings of the XML Europe conference 2002*. Barcelona, Spanien.
→ https://ssl.bnt.com/idealliance/papers/xmle02/dx_xmle02/papers/03-03-07/03-03-07.pdf

Goecke, D., Lüngen, H., Metzing, D., Stührenberg, M. & Witt, A. (2010). Different Views on Markup: Distinguishing Levels and Layers. In Witt, A. & Metzing, D. (Hrsg.). *Linguistic Modeling of Information and Markup Languages.* Text, Speech and Language Technology 40. Dordrecht: Springer, S. 1-21.

Goldfarb, C. (1996). *The Roots of SGML – A Personal Recollection.*
→ http://www.sgmlsource.com/history/roots.htm

Goldfarb, C. (1997). SGML: the reason why and the first published hint. In *Journal of the American Society for Information Science*, Vol. 48, Nr. 7, S. 656-661.

Hilbert, M., Schonefeld, O. & Witt, A. (2005). Making CONCUR work. In *Proceedings of Extreme Markup Languages 2005*, Montreal, Kanada.
→ http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.634&rep=rep1&type=pdf

Huitfeld C. & Sperberg-McQueen, C. M. (2003). *TexMECS: An experimental markup meta-language for complex documents.*
→ http://decentius.aksis.uib.no/mlcd/2003/Papers/texmecs.html

Ide, N. & Suderman, K. (2007). GrAF: A graph-based format for linguistic annotations. In *Proceedings of The Linguistic Annotation Workshop (LAW) 2007.* Prag, Tschechien, S. 1-8.

ISO (International Organization for Standardization) (1986). *ISO 8879: Information processing – Text and office systems – Standard Generalized Markup Language (SGML).* Genf: ISO.

Kay, M. (2007). *XSL Transformations (XSLT) Version 2.0*. World Wide Web Consortium – W3C Recommendation.

Kay, M. (2008). *XSLT 2.0 and XPath 2.0 Programmer's Reference.* 4. Auflage, Indianapolis: Wiley Publishing.

Kloss, J. H. (2010). *X3D: Programmierung interaktiver 3D-Anwendungen für das Internet.* München: Addison-Wesley.

Marinelli, P., Vitali, F. & Zacchiroli, S. (2008). Towards the unification of formats for overlapping markup. In: *New Review of Hypermedia and Multimedia*, Vol. 14, Nr. 1, S. 57-94.

Nassourou, M. (2010). *Markup Overlap: Improving Fragmentation Method.*
→ http://www.opus-bayern.de/uni-wuerzburg/volltexte/2010/4908/pdf/Bal2010nass0509.pdf

Pianta, E. & Bentivogli., L. (2004). Annotating Discontinuous Structures in XML: the Multiword Case. In *Proceedings of LREC 2004 Workshop on "XML-based richly annotated corpora"*, Lissabon, Portugal, S. 30-37.

Piez, W. (2010). *Towards Hermeneutic Markup: An architectural outline.* Paper präsentiert am 09.07.2010 auf der Digital Humanities 2010, King's College, London.
→ http://www.piez.org/wendell/papers/dh2010/index.html

Pondorf, D. & Witt, A. (2010). Freestyle Markup Language: Specification of an intuitive, powerful, polyhierarchical new extensible markup language. In *Proceedings of Balisage: The Markup Conference 2009*. Balisage Series on Markup Technologies, Vol. 5.
→ http://www.balisage.net/Proceedings/vol5/html/Pondorf01/BalisageVol5-Pondorf01.html

Rehm, G., Schonefeld, O., Trippel, T. & Witt, A. (2010). Sustainability of Linguistic Resources Revisited. In *Proceedings of the International Symposium on XML for the Long Haul: Issues in the Long-term Preservation of XML.* Balisage Series on Markup Technologies, Vol. 6.
→ http://www.balisage.net/Proceedings/vol6/html/Witt01/BalisageVol6-Witt01.html

Schmidt, D. (2010). The inadequacy of embedded markup for cultural heritage texts. In *Literary and Linguistic Computing*, Vol. 25, Nr. 3, Oxford: University Press, S.337-356.

