

# **Efficient Conversion of Scientific Legacy Documents into Semantic Web Resources**

**using biosystematics as a working example**

zur Erlangung des akademischen Grades eines

**Doktors der Ingenieurwissenschaften**

von der Fakultät für Informatik  
des Karlsruher Instituts für Technologie (KIT)

**genehmigte**

**Dissertation**

von

**Guido Sautter**

aus Königswinter

Tag der mündlichen Prüfung: 09. Februar 2011

Erster Gutachter: Prof. Dr.-Ing. Klemens Böhm

Zweiter Gutachter: Prof. em. Robert A. Morris



## **Dedication**

I dedicate this thesis to my family and friends. It is your constant reassurance and support that have enabled me to create this work.



## Zusammenfassung

### 1 Einleitung

Wissen wird heute konsequent in digitaler Form gespeichert. Tatsächlich ist das jedoch erst seit ca. 30 Jahren der Fall. Zuvor gewonnenes Wissen wurde, ebenso wie die zugrunde liegenden Daten, in gedruckter Form in Bibliotheken und Archiven gespeichert und ist damit nur wenigen Personen zugänglich. In den letzten 10 Jahren haben Projekte wie Google Books, Internet Archive und Biodiversity Heritage Library (BHL) damit begonnen, in großem Stil den Inhalt ganzer klassischer Bibliotheken in digitale Form zu überführen. Hierbei werden die Dokumente zuerst abfotografiert und dann mittels Texterkennung (Optical Character Recognition, OCR) in maschinenverarbeitbaren Text umgewandelt. Hierdurch wird der Inhalt der Dokumente elektronisch durchsuchbar und kann damit volltext-indiziert und über das Internet einer breiten Öffentlichkeit zugänglich gemacht werden.

So digitalisierte Dokumente sind allerdings nur für menschliche Leser verständlich. Eine maschinelle Verarbeitung des enthaltenen Wissens ist nicht möglich, also etwa die Visualisierung von Orten auf einer Karte oder die Verknüpfung und Vernetzung mit Wissen aus anderen digitalen Quellen. Hierfür müssen die eigentlichen Daten in eine nicht nur maschinenlesbare, sondern auch maschineninterpretierbare Form gebracht werden. Dies bedeutet, dass der logische Inhalt maschinenverständlich ist. Hierfür muss beispielsweise das kontextuelle Wissen, das der menschliche Verstand beim Lesen unterbewusst zum Verständnis hinzuzieht, explizit gemacht werden. Nur so steht das kontextuelle Wissen der maschinellen Verarbeitung zur Verfügung. Und nur so lassen sich sinnvolle Ergebnisse erreichen, insbesondere bei der Vernetzung der Informationen.

Beispiel: Wenn ein Text aus dem Jahr 1955 den „Bundeskanzler“ erwähnt, so ist dem menschlichen Leser implizit klar, dass dies in diesem Kontext ein Synonym für die Person „Konrad Adenauer“ darstellt. Für die maschinelle Verarbeitung müssen zwei Detailinformationen explizit dargestellt werden: erstens dass die Zeichenfolge „Bundeskanzler“ ein Person bezeichnet, und zweitens dass diese Person durch den Namen „Konrad Adenauer“ identifiziert ist.

Wie im Beispiel bereits angedeutet, spielen Bezüge zu realweltlichen Dingen (sogenannte „benannten Entitäten“, Named Entities, beispielsweise Personen, Organisationen, Datumsangaben, Orte, etc) für die maschinelle Verarbeitbarkeit eine besonders wichtige Rolle. Doch auch die logische Struktur eines Dokumentes ist von entscheidender Bedeutung, da sie die erwähnten Named Entities zueinander in Beziehung setzt. Und erst die Zusammenhänge zwischen Named Entities stellen in engerem Sinne Informationen dar. Allein das korrekte Markieren der Named Entities, das Hinzufügen der Kontext-Informationen und die Markierung der Dokument-Struktur ist ein aufwendiger Vorgang. Zudem müssen die Dokumente von Text befreit werden, der erst bei der Drucklegung eingefügt wurde, wie etwa Seitentitel.

Die in diesen enthaltenen Informationen stehen in keinem Zusammenhang zum umgebenden eigentlichen Text und stellen daher eine Quelle von Fehlern und Mehrdeutigkeiten bei der maschinellen Verarbeitung der Daten dar.

## **2 Problembeschreibung**

Die manuelle Konvertierung digitalisierter Dokumente in eine maschineninterpretierbare Form ist mit prohibitivem Aufwand verbunden, zumal die Konvertierung wissenschaftlicher Dokumente oftmals Expertenwissen der jeweiligen Disziplin erfordert.

Eine vollautomatische maschinelle Konvertierung würde diesen Aufwand umgehen, ist jedoch nicht möglich. Zwar existieren für einige Teile der Konvertierung, beispielsweise das Markieren von Named Entities im Bereich der Natürlichen Sprachverarbeitung (Natural Language Processing, NLP), seit einiger Zeit automatische Verfahren. Die Ergebnisse dieser Verfahren sind allerdings (probleminhärent) selten genauer als 95%. Dies ist bei weitem unzureichend, vor allem da die Konvertierung ein komplexer vielschrittiger Prozess ist, dessen Einzelschritte aufeinander aufbauen. Arbeitet nun jeder Schritt 95% genau, fehlerfreie Eingangsdaten vorausgesetzt, so beeinträchtigen die 5% Fehler die nachfolgenden Schritte. Die Datenqualität des fertig konvertierten Dokumentes wird letztendlich so stark gemindert, dass von der maschinellen Verarbeitung der (nun maschineninterpretierbaren) Informationen keine sinnvollen Ergebnisse zu erwarten sind. Insbesondere Schlussfolgerungen aus der Vernetzung der Informationen wären kaum mehr als beliebig, da eine einzige falsch markierte Information in einer Herleitungskette ausreicht, um zu einer vom Dokumentinhalt nicht gestützten Folgerung zu kommen. Würde das Wort „Bundeskanzler“ im Beispiel etwa als „Helmut Kohl“ interpretiert, so würde dies eventuell Schlussfolgerungen über letzteren implizieren, die eigentlich auf Konrad Adenauer zutreffen. Bei einer weitreichenden Verknüpfung vieler Informationen hätte das Ergebnis der maschinellen Interpretation nichts mehr mit der Realität zu tun.

Quintessenz: Bei der Konvertierung digitalisierter Dokumente in eine maschineninterpretierbare Form muss darauf geachtet werden, dass die maschineninterpretierbar gemachten Fakten genau mit den verbal beschriebenen übereinstimmen. Andernfalls ist die Verwendbarkeit der durch die Konvertierung gewonnenen Daten stark eingeschränkt. Bisher existiert kein Verfahren, um die erforderliche hohe Datenqualität mit akzeptablem manuellem Aufwand zu erreichen.

## **3 Beitrag der Arbeit**

Die vorgelegte Arbeit beschreibt ein semiautomatisches Verfahren zur Konvertierung digitalisierter Dokumente in eine maschineninterpretierbare Form. Hierbei reduzieren existierende automatische Verfahren den manuellen Aufwand der einzelnen Konvertierungsschritte, während die manuelle Korrektur der automatisch erstellten Ergebnisse die Fortpflanzung von Fehlern verhindert und so die Datenqualität der konvertierten Dokumente sicherstellt.