Sperberg-McQueen, C. M., Huitfeld, C. & Renear, A. (2000). Meaning and interpretation of markup. In *Markup Languages: Theory and Practice*, Vol. 2, Nr.3, S. 215-234.

Sperberg-McQueen, C. M., Dubin, D., Huitfeld, C. & Renear, A. (2002). Drawing inferences on the basis of markup. In *Proceedings of Extreme Markup Languages 2002*, Montreal, Kanada.
→ http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.2721&rep=rep1&type=pdf

Sperberg-McQueen, C. M. & Huitfeld, C. (2004). GODDAG: A Data Structure for Overlapping Hierarchies. In King, P. & Munson, E. V. (Hrsg.). *DDEP-PODDP 2000.* Lecture Notes in Computer Science 2023. Berlin/Heidelberg: Springer, S. 139-160.

Sperberg-McQueen, C. M. & Huitfeld, C. (2008). Markup Discontinued: Discontinuity in TexMECS. In *Proceedings of Balisage: The Markup Conference 2008*. Balisage Series on Markup Technologies, Vol. 1.
→ http://www.balisage.net/Proceedings/vol1/html/Sperberg-McQueen01/BalisageVol1-Sperberg-McQueen01.html

Stegmann, J. & Witt, A. (2009). TEI Feature Structures as a Representation Format for Multiple Annotation and Generic XML Documents. In *Proceedings of Balisage: The Markup Conference 2009*. Balisage Series on Markup Technologies, Vol. 3.
→ http://www.balisage.net/Proceedings/vol3/html/Stegmann01/BalisageVol3-Stegmann01.html

Stührenberg, M. & Goecke, D. (2008). SGF - An integrated model for multiple annotations and its application in a linguistic domain. In *Proceedings of Balisage: The Markup Conference 2008*. Balisage Series on Markup Technologies, Vol. 1.
→ http://www.balisage.net/Proceedings/html/2008/Stuehrenberg01/Balisage2008-Stuehrenberg01.html

Stührenberg, M. & Jettka, D. (2009). A toolkit for multi-dimensional markup: The development of SGF to XStandoff. In *Proceedings of Balisage: The Markup Conference 2009*. Balisage Series on Markup Technologies, Vol. 3.
→ http://www.balisage.net/Proceedings/vol3/html/Stuhrenberg01/BalisageVol3-Stuhrenberg01.html

SyncRO Soft Ltd. (o.A.). *Oxygen XML Editor 12.1.* User Manual.
→ http://www.oxygenxml.com/doc/Oxygen-UserManual.pdf

Tennison, J. (2002). Layered markup and annotation language (LMNL). In *Proceedings of Extreme Markup Languages 2002*, Montreal, Kanada.

Walsh, N., Milowski, A. & Thompson, H. S. (Hrsg.) (2010). *XProc: An XML Pipeline Language.* W3C Recommendation 11 May 2010.
→ http://www.w3.org/TR/xproc/

Witt, A. (2001). *Multiple Informationsstrukturierung mit Auszeichnungssprachen. XML-basierte Methoden und deren Nutzen für die Sprachtechnologie.* Dissertation, Fakultät für Linguistik und Literaturwissenschaft, Universität Bielefeld.

Witt, A. (2005). Multiple Hierarchies: New Aspects of an Old Solution. In Dipper, S., Götze, M. & Stede, M. (Hrsg.). *Heterogeneity in Focus: Creating and Using Linguistic Databases.* Interdisciplinary Studies on Information Structure (ISIS) 2, Potsdam: Universitätsverlag Potsdam, S. 55-86.
→ http://www.sfb632.uni-potsdam.de/publications/isis02_4witt.pdf

Witt, A., Goecke, D., Sasaki, F. & Lüngen, H. (2005). Unification of XML Documents with Concurrent Markup. In *Literary and Linguistic Computing*, Vol. 20, Nr. 1, Oxford: University Press, S. 103-116.