Der Hauptteil der Arbeit befasst sich mit der Optimierung dieses Verfahrens. Sie entwickelt Ansätze zur Unterstützung des Benutzers auf unterschiedlichen Ebenen, die den manuellen Aufwand weiter reduzieren. Im Einzelnen:

1. Die Komplexität des manuellen Bearbeitens von Dokumenten und der Bedienung oft kommandozeilenbasierter NLP-Werkzeuge wurde in ein für Domänenexperten gut bedienbares Konvertierungswerkzeug gekapselt.
2. Die NLP-Werkzeuge wurden auf gute Korrigierbarkeit ihrer Fehler hin optimiert. Es hat sich gezeigt, dass hierbei die Ausbeute wichtiger ist als die Genauigkeit: Nur eine 100%ige Ausbeute vermeidet beispielsweise das aufwendige Suchen übersehener Named Entities im Dokument.
3. Im Rahmen einer Feldstudie wurden Regeln erstellt, die helfen, die Reihenfolge der einzelnen Schritte im Konvertierungsprozess so zu optimieren, dass jedes einzelne NLP-Werkzeug eine optimale Basis für seine Entscheidungen hat. Dies vermindert Fehler und senkt damit den Korrekturaufwand.
4. Konvertierungsprozesse sind komplex. Dies fördert das Übersehen von Fehlern bei der manuellen Korrektur, und auch das versehentliche Auslassen von Schritten, wodurch weitere Fehler entstehen. Um dem Benutzer das Erlernen komplexer Konvertierungsprozesse abzunehmen und das Übersehen von Fehlern zu verhindern wurde ein Mechanismus entwickelt, der den Benutzer durch den Prozess führt und ihn beim Korrigieren auf mögliche verbliebene Fehler hinweist.
5. Die durchgeführten Studien haben gezeigt, dass etwa 50% des Aufwandes auf die Bereinigung der Dokumente entfallen, die kein Domänenwissen erfordert. Daher wurde eine Infrastruktur geschaffen, die das Auslagern der betreffenden Schritte an eine Benutzergemeinschaft im Internet (Crowd-sourcing) ermöglicht und so die Domäneexperten weiter entlastet.

Das vorgestellte Konvertierungsverfahren wurde in mehreren Dokumentdigitalisierungs- und -aufbereitungsprojekten eingesetzt, jeweils mit begleitenden Feldstudien. Insgesamt haben Biologen während dieser Projekte über 5.000 Seiten biosystematischer Literatur konvertiert. Das Verfahren hat sich als sehr geeignet zur Lösung des gestellten Problems erwiesen: Mit einem normalen textbasierten XML-Editor beträgt der durchschnittliche Aufwand pro Seite bei der Konvertierung eines Dokuments ca. 30 Minuten; bei Einsatz der im Rahmen der Arbeit entwickelten Werkzeuge sinkt dieser Aufwand auf etwas über eine Minute. Das Ergebnis wurde in Laborexperimenten mit Koch-Literatur unter kontrollierten Bedingungen bestätigt, was auch die Übertragbarkeit des Verfahrens in andere Domänen zeigt.

Eine Infrastruktur zu Auslagerung des Benutzeraufwandes für die Normalisierung der Dokumente an eine Benutzergemeinschaft im Internet wurde geschaffen. Um mit Fehlern umzugehen, die Mitglieder einer solchen Gemeinschaft beim Bearbeiten der Dokumente eventuell machen, beinhaltet diese Infrastruktur einige neu entwickelte generische Mechanismen zur Sicherung der Datenqualität, die mit deutlich geringerer Redundanz auskommen als bisherige Mechanismen und daher den Durchsatz erhöhen. Im praktischen Einsatz hat sich leider gezeigt, dass das Korrigieren der Struktur von Seiten in OCR-Resultaten bei weitem nicht den gleichen Reiz auf die avisierte Zielgruppe ausübt wie etwa das Klassifizieren von Galaxien oder die Suche

nach außerirdischem Leben. Es sind vermutlich stärkere Anreize notwendig als Punkte in einer Rangliste um eine genügende Anzahl von Benutzer zur Beteiligung and der Normalisierung von Dokumenten zu bewegen, etwa monetäre Anreize. Sobald solche Anreize aber geschaffen sind, wird sich für die Domänenexperten die Arbeitszeit pro Seite noch einmal halbieren, da über alle anderen Ansätze hinweg der Arbeitszeit-Anteil der Schritte, die Expertenwissen erfordern, bei etwa der Hälfte der gesamten Arbeitszeit pro Seite lag, unabhängig von anderweitigen Unterstützungsmechanismen.

#### **4 Fazit**

Die vorgelegte Arbeit beschreibt ein Verfahren zur semiautomatischen Konvertierung digitalisierter Dokumente in eine maschineninterpretierbare Form, unter Sicherung der Datenqualität durch Experten. Der Konvertierungsprozess wurde studiert und in mehreren, orthogonalen Richtungen optimiert. Dadurch konnte der benutzerseitige Aufwand von ca. 30 auf ca. eine Minute pro Dokument-Seite gesenkt werden. Diese Werte wurden sowohl in Labor-Experimenten als auch in längerfristigen Praxis-Studien nachgewiesen. Durch Crowdsourcing sollte sich der Aufwand für die Domänenexperten noch einmal halbieren lassen.

Ein Großteil der Forschungsarbeit fand vor einem biosystematischen Hintergrund statt. Jedoch ist lediglich ein Bruchteil der Ergebnisse spezifisch für die Biosystematik, so dass die gewonnenen Erkenntnisse auf andere Domänen übertragbar sind. Auch die Übertragung der entwickelten Werkzeuge sollte sich auf kleinere Anpassungen beschränken.

#### **5 Grundlegende Annahmen**

Einige wenige grundlegende Annahmen stehen hinter den Mechanismen und Ergebnissen, die im Hauptteil diese Arbeit vorgestellt werden, vornehmlich solche in Bezug auf die Motivation, die Fähigkeiten, die Einschränkungen und das Verhalten der Benutzer:

1. Benutzer sind grundlegend wohlwollend und nicht darauf aus, in einzelnen Dokumenten oder gar am gesamten Aufbereitungssystem Schaden anzurichten. Diese Annahme liegt darin begründet, dass Domänenexperten ein inhärentes Interesse an der Qualität der aufbereiteten Dokumente haben, da sie die extrahierten Daten für ihre eigene Arbeit nutzen können.
2. Jedem Benutzer kann allerdings gelegentlich ein Fehler unterlaufen. Diese Annahme modelliert zwei Probleme, die bei der manuellen Datenbearbeitung inhärent auftreten: Erstens ist das zeichenweise Bearbeiten von XML insbesondere für Nicht-Informatiker eine Herausforderung, und zweitens ist das strikte Einhalten eines vorgegebenen Prozesses ebenso fordernd.
3. Häufige Ausführung simpler Standard-Aufgaben schreckt Benutzer ab und senkt ihre Motivation; dies ist beim Entwurf der Werkzeuge für die manuelle Dokumentbearbeitung zu berücksichtigen.



4. Technische Komplexität schreckt insbesondere Nicht-Informatiker ab, also beispielsweise Experten aus anderen Domänen, und muss daher so weit wie möglich vor ihnen verborgen werden.

Der Crowdsourcing-Ansatz (Beitrag 5, siehe Kapitel 10) stützt sich nicht auf die erste Annahme, da generelles Wohlwollen der Benutzer in einer anonymen Online-Gemeinschaft nicht generell vorausgesetzt werden kann, insbesondere nicht wenn die Benutzer keinen spezifischen Bezug zu den Dokumenten haben, die sie bearbeiten. Andererseits müssen die anderen drei Annahmen in einer solchen Benutzergemeinschaft als verschärft zutreffend betrachtet werden, da sie generelle Einschränkungen modellieren.



## Table of Contents

1	Overview .....	1
1.1	Current Situation .....	1
1.2	Problem Statement .....	2
1.3	Contribution .....	3
1.4	Results .....	4
1.5	Basic Assumptions .....	4
2	Preliminaries .....	7
2.1	XML .....	7
2.1.1	Data Centric XML .....	8
2.1.2	Document Centric XML .....	9
2.1.3	Summary & Discussion .....	11
2.1.4	Semantic Markup .....	11
2.2	XML Validation Techniques .....	13
2.2.1	XML Schema .....	13
2.2.2	SchemaTron .....	13
2.3	Natural Language Processing .....	14
2.3.1	Common NLP Tasks .....	15
2.3.2	Error Metrics .....	16
2.3.3	Common NLP Decision Models .....	17
2.3.4	Machine Learning Techniques .....	18
2.4	Machine Reasoning .....	19
3	Basic Assumptions & Requirements .....	21
3.1	Assumptions Regarding Existing Algorithms & Technology .....	21
3.2	Assumptions Regarding Human Users .....	22
3.3	Assumptions Regarding Data .....	23
4	Example Scenarios .....	25
4.1	The Biosystematics Scenario .....	25
4.2	The Pasta Recipe Scenario .....	30
5	Basic Approach & Research Problems .....	33
5.1	Automated Semantic Markup Generation .....	33
5.2	Manual Semantic Markup Generation .....	33
5.3	Discussion & Synthesis .....	33
5.4	Research Problems .....	35
5.5	Design Goals .....	37
6	Related Work .....	41
6.1	Markup Visualization .....	41
6.1.1	Syntax Highlighting .....	41
6.1.2	Code Folding .....	42
6.1.3	Semantic Coloring .....	42
6.1.4	Summary .....	43

6.2	Existing Applications & Frameworks .....	43
6.2.1	Text and XML Editors.....	43
6.2.2	GATE .....	44
6.2.3	Knowtator .....	45
6.2.4	WordFreak.....	46
6.2.5	Summary .....	47
6.3	Specialized Editing Views .....	48
6.4	Natural Language Processing Implementations & Models .....	49
6.4.1	Data Models .....	49
6.4.2	NLP Implementations.....	51
6.4.3	Incremental Learning Techniques .....	52
6.5	Process Control Mechanisms .....	53
6.5.1	Workflow Management Systems.....	53
6.5.2	Annotation Control Mechanisms.....	54
6.6	Crowdsourcing .....	55
6.6.1	Amazon Mechanical Turk .....	55
6.6.2	Competitive Games .....	56
6.6.3	Other Crowdsourcing Applications .....	56
6.7	User Motivation & Data Quality Enforcement Mechanisms .....	57
6.7.1	Countering Accidental Errors.....	57
6.7.2	Countering Cheating.....	58
7	Assisting Users in Semantic Markup Generation .....	61
7.1	General Application Design.....	61
7.1.1	Encapsulation of NLP Tools.....	61
7.1.2	Document Display .....	62
7.1.3	Editing Facilities for Document Text and Markup.....	63
7.2	Specialized Views for Specific Correction Tasks .....	64
7.3	The GAMTA Data Model.....	64
7.4	The GoldenGATE Editor .....	65
7.4.1	The Document Editor .....	65
7.4.2	The Extension Plug-In Host .....	67
7.5	Evaluation .....	68
7.5.1	Experimental Design .....	69
7.5.2	Pilot Study .....	69
7.5.3	Tutorial .....	70
7.5.4	Participants .....	70
7.5.5	Tasks.....	71
7.5.6	Sample Size .....	71
7.5.7	Document Quality .....	72
7.5.8	Results .....	72
7.5.9	Questionnaire.....	73
7.6	Discussion .....	74
8	Optimizing Interactive Semantic Markup Generation .....	75
8.1	The Madagascar Corpus Project.....	75
8.1.1	The Madagascar Corpus.....	76
8.1.2	The Applications .....	76

8.2	The Markup Steps .....	77
8.2.1	Document Cleanup & Normalization .....	77
8.2.2	Structural Markup.....	78
8.2.3	Detail Markup.....	79
8.3	Optimization of Markup Steps .....	80
8.3.1	Markup Task Classification.....	80
8.3.2	Improvements during the Madagascar Corpus Project.....	82
8.3.3	Generalization .....	83
8.4	Process Level Optimization .....	84
8.4.1	Dependencies between Semantic Markup Generation Steps.....	84
8.4.2	Dependency Based Optimization of Step Order.....	86
8.4.3	The TaxonX Process.....	86
8.4.4	Generalization .....	88
8.5	Discussion .....	89
8.6	Optimizing NLP Components for Interactive Use .....	90
8.6.1	Specific Requirements.....	90
8.6.2	Suitability of Common NLP Data Models .....	91
8.6.3	Design Considerations.....	93
8.6.4	Avoid Error, Reduce Uncertainty.....	93
8.6.5	Metrics for Interactivity.....	95
8.6.6	Excursion: the FAT Taxonomic Name Recognizer.....	95
9	Controlling Complex Semantic Markup Processes.....	97
9.1	ProcessTron.....	98
9.1.1	Describing Markup Processes with SchemaTron Schemas .....	98
9.1.2	ProcessTron Execution Model.....	99
9.2	Evaluation .....	100
9.2.1	Experimental Setup .....	101
9.2.2	Laboratory Experiment.....	102
9.2.3	The ZooTaxa Corpus Project.....	103
9.3	Modeling the TaxonX Process .....	104
9.3.1	Modeling Possibly Traceless Steps .....	104
9.3.2	Further Process Modeling Guidelines .....	106
9.4	Discussion .....	107
10	Relieving Experts from Non-Expert Work.....	109
10.1	Decisions, Tasks & Functions.....	110
10.2	Types of Errors.....	113
10.2.1	Accidental Errors.....	114
10.2.2	Cheating Errors.....	114
10.2.3	Destructive Errors.....	114
10.2.4	Combined Error Probability .....	114
10.3	Parameters & Figures .....	115
10.4	Related Work, Revisited .....	115
10.4.1	r-Redundancy .....	115
10.4.2	Agreement Games .....	117
10.4.3	Centrality-based Approaches.....	118
10.4.4	ReCAPTCHA.....	118

10.5	High-Throughput Crowdsourcing .....	119
10.5.1	Running Example .....	119
10.5.2	Base Case .....	120
10.5.3	v-Voting .....	120
10.5.4	Vote Boosting.....	124
10.5.5	Sampled Probing .....	127
10.6	Simulations.....	131
10.6.1	Experimental Setup .....	131
10.6.2	Results .....	132
10.7	Deployment Experience .....	134
10.7.1	Research Hypotheses.....	136
10.7.2	Experimental Setup .....	136
10.7.3	Results .....	137
10.7.4	Discussion .....	137
10.8	Conclusions .....	137
11	Conclusions & Future Work.....	139
	Acknowledgements .....	141
	References .....	143
	Appendix .....	149
A.	The GAMTA Data Model .....	149
B.	The GoldenGATE Editor.....	151
C.	GoldenGATE Evaluation Questionnaire .....	152
D.	The GoldenGATE Server System .....	154