Witt, A. (2007). Guideline: Multiple Hierarchies. In *Dagstuhl Seminar Proceedings 06491: Digital Historical Corpora – Architecture, Annotation, and Retrieval.*
→ http://drops.dagstuhl.de/volltexte/2007/1040/pdf/06491.WittAndreas.Paper.1040.pdf

Wörner, K. (2009). *Werkzeuge zur flachen Annotation von Transkriptionen gesprochener Sprache.* Dissertation, Fakultät für Linguistik und Literaturwissenschaft, Universität Bielefeld.

[alle Online-Dokumente waren zugänglich am 18.02.2011]

## 6.2 Abbildungsverzeichnis

## 6.3 Tabellenverzeichnis

# Anhang A: Stylesheet Dokumentation

## A.1 XSLTDoc

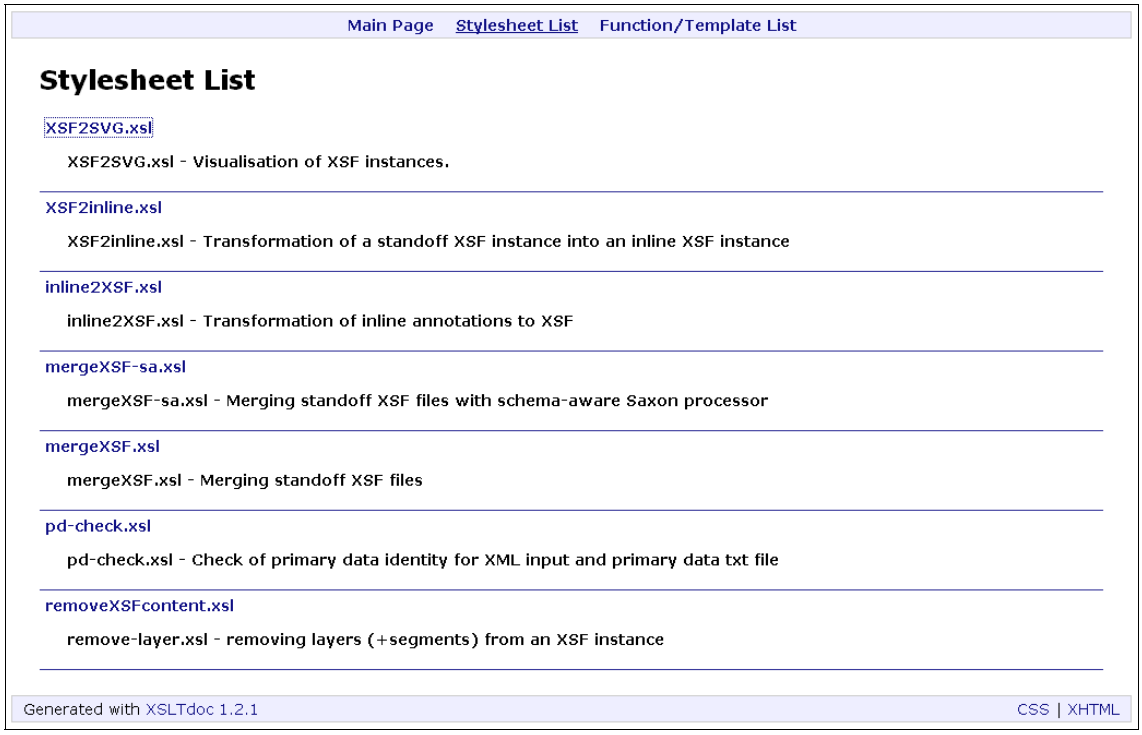



*Abbildung 13: Stylesheet-Übersicht generiert auf Basis einer Konfigurationsdatei durch XSLTDoc*

Main Page Stylesheet List Function/Template List

## XSF2SVG.xsl

**XSF2SVG.xsl - Visualisation of XSF instances.**

**Version 10.02.2011, 10:18 (GMT)**

This program is free software: you can redistribute it and/or modify it under the terms of the GNU GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU GNU Lesser General Public License for more details.