## Index of Figures

Figure 4.1: Excerpt from scanned document .....	26
Figure 4.2: The XML Schema for the pasts recipes .....	30
Figure 6.1: The document from Example 2.3.2 with XML syntax highlighting .....	41
Figure 6.2: The document from Example 2.3.2 with elements folded away .....	42
Figure 6.3: The document from Example 2.3.2 with Semantic Coloring .....	43
Figure 6.4: The GATE user interface .....	45
Figure 6.5: The Knowtator user interface .....	46
Figure 6.6: WordFreak user interface for NER.....	47
Figure 6.7: WordFreak user interface for POS and Chunking <sup>9</sup> .....	47
Figure 7.1: The annotation editor .....	66
Figure 7.2: List view of XML elements for correcting detail markup .....	68
Figure 7.3: Counterbalanced experimental design.....	69
Figure 7.4: XML schema for the experiment.....	71
Figure 7.5: Boxplot of the completion time distributions .....	72
Figure 8.1. The Madagascar Corpus Process .....	87
Figure 8.2: Average working time per page .....	89
Figure 9.1: The ProcessTron execution model .....	99
Figure 9.2: Document size in pages & working time per page in minutes .....	103
Figure 10.1: Effects of Vote Boosting .....	132
Figure 10.2: Per-task and overall payoff depending on cheating.....	133
Figure 10.3: A page structuring task in the Facebook application.....	135





## **Index of Tables**

Table 2.1: Error metrics example .....	16
Table 4.1: Types of subsections in treatments .....	28
Table 8.1: Time per step (in minutes) for 10 pages .....	90
Table 9.1: Results of laboratory experiment.....	102
Table 9.2: Measurements from ZooTaxa Project.....	104
Table 10.1: Inputs per task and remaining error .....	132
Table 10.2: Inputs required for achieving 99.5% result accuracy.....	134
Table C.1: Users' detail assessment of the GoldenGATE Editor .....	152



## Index of Examples

Example 1.1: Text understanding requires contextual knowledge .....	1
Example 2.1: A data centric XML document .....	8
Example 2.2: An XHTML document .....	9
Example 2.3.1: A document centric XML document with semantic markup .....	10
Example 2.3.2: The same XML document with additional semantic markup .....	11
Example 2.4: Data centric record extracted from the document in Example 2.3.2 .....	12
Example 2.5: A SchemaTron rule that validates the markup of sentences .....	14
Example 2.6: Data from Example 2.4 represented in RDF/XML .....	20
Example 4.1: Main text and caption mixed up (hypothetical) .....	26
Example 4.2: OCR output for Figure 4.1 and page above it .....	27
Example 4.3: OCR output from Example 4.2 after semantic markup .....	29
Example 6.1: Slash-Tag notation for named entities .....	49
Example 6.2: Tag-Array format for named entities .....	50
Example 6.3: Spans marking named entities .....	50
Example 9.1: A ProcessTron rule .....	98
Example 9.2: Using side effects of steps .....	105
Example 9.3: Using regular expression patterns .....	105
Example 9.4: Using multiple related types of markup elements .....	106
Example 9.5: Formulating rules dependent on NLP tools .....	106
Example 10.1: Instances of concrete input aggregation functions .....	112
Example 10.2: Ambiguous decisions in r-Redundancy .....	116
Example 10.3: The benefit of decision-wise voting .....	121
Example 10.4: Result accuracy comparison for 2-Voting in the example scenario ..	122
Example 10.5: Throughput of 2-Voting in the example scenario .....	123
Example 10.6: Throughput of 2-Voting with other exogenous parameter values ...	123
Example 10.7: Development of boost probability .....	126
Example 10.8: Increased payoff due to cheating .....	128
Example 10.9: Probing a user .....	129
Example 10.10: Expected payoff for cheating with Sampled Probing .....	130



# 1 Overview

Printed documents hold almost all human knowledge gathered more than 30 years ago, and thereby confine its lion's share to libraries. This work tackles efficient extraction of knowledge from digitized literature. After an overview of the current situation, including document digitization projects, this chapter defines the problem of accurate and efficient extraction of knowledge from digitized text. It then gives a brief summary of how later chapters of this work tackle the efficiency problem in multiple orthogonal dimensions, as well as the results of these efforts. Finally, this chapter briefly introduces the basic assumptions underlying this work.

## 1.1 Current Situation

Today, human knowledge is stored strictly in digital form. However, this has been the case only for about 30 years. Knowledge gathered before, as well as its underlying data, was stored in printed form in libraries and other archives, and thus is accessible only to very few people. In the last 10 years, projects like Google Books, Internet Archive and Biodiversity Heritage Library (BHL) have started digitizing entire classical libraries at a large scale. Their process consists of first scanning the documents, and then OCR-ing them into machine readable text. As a result, search engines can full text index the documents' content and make it available to the public over the Internet. For instance, the Biodiversity Heritage Library (BHL) [BHL], a mass digitization project in the biology domain, has digitized and OCR-processed over 86.000 documents with a total of over 32 million pages, as of January 2011.

However, only human readers can actually understand documents digitized with the above process. Machine processing of their contained knowledge remains impossible, be it visualizing locations on a map, be it linking data to other sources, or be it machine reasoning. These applications require the actual data in a form that is not only machine readable, but also actually machine interpretable, i.e., the documents' logical content is machine understandable. This implies, for instance, that the contextual knowledge a human brain involuntarily uses to understand a text has to be explicit. Only then this contextual knowledge is available during machine processing, and only then results have a chance to be meaningful, especially in machine reasoning.

Consider a text from the year 1905 contains the word „President“. A human reader will implicitly understand that in this time context this is a synonym of or reference to the person „Theodore Roosevelt“. Machine processing, on the other hand, requires two facts to be explicit: that the character sequence „President“ refers to a person, and that the name „Theodore Roosevelt“ is the identifier of this person.

### **Example 1.1: Text understanding requires contextual knowledge**

As Example 1.1 already hints, references to named entities (persons, organizations, dates, locations, etc.) are highly important for machine processing. The logical document structure is just as important, however, as it specifies the relation between the named entities mentioned, and only these relations are actual information in a narrower sense. Marking the named entities, adding contextual information, and marking the document structure is considerable effort. In addition, however, the documents have to be cleared from text that was added only in print layout, e.g. page headings. This is because the information contained in the latter is in no way related to the information given in the document content and thus is a source of errors and ambiguities in machine processing.

## 1.2 Problem Statement

Manual conversion of digitized documents into a machine interpretable form is prohibitive effort, even more so as the conversion of scientific documents often requires considerable domain knowledge.

Fully automated conversion would circumvent this effort, but is impossible. Natural Language Processing does provide fully automated methods for several parts of the conversion process, e.g. for marking named entities, but the accuracy of these methods rarely exceeds 95%, and inherently so. This is by far insufficient, as the conversion is a complex multi-step process whose individual steps build on each other's results. Consider every single step working at 95% accuracy if provided with 100% accurate input, the 5% of error will hamper subsequent steps. This would affect data quality of the final conversion result to such a degree that machine processing the (now machine interpretable) data is highly unlikely to yield any meaningful results; the overall accuracy figures for information extraction systems given in [Marsh 1998] strikingly show this. Especially conclusions derived through machine reasoning would be little more than arbitrary, as a single wrong piece of information in a reasoning chain is enough to reach a conclusion not supported by the actual document content. Would, for instance, the word „President“ be interpreted as a reference to „Bill Clinton“ instead of “Theodore Roosevelt” in Example 1.1, machine reasoning would reach conclusions about the former that in reality apply to the latter. Machine reasoning over large amounts of data would yield interpretations that have no relation to the real world.

In addition, there are several applications for which considerably sized fact bases of high quality are not only required, but outright crucial. Think of support systems for surgery teams that suggest adequate countermeasures to take in order to save a patient who shows a specific combination of symptoms. Similar systems are imaginable for disaster response teams, e.g. to propose reactions to specific chemical hazards in a specific environment, for environmental studies, etc.

The majority of the fact data backing such systems may well be given in printed form, published on paper over the centuries. For a fact base extracted from such documents to be useful, and safe to use in particular, maximum data quality is essential.

In essence, the above means that any mechanism that converts digitized documents into machine interpretable data has to make sure that the actual facts it makes machine interpretable are exactly the ones expressed verbally in the document text.

Otherwise, the conversion results are of less than limited use. So far, there is no conversion mechanism or approach that yields results of acceptable data quality at acceptable manual effort.

### 1.3 Contribution

This work describes a semi-automated technique for converting digitized documents into a machine interpretable form. In this technique, existing automated mechanisms reduce the manual effort of the individual conversion steps, and manual correction prevents propagation of the automated mechanisms' errors, so to ensure that the final conversion results are of high data quality.

The main part of this work focuses on optimizing the presented technique. It develops approaches and mechanisms that support the user on multiple levels, so to further reduce his manual effort. In particular:

1. A conversion tool that is easy to use for domain experts encapsulates the complexity of both manual XML editing and deployment of often console based NLP tools.
2. NLP tools have been optimized for their errors to be easy to correct. It turns out that recall is way more important than accuracy in this context: In named entity recognition, only 100% of recall avoid a cumbersome quest for false negatives through the entire document, for instance.
3. A field study has yielded rules that help optimizing the order of the individual steps of the conversion process in such a way that every single NLP tool is provided with an optimal basis for its decisions. This reduces errors and thus mitigates correction effort.
4. Conversion processes in themselves are complex. This increases the chance of missing errors during manual correction and of accidentally skipping entire steps, which results in additional errors. A dedicated mechanism both relieves users from learning complex conversion processes and prevents them from missing errors. In particular, this mechanism guides users through the conversion process and points them to remaining possible errors during manual correction.
5. Multiple experiments and field studies have shown that about 50% of the manual effort result from cleaning the documents, which does not require any domain knowledge. A dedicated infrastructure allows for crowdsourcing the respective steps to online communities and thus further reduces the domain experts' effort.

The conversion technique described above has been deployed in several document digitization and conversion projects, each time with an accompanying field study. In total, biologists have converted over 5,000 pages of biosystematics literature in the course of the projects. The conversion technique has proved itself as a highly viable solution to the presented problem. Using an off-the-shelf XML editor the conversion takes about 30 minutes per document page on average; the tools presented in this work reduce this effort to little more than one minute per document page. Several laboratory experiments with pasta recipes have substantiated this finding under

controlled conditions. Along the way, these experiments have demonstrated that the presented conversion technique transfers to other domains with relative ease.

An infrastructure for crowdsourcing the manual effort that arises from cleaning of the documents has been developed. To deal with possible errors contributing users make when working on the crowdsourced correction tasks, this infrastructure incorporates several new generic data quality enforcement mechanisms that strongly mitigate the tradeoff between the expectable quality of the crowdsourcing results and the expectable throughput. In practical deployment, correcting the structure of pages in OCR output turned out to appeal to the target audience only to a very limited degree, unfortunately; it is simply not as appealing as classifying galaxies or searching for aliens. Stronger incentives than scores on a virtual ranking will be required to enable processing large number of documents via crowdsourcing, e.g. monetary ones. Once such incentives are in place, however, crowdsourcing will roughly halve the domain experts' effort because the steps that do require domain knowledge account for roughly half of the working time, regardless of other mechanisms in use.

## 1.4 Results

This work presents a semi-automated technique for converting digitized documents into a machine interpretable form, with domain experts ensuring data quality. It studies the process of conversion and optimizes it in multiple mutually orthogonal dimensions. As a result, manual has dropped from about 30 minutes to little more than one minute per document page. These values arise from long-term practice studies and have been substantiated in laboratory experiments. Crowdsourcing is expectable to halve the domain experts' effort yet again once sufficiently strong incentives are in place; whatever the latter will be, the cost will be far lower than paying a domain expert for the same work.

The lion's share of the research presented in this work has been in the context of biosystematics. However, only a small fraction of the findings is specific to that domain, and most of the conclusions easily transfer to other domains. Analogously, transferring the implemented tools should require only minor customizations.

## 1.5 Basic Assumptions

Though they are few, there are some basic assumptions underlying the techniques and results presented in this work, specifically ones reflecting users' motivation, capabilities / restrictions, and behavior:

1. Users are generally benevolent, not malicious or even colluding to defraud the system or compromise data. The rationale behind this assumption is that expert users have an inherent interest in the documents and their contained knowledge, as it benefits their own work.
2. Users can and do make occasional mistakes, though. This assumption acknowledges two problems in manually editing data: First, editing XML on character level is highly error prone, especially for a non-computer-science people. Second, strictly adhering to a complex data editing process is gene-



rally challenging, especially in the face of distractions. These two problems have to be addressed.

3. Having to do routine tasks repeatedly deters users and diminishes their motivation. This has to be addressed in the design of manual editing tasks.
4. Technical complexity scares non-computer-science people, like domain experts, and thus has to be hidden or encapsulated as far as possible.

The crowdsourcing approach (Contribution 5, cf. Chapter 10) departs from the first assumption, however, as general benevolence cannot be assumed in an anonymous online user community that has no specific relation to the documents they work on, the data contained in them, or even the domain the documents belong to. On the other hand, the other three assumptions are assumed as aggravated in an online user community, as they model the users' restrictions.



## 2 Preliminaries

This chapter introduces some preliminaries, technical basics, and definitions / notions required for understanding the rest of this work. In particular, this chapter covers (1) XML and the distinction between its data centric and its document centric flavor, (2) a definition of semantic markup, (3) XML validation techniques, (4) natural language processing details, including common models, learning techniques and error metrics, and (5) a basic overview of machine reasoning techniques.

### 2.1 XML

The eXtensible Markup Language (XML) [XML] is a semi-structured, text based data format. Logically, XML documents are trees; the leaf nodes contain the textual data<sup>1</sup>, while the inner nodes (also referred to as **elements**) represent the document structure, and thereby define the semantics of the leaf nodes in their sub tree. In textual representation, the inner nodes are denoted as pairs of **start** and **end tags**, the start tag including their **attribute** children; textual data remain as are (except for some escaping), denoted between the tags representing their ancestor nodes.

The start tag of an element has the general form denoted below; **name** is the element name, **attribute1** though **attributeN** are arbitrary, but mutually distinct attribute names, **value1** though **valueN** are arbitrary attribute values. The **angle brackets** delimit the tag as a whole; the **equal signs** separate attribute names from values; the **quotes** delimit attribute values. If the element a tag represents does not have any attributes, the tag simply contains the element name between the pair of angle brackets.

```
<name attribute1="value1" ... attributeN="valueN">
```

The end tag of an element has the form below; **name** is the name of the element it closes, matching the name given in the respective start tag. An end tag does not enclose any attributes. Again, the **angle brackets** delimit the tag; the leading **forward slash** marks it as an end tag.

```
</name>
```

If an element does not have any children except for attributes, it is represented as an empty tag, as shown below. The **forward slash** preceding the closing angle bracket indicates that the tag is empty. Otherwise, the structure is the same as for a start tag (as described above).

```
<name attribute1="value1" ... attributeN="valueN"/>
```

Depending on whether tags or textual content dominate a document, it's called more **data centric** or more **document centric**, respectively. Both flavors are discussed in respective sections below, followed by a summary discussion and a more formal definition of semantic markup.

---

<sup>1</sup> Comments and processing instructions are leaf nodes as well, but both are not described here, see [XML] for the details.