You should have received a copy of the GNU GNU Lesser General Public License along with this program (file 'lgpl-3.0.txt'). If not, see http://www.gnu.org/licenses/.

The stylesheet XSF2SVG.xsl transforms an XSF instance into an SVG file which displays the levels and layers, and their content respectively, in an hierarchical order. The correspondence of annotation and primary data is demonstrated by mouseover effects implemented in JavaScript. The underlying concept allows for the coverage of overlapping markup.

The realisation of the visualisation is strongly influenced by Wendell Piez' solution to visualising overlapping markup.

**Input XML file:** XSF instance

**Additional file (required):** a file containing the primary data in plain text

**Stylesheet parameter:** none

**Execution via command line (XSLT processor Saxon9):** java -jar saxon9.jar [optional Saxon Parameters] -o [SVG output filename] [XSF input filename] XSF2SVG.xsl

**Author:**
Daniel Jettka; daniel.jettka@uni-bielefeld.de

**Copyright:**
GNU Lesser General Public License, see below for details

### Variables Summary

`xs:double` **char-width-element-overview** - source

Width of a single char in the element names overview

`xs:integer` **count-distinct-elements** - source

Amount of distinct elements present in all XSF layers

`xs:integer` **count-lines-primary-data** - source

Amount of lines present in primary data

*Abbildung 14: Ausschnitt aus einer Stylesheet-Dokumentation mit XSLTDoc*

## A.2 Oxygen XML Editor


*Abbildung 15: Ausschnitt aus einer Stylesheet-Dokumentation mit dem Oxygen XML Editor*

# Anhang B: mergeXSF – Keys vs. Prädikate

Die intensive Nutzung von Keys in XSLT (<xsl:key> und key(); vgl. Kay, 2008: 376ff und 812ff) unterstützt eine effiziente Verarbeitung bzw. Indizierung von XML-Bestandteilen. Dieses wird insbesondere in dem Vergleich zweier Stylesheet-Versionen von mergeXSF.xsl deutlich, die unterschiedliche Entwicklungsstadien widerspiegeln. Sie sind auf der beiliegenden CD zu finden: „key-test\xsl\old-mergeXSF.xsl“ und „key-test\xsl\mergeXSF.xsl“. Ein Beispiel für die Optimierung des Stylesheets mergeXSF.xsl durch den Gebrauch von Keys wird im Folgenden anhand von Ausschnitten der beiden Stylesheet-Versionen, die die gleiche Funktion erfüllen, verdeutlicht. Der folgende Ausschnitt stammt aus der alten Version old-mergeXSF.xsl:

```
1   <xsl:variable name="XSF1segments" as="element()+">
2      <xsl:for-each select="$XSF1//xsf:segment">
3         <xsf:segment XSF1ID="{@xml:id}"
4             XSF2ID="
5             {$XSF2//xsf:segment[
6                       number(@start)=number(current()/@start)
7                       and
8                       number(@end)=number(current()/@end)
9                    ]/@xml:id}">
10          <xsl:copy-of select="@*[name()!='xml:id']"/>
11       </xsf:segment>
12    </xsl:for-each>
13  </xsl:variable>
```

Dieser Teil des Stylesheets ist dazu gedacht, für alle xsf:segment-Elemente aus der globalen Variable $XSF1 (Zeile 2) jeweils ein neues Element  (beginnt in Zeile 3) in der Variablen $XSF1segments zu speichern. Hierbei soll der Wert des Attributs @xml:id als Wert von @XSF1ID übernommen werden (Zeile 3). Der optimierbare Teil ist bei der Ermittlung des Werts für @XSF2ID zu finden. Hier soll der @xml:id-Wert von xsf:segment-Elementen aus der globalen Variablen $XSF2

(beginnend in Zeile 5) gespeichert werden, wobei die gesuchten Elemente aus $XSF2 die Bedingung erfüllen müssen, dass die Werte ihrer start- und end- Attribute mit denen des jeweils aktuellen xsf:segment-Elements aus $XSF1 (current()) übereinstimmen (Zeile 6-8).

Um die aufwändige Prüfung des XPath-Ausdrucks zu vereinfachen, ist es möglich einen Key zu definieren, welcher xsf:segment-Elemente (bzw. ihren @xml:id-Wert; Zeile 2) anhand der start- und end-Attribute (Zeile 3) des Elements indiziert:

```
1   <xsl:key name="seg-id-by-start-and-end"
2      match="xsf:segment/@xml:id"
3      use="../concat(@start, '#', @end)"/>
```

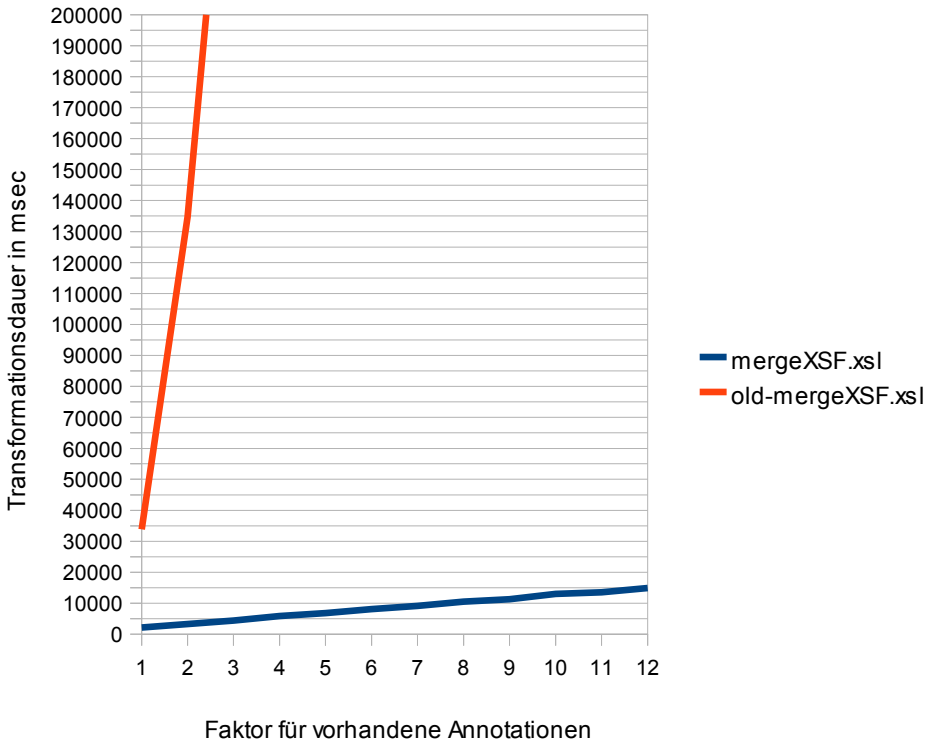
Mit Hilfe dieses Keys kann der XPath-Ausdruck in der neuen Stylesheet-Version durch die Anwendung der key()-Funktion ersetzt werden (Zeile 5-9), welche die gleiche Funktion erfüllt:

```
1   <xsl:variable name="XSF1segments" as="element(xsf:segment)*">
2      <xsl:for-each select="$XSF1//xsf:segment">
3         <xsf:segment XSF1ID="{@xml:id}"
4             XSF2ID="{
5                key(
6                   'seg-id-by-start-and-end',
7                   concat(@start, '#', @end),
8                   $XSF2
9                )}">
10            <xsl:copy-of select="@*[name()!='xml:id']"/>
11         </xsf:segment>
12      </xsl:for-each>
13  </xsl:variable>
```