### 2.1.1 Data Centric XML

Data centric XML is what most commonly comes to mind as XML in general. It is highly structured, with relatively many inner nodes in comparison to text nodes, and consequently with a relatively deep tree nesting. The text data usually are the values of records, similar to those occurring in relational databases, with the addition that the values themselves can and often are subordinate records. On the other hand, flowing text data composed of whole natural language sentences are relatively few. Example 2.1 illustrates the notion of data centric XML.

```
<bookStore>
  <book>
    <title>My Life and Times</title>
    <author>
      <firstName>Paul</firstName>
      <lastName>McCartney</lastName>
    </author>
    <date>1998</date>
    <publisher>McMillin Publishing</publisher>
    <ISBN>1-56592-235-2</ISBN>
  </book>
  <book>
    <title>Adventures of a Reluctant Messiah</title>
    <author>
      <firstName>Richard</firstName>
      <lastName>Bach</lastName>
    </author>
    <date>1977</date>
    <publisher>Dell Publishing Co.</publisher>
    <ISBN>0-440-34319-4</ISBN>
  </book>
  <book>
    <title>The First and Last Freedom</title>
    <author>
      <firstName>J.</firstName>
      <lastName>Krishnamurti</lastName>
    </author>
    <date>1954</date>
    <publisher>Harper & Row</publisher>
    <ISBN>0-06-064831-7</ISBN>
  </book>
</bookStore>
```

#### Example 2.1: A data centric XML document

Example 2.1 describes a bookstore containing three books. Each book is a tuple with the attributes title, author with first and last name, date of publication, publisher, and ISBN (international standard book number), the latter being the book's identifier.

The example illustrates the capability of XML to express hierarchical structures, which do not map to a relational table in a straightforward way. Think of a case in which a book can have a theoretically unlimited number of authors: In the example above, there would simply be multiple author elements nested in an affected book

element; in a relational database, one would have to resort to multiple tables and foreign key relations to model such a case.

In some cases, the hierarchical structure can also be a drawback in comparison to flat relational tables, however. Think of one author having written multiple books: With relational tables and foreign keys, there would simply be multiple book records pointing to the same author record; in the example document, one would have to nest the author data in every single book<sup>2</sup>.

### 2.1.2 Document Centric XML

Document centric XML consists of less markup (tags) and more textual content than data centric XML. In this flavor, usually markup is embedded in natural language text to convey additional information on specific words in a machine understandable format. This additional information can be of multiple kinds, as we will see in the following. With regard to the structure of the XML tree, embedding tags in text means that quite often inner nodes have both text and element children.

Throughout the examples, we will use the following text passage, which might originate from a book on rock stars' attempts at eternalizing themselves:

```
Paul McCartney wrote his autobiography "My Life and Times"
in 1998. McMillin Publishing published the book the same
year (ISBN 1-56592-235-2).
```

**XHTML.** In XHTML [XHTML], for instance, the embedded tags carry layout information that tells web browsers how to display the document text. This layout information usually covers both the page structure and the font level styling of individual words or phrases<sup>3</sup>.

```
<html>
  <head>...</head>
  <body>
    <h1>Paul McCartney</h1>
    <p>
      Paul McCartney wrote his autobiography "<i>My Life
      and Times</i>" in 1998. McMillin Publishing published
      the book the same year (<b>ISBN 1-56592-235-2</b>).
    </p>
    <p>...</p>
  </body>
</html>
```

#### Example 2.2: An XHTML document

A browser displays the content of the body element of Example 2.2 as follows (tags referred to in bold for clarity): The piece of text inside the h1 element is rendered as a first level heading; the contents of the p elements are laid out as

---

<sup>2</sup> IDREFs can solve this redundancy problem, e.g. by giving the actual author data only on the first occurrence and referencing it on any subsequent one. But this would incur new problems, e.g. if the book containing the first occurrence of an author is deleted, and the author data with it, voiding all the references.

<sup>3</sup> Elements that provide functionality, e.g. script, form, or hyperlink elements, are not discussed here, as this would not contribute to illustrating the notion of document centric XML.

paragraphs. The `i` element embedded in the first paragraph causes the browser to render the phrase “My Life and Times” in italics; the `b` element causes a browser to render the ISBN in bold.

**Semantic Tags.** Just as layout information, markup embedded in a natural language text can also carry semantic information that enables machines to understand the actual meaning of natural language text. For instance, embedded XML tags can express the grammatical roles of individual phrases in the text, optionally with attributes providing normalized forms of the words to simplify machine interpretation, as shown in Example 2.3.1:

```
<paragraph>
  <sentence>
    <subject>Paul McCartney</subject>
    <predicate baseForm="write" tense="past">
      wrote</predicate> his
    <object>autobiography "My Life and Times"</object>
    <adverbial type="time" value="1998">
      in 1998</adverbial>.
  </sentence>
  <sentence>
    <subject>McMillin Publishing</subject>
    <predicate baseForm="publish" tense="past">
      published</predicate>
    <object>the book</object>
    <adverbial type="time">the same year</adverbial>
    (<attributive>ISBN 1-56592-235-2</attributive>)
  </sentence>
</paragraph>
```

**Example 2.3.1: A document centric XML document with semantic markup**

Additional tags and attributes can make more explicit the nature of the entities mentioned in the text, or implicit meanings of pronouns or synonymous references to entities mentioned earlier. For clarity, these parts are in bold in Example 2.3.2.

```
<paragraph>
  <sentence>
    <subject>
      <person>
        <firstName>Paul</firstName>
        <lastName>McCartney</lastName>
      </person>
    </subject>
    <predicate baseForm="write" tense="past">
      wrote</predicate> his
    <object>
      <workOfArt type="book" title="My Life and Times">
        autobiography "My Life and Times"</workOfArt>
    </object>
    <adverbial type="time" detailType="year" value="1998">
      in 1998</adverbial>.
  </sentence>
```

```

<sentence>
  <subject>
    <organization type="company">
      McMillin Publishing</organization>
    </subject>
    <predicate baseForm="publish" tense="past">
      published</predicate>
    <object>
      <coreference
        target="preceding::workOfArt[@type='book'] [1]">
        the book</coreference>
    </object>
    <adverbial type="time">
      <coreference
        target="preceding::adverbial[@type='time'] [1]">
        the same year</coreference>
    </adverbial>
    (<attributive target=" ../object" type="ISBN">
      ISBN 1-56592-235-2
    </attributive>).
  </sentence>
</paragraph>

```

### Example 2.3.2: The same XML document with additional semantic markup

Given all this semantic information in a machine interpretable form, it now is possible to extract actual entities and their attributes and relations, just as a human reader naturally does when reading the example text.

### 2.1.3 Summary & Discussion

The last two sections have introduced and illustrated the notions of data centric and document centric XML. However, Example 2.3.2, the last example for document centric markup, contains lots of tags already, making it look rather data centric. This illustrates that the boundary between data centric and document centric XML is somewhat fuzzy. Nevertheless, the concatenation of the text nodes in Example 2.3.2 still reads as natural language, despite all the markup, which does not hold for the data centric example.

In real-world applications, both flavors can well occur together, nested either way: The book elements in Example 2.1 might well contain a description element that gives a natural language description of the book, possibly marked with XHTML internally. On the other hand, the person element in Example 2.3.2 is marks in a very data centric fashion. The latter phenomenon occurs quite often in document centric XML, in particular when named entities are augmented with external information that is attached to them in a data centric way.