Da die Indizierung von Elementen oder Attributen durch Keys nur einmal zu Beginn einer XSLT-Transformation erfolgt, führt das beschriebene Vorgehen, welches noch an weiteren Stellen im Stylesheet angewendet wurde, zu einem deutlichen Anstieg der Transformationsgeschwindigkeit. Dieses lässt sich an einem Vergleich nachvollziehen, der auf Basis der beiden Stylesheet-Versionen durchgeführt wurde. Hierzu wurden die XML-Annotationen buergschaft-silbe.xml und buergschaft-wort.xml mit Hilfe des Primärdatums buergschaft-pd-utf8.txt[26] durch inline2XSF.xsl

26 Alle erwähnten Dateien sind auf der beiliegenden CD in „key-test\pd“ und „key-test\xml“ zu finden

nach XStandoff überführt. Um eine kontinuierliche Steigerung der Komplexität der Transformation zu erreichen, wurden zusätzliche Annotationen erstellt, die die Ausgangsannotationen der buergschaft-Dokumente mehrfach enthalten (siehe z.B. buergschaft-wortX2.xml). Die korrespondierenden Primärdaten beinhalten den entsprechend vervielfältigten Text (z.B. buergschaft-pdX2.txt). Durch dieses Vorgehen kann gezeigt werden, dass die Dauer der Zusammenführung der XStandoff-Instanzen buergschaft-wortX(...)-xsf.xml und buergschaft-silbeX(...)-xsf.xml[27] durch das Stylesheet mergeXSF.xsl ungefähr linear von der Anzahl der vorhandenen Segmente abhängt. Im Gegensatz dazu steigt die Transformationsgeschwindigkeit für die alte Stylesheet-Version old-mergeXSF.xsl exponentiell an:




Hier zeigt sich, dass der Einsatz von XSLT-Keys ein äußerst wirkungsvolles Instrument zur Optimierung von XSLT-Stylesheets ist.

27 Beiliegende CD, Ordner „key-test\xsf“

# Anhang C: XStandoff-Visualisierungen

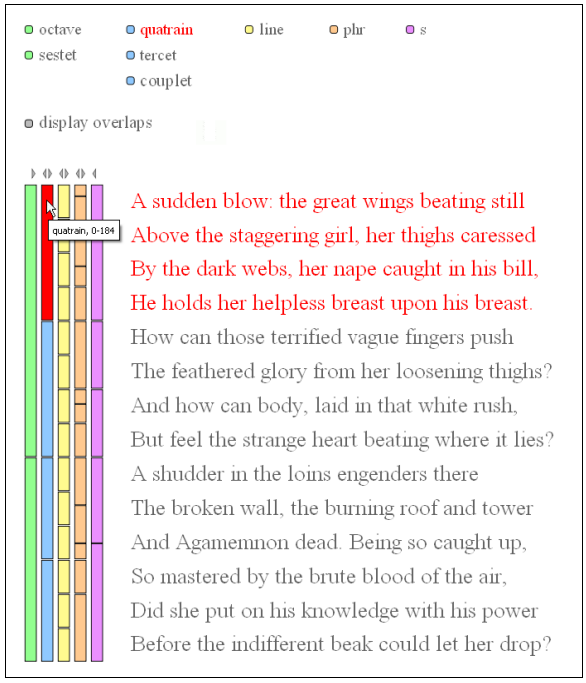



*Abbildung 16: Mouseover-Effekt für Elemente*

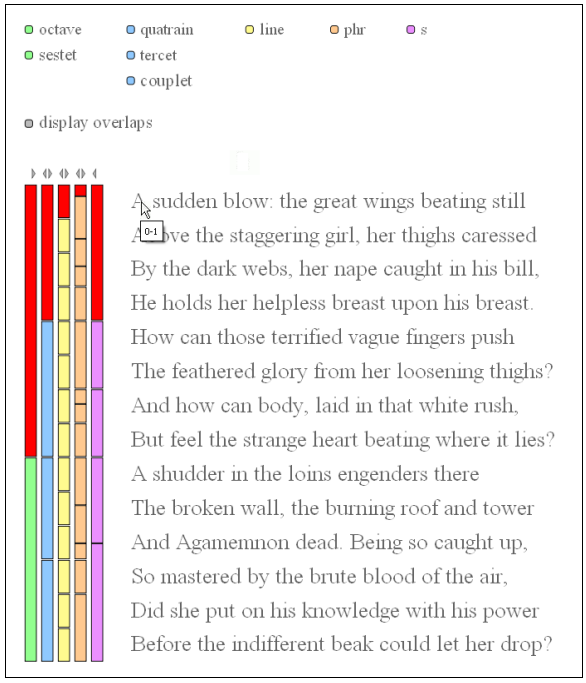



*Abbildung 17: Mouseover-Effekt für Primärdaten*

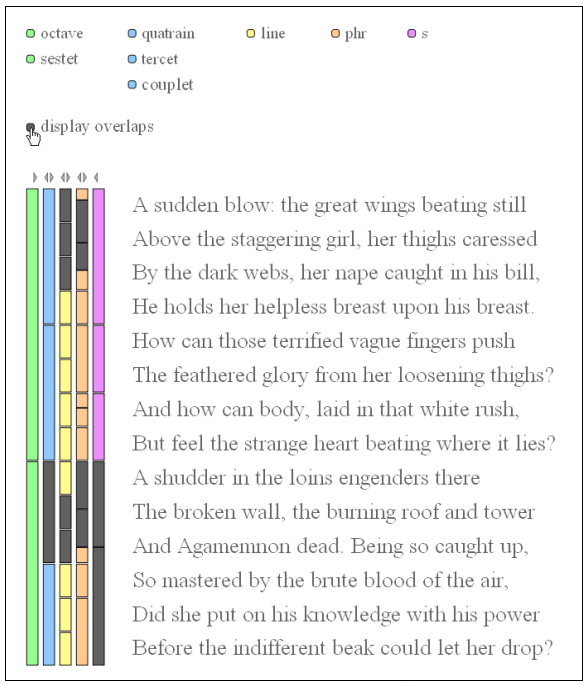



*Abbildung 18: Anzeige klassischer Überlappungen*

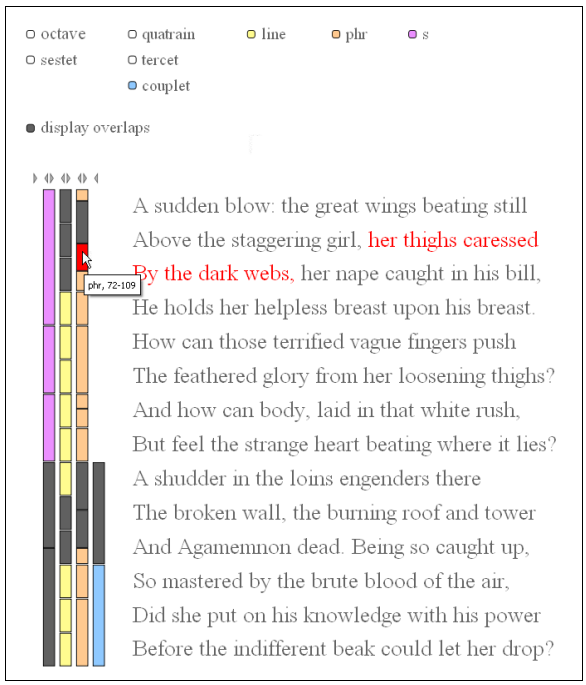



*Abbildung 19: Reorganisation von Layern/Ausblenden von Elementtypen*

# Erklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbständig verfasst und gelieferte Datensätze, Zeichnungen, Skizzen und graphische Darstellungen selbst-ständig erstellt habe. Ich habe keine anderen Quellen als die angegebenen benutzt und habe die Stellen der Arbeit, die anderen Werken entnommen sind - einschl. verwendeter Tabellen und Abbildungen - in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht.

Bielefeld, den

_____

(Unterschrift)