### 2.1.4 Semantic Markup

As already implied in the last section, the notion of **semantic markup** refers to document centric XML markup that makes the semantics of natural language text understandable to machines. Semantic markup is the basis for extracting the factual

data that the original author of the text described in a human-oriented way. Machines then can convert this data into fact-only data centric XML for more advanced applications to process further, e.g. reasoners (see Section 2.4). Enabling this extraction and advanced processing is the major reason for all efforts towards adding semantic markup to digitized legacy literature. As an example for data extraction, Example 2.3.2 implicitly specifies the following record, which is identical to the first book element in Example 2.1:

```
<book>
  <title>My Life and Times</title>
  <author>
    <firstName>Paul</firstName>
    <lastName>McCartney</lastName>
  </author>
  <date>1998</date>
  <publisher>McMillin Publishing</publisher>
  <ISBN>1-56592-235-2</ISBN>
</book>
```

**Example 2.4: Data centric record extracted from the document in Example 2.3.2**

Note that the notion of semantic markup covers the semantic details of a text, as shown above, as well as its logical structure, comprising chapters, sections, paragraphs, etc. The latter is required because the subject facts refer to may be rather far from the place where the text explicitly mentions that subject. For instance, the subject may be explicitly mentioned in a section heading only, and implicitly referred to by the remainder of the section. Without markup indicating the section boundaries, it may not be possible for machines to understand that reference in the general case, foiling extraction of the facts. Another example is figuring out the target of co-references, as shown in Example 2.3.2, which depends on paragraph boundaries.

Furthermore, not shown in Example 2.3.2, but implied in the introduction, semantic markup also includes disambiguation in a global context. To a human reader, it is intuitively clear that the person name “Paul McCartney” in the example text refers to that very famous person by the name of Sir James Paul McCartney, born June 18<sup>th</sup>, 1942 in Liverpool, England, UK, founding member of The Beatles, etc. For a machine, this intuition is by no means clear. On the other hand it would be catastrophically wrong to generally interpret the name “Paul McCartney” as to refer to this one famous person, as there may well be other persons with that name. Thus, to enable machines to make unambiguous interpretations, links, and conclusions, i.e., to really enable semantically meaningful processing, semantic markup also includes providing some sort of unique identifier with mentions of real-world entities.

Finally, all these considerations lead to the following definition of semantic markup:

**Definition 2.1:** Semantic markup is document centric XML embedded in natural language documents that makes explicit and machine interpretable both the logical document structure and semantic details of the text, and unambiguously identifies the real-world objects the text refers to.



## 2.2 XML Validation Techniques

XML validation is a means to test whether or not a given XML document meets specific conditions. Among others, this is useful for applications that read data from XML documents to make sure the input data is formatted in the expected way. There are several ways of expressing these conditions and testing a given XML document against them. One major distinction between individual techniques is whether they take a closed-world approach or an open-world approach. With the former, XML documents can only have the content (elements, attributes, text, etc) explicitly permitted. With the latter, XML documents can have any content as long as it does not violate the given constraints.

This section briefly introduces the most commonly used validation techniques, which later chapters frequently refer to. DTD is omitted, as it is mostly obsolete since XML Schema has been widely used.

### 2.2.1 XML Schema

**XML Schema** [XmlSchema] defines the structure, i.e., the permitted element names, attributes, and attribute values, as well as permitted element nestings and locations for textual content, by means of a context free grammar, taking a closed-world approach. XML Schemas can import other XML Schemas to make use of the content definitions they provide. In such a setting, each imported schema is identified by a namespace prefix, which is also included in element names to indicate which namespace they belong to, so to prevent ambiguities, e.g. in cases where multiple imported schemas define elements with the same name, but different content and semantics:

```
<namespace:name ...>
```

In many domains, there are schemas whose purpose is to provide a domain specific set of element names to be imported and reused by other schemas rather than defining structuring rules for entire XML documents. A good example of such a schema is Dublin Core [DublinCore], a widely used set of elements for marking document meta data such as author(s), title, date of publication, etc. Darwin Core [DarwinCore] is another example of a pure namespace schema, coming from the biology domain. An example for a schema that does define elements and structure, yet is intended to be imported in other schemas more than for standalone use is the Metadata Object Description Schema [MODS], another schema for marking document meta data; the explicit intention of the MODS is to a more structured approach than Dublin Core.

**Relax NG** [RelaxNG] is very similar to XML Schema in terms of approach, constructs, and expressiveness. It differs from XML Schema mainly in its syntactical representation. In the remainder of this work, all the comments and comparisons to XML Schema, especially in Related Work sections, also apply to Relax NG, even if the latter is not mentioned explicitly.

### 2.2.2 SchemaTron

**SchemaTron** [SchemaTron] is an XML validation technique that takes the open-world approach. It uses constraints formulated as XPath expressions [XPath] (called **rules**) to validate XML documents. Each rule stands by itself, so it is possible to target specific parts of the markup in isolation.

A SchemaTron schema is structured as follows: It contains one or more **rules**, with each rule validating a specific part of an XML document, the so-called **rule context**, which is denoted as an XPath expression. Each rule contains one or more **reports** or **assertions**, which perform the actual validity tests. A report consists of an **XPath predicate**, which refers to the context of the surrounding rule. If the predicate evaluates to true, the report outputs an error message. Assertions work the other way around; they output an error message if the predicate evaluates to false.

In principle, validation of an XML document against a SchemaTron rule works as follows: First, the validator evaluates the XPath query from the rule's context attribute against the XML document to select the elements to test, the so-called **context elements**. In Example 2.5, these are all **sentence** elements. Second, the validator evaluates the XPath predicates specified in the **test** attributes of the reports and assertions, for each of the context elements. If the predicate of a report evaluates to **true**, the validator outputs the report's error message, which is given as the textual content of the **report** element. If the predicate of an assertion evaluates to **false**, in turn, the validator outputs the assertion's error message, which is given as the textual content of the **assertion** element.

```
<rule context="sentence">
  <report test="matches(text(), '[a-z].+')">
    A sentence must not start with a lower case word.
  </report>
  <assert test="./subject">
    A sentence must have a subject.
  </assert>
</rule>
```

### Example 2.5: A SchemaTron rule that validates the markup of sentences

The rule in Example 2.5 checks the detail markup of sentences, as those marked in Example 2.3.x. In particular, the assertion ensures that every sentence has a subject marked. The report tests whether or not sentence boundaries are valid, i.e., it outputs an error message for all sentence elements whose first textual content starts with a lower case word. The latter check is very crude and by no means sufficient to detect all errors in practice; it is only meant to provide an intuitive example.

## 2.3 Natural Language Processing

The term Natural Language Processing (NLP) subsumes a multitude of techniques that automatically figure out the semantics of natural language text without the backing of semantic markup. Generally, NLP is not limited to written text, but also addresses spoken language. This work, however, focuses on written language, or, even more specifically, on printed text. In particular, this means text with capitalization and punctuation information, as opposed to speech transcript, for instance, which does not have either. On the other hand, digitized legacy documents may contain considerable OCR noise, so the text cannot be and is not assumed to be error free.

As NLP techniques can figure out the semantics of text, it is a powerful tool for generating semantic XML markup for previously plain text. This is the primary appli-

cation NLP will be used for throughout the rest of this work. The semantic XML markup contained in Example 2.3.2 will serve as an example throughout this section.

The rest of this section gives an overview of NLP, in particular of (1) specific NLP tasks that are particularly relevant to semantic markup generation, (2) the error metrics that are commonly used to assess the quality of NLP results, (3) hand crafted and statistical models commonly used in NLP components, (4) machine learning techniques used to train these models, and (5) available NLP implementations and toolkits.

### 2.3.1 Common NLP Tasks

This section gives an overview of NLP tasks that are particularly relevant to semantic markup. It is by no means intended to give a complete overview of NLP tasks in general, as there are many of them that go way beyond what is required for semantic markup, and just as many that deal with spoken language, and thus are not very relevant to this work.

**Named Entity Recognition** (NER) [Chinchor 1997] refers to the identification of references to real-world objects in natural language text. Classical targets of NER are, as already implied in the introduction, names of persons, organizations, and locations, dates, monetary amounts, and percentages. In semantic markup, however, other categories may be relevant as well, e.g. works of art, as implied in Example 2.3, or domain specific named entities, like the scientific names of living species (e.g. *Drosophila melanogaster*) in biological texts. For instance, in Example 2.3.2, the tags marking persons, organizations, and works of art might be the result of NER. In general, the NER problem can be as twofold: The first partial problem is to identify sequences of words that are references to names entities, the second one is to identify for each such word sequence the category of named entity it refers to.

**Sentence Breaking** [Reynar 1997], also known as **Sentence Boundary Disambiguation**, is the process of identifying sentence boundaries in a given chunk of text. The tags marking sentence boundaries in Example 2.3, for instance, might be the result of sentence breaking.

**Part-Of-Speech Tagging** (POS-Tagging) [Garside 1987] is the process of classifying each word in a given text as to what grammatical category of word it belongs. Common categories are noun, verb, adjective, adverb, determiner, preposition, pronoun, etc. While POS tags themselves are not very useful for semantic markup, they are part of the basis of many other tasks, especially chunking and parsing (see below), and thus shall not be omitted here.

**Full Parsing** [Koskenniemi 1990] names the full, in-depth grammatical analysis of a given sentence. In particular, this includes chunking a given sentence into noun, verb, and adverbial phrases, and then identifying the role of each phrase. For instance, in Example 2.3, the tags identifying the subject, predicate, and object as well as adverbial phrases might be the result of full parsing. The identification of the individual phrases themselves is often referred to as **Shallow Parsing**, the exclusive identification of noun phrases as **Noun Phrase Chunking (NP-Chunking)**.

**Coreference Resolution** [Sidner 1981] is the process of determining the actual entity any given pronoun or referential phrase refers to. In Example 2.3.2, for instance, this would be figuring out the actual year the phrase “the same year”, and which actual book phrase “the book” refers to, respectively. As such, the `target`

attributes of the `coreference` tags in Example 2.3.2 might be the result of `coreference` resolution.

### 2.3.2 Error Metrics

This section introduces common error metrics used in NLP and other areas. These metrics will be frequently used throughout the rest of this work

Let us introduce the metrics with an example for illustration: Assume an NER algorithm processing a given document. Then, there are two types of errors that can occur: (1) an actual named entity is not recognized, a so-called **false negative** or **miss**, or (2) a sequence of words that is not an actual named entity is recognized as one anyway, a so-called **false positive** or **false hit**. Furthermore, there are two correct outcomes: (1) an actual named entity is recognized as such, a so-called **true positive** or **hit**, or (2) a sequence of words that is not a named entity is not recognized as such, a so-called **true negative**.

Based on true and false positives and negatives, there are two basic metrics:

**Precision** [Olson 2008] is the fraction of true positives in all positives:

$$\text{precision} := \frac{|\{\text{true positives}\}|}{|\{\text{true positives}\}| + |\{\text{false positives}\}|}$$

**Recall** [Olson 2008] is the fraction of all positives that have been recognized:

$$\text{recall} := \frac{|\{\text{true positives}\}|}{|\{\text{true positives}\}| + |\{\text{false negatives}\}|}$$

Please observe that true negatives are not part of any one of these two definitions. Based on precision and recall, there are two combined metrics that are commonly used for assessing the overall performance of an algorithm:

**Accuracy** [Olson 2008] is the product of precision and recall:

$$\text{accuracy} := \text{precision} \cdot \text{recall}$$

The **f-score** [Olson 2008] is the harmonic mean of precision and recall:

$$\text{f-score} := \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Getting back to the example from above, let us assume there are 100 named entities in a given document. Let us further assume an NER algorithm correctly identifies 97 of them, missing 3, and mistakes another 5 word sequences for named entities that are actually none. This gives us the following:

Metric	Formula	Decimal	Percentage
Precision	$97 / (97+5)$	0.951	95.1 %
Recall	$97 / (97 + 3)$	0.970	97.0 %
Accuracy	$0.951 * 0.97$	0.922	92.2 %
F-Score	$(2 * 0.951 * 0.97) / (0.951 + 0.97)$	0.960	96.0 %

**Table 2.1: Error metrics example**

### 2.3.3 Common NLP Decision Models

This section presents knowledge models commonly used in NLP algorithms and components as the basis for decisions. This section only covers the models themselves and how they work. Methods for creating models from training data via machine learning are discussed in the next section.

Handcrafted **Gazetteer Lists & Rules** [Mikheev 1999] are among the earliest approaches at NLP tasks like NER. They try to identify named entities based on capitalization, lists of frequent names of the individual categories (see Section 2.3.1), and morphological and grammatical rules. The latter are often language specific (usually to English, as English text was used in the first competitions, e.g. MUC-1), exploiting prepositions, common first names, etc. These systems are relatively expensive to create, as their creation requires considerable linguistic expertise. On the other hand, they are fast at runtime. Today, handcrafted gazetteer list and rule systems have been mostly replaced by statistical ones (see below) trained through machine learning (see next section) in general NLP. However, (partially) handcrafted NLP systems continue to exist in domain specific applications for various reasons.

**Decision Trees** [Magerman 1995] represent a hierarchy of usually binary decision rules. The inner nodes of the tree represent one binary decision each, and the leaves are the actual classifications a series of decisions leads to. Decision trees need not be balanced and often are not. Primarily used in operations research and data mining, where they are often generated by dedicated learning algorithms [Bahl1989], decision trees in their purest sense are relatively rare in NLP. Nevertheless, hierarchical systems of rules can be seen as handcrafted, hard coded linear decision trees. In NLP, this flavor of decision trees can be found in the classification part of NER, among others.

A **Hidden Markov Model** (HMM) [Rabiner 1986] is a statistical model that assumes a sequence of events to be produced by a Markov process, but only the output of that process to be visible, i.e. the events, not its states or state transitions. In the NLP domain, the events usually are the individual words of the given text together with their labels, e.g. POS tags. In order to find the labels for the words in a given piece of text, NLP applications compute the most likely sequence of events (pairs of words and labels) that produces the given word sequence and from that sequence derive the labels. HMMs are usually created through supervised learning (see next section). Their application ranges from NER to POS tagging to chunking to shallow and full parsing.

**Conditional Random Fields** (CRFs) [Lafferty 2001] are a generalization of HMMs (see above). They are more computationally expensive to train, often requiring numerical methods, but on the other hand yield somewhat better results. In NLP, CRFs are usually used for NER and shallow parsing or chunking.

**Support Vector Machines** (SVMs) [Cristianini 2000] are often used for (usually binary) classification tasks. They model individual inputs as points in a high dimensional vector space. A hyperplane divides this vector space into two distinct regions, each of which corresponds to one of the two classes. When classifying a previously unseen input, the SVM computes the vector representation of that input, check on which side of the hyperplane that vector is located, and outputs a corresponding classification. For higher-than-binary classification problems, usually several binary SVMs are combined [Isozaki2002]. SVMs are often generated by

supervised or active learning (see next section). In NLP, they often serve for the classification part in NER, but also for other tasks.

**Neural Networks** [Hertz 1990] are inspired by the human nervous system, based on the desire to duplicate the way a human being understands and solves problems in a computer. Neural networks are trained through either of supervised, unsupervised and reinforcement learning (see below). A very specific property of neural networks is the ability to constantly learn when reinforcement learning is used. However, neural networks are highly expensive to train in terms of required training data, and computationally highly expensive to use. In NLP, neural networks are mainly used on the sentence level and below [Collobert 2008], namely for parsing, chunking, and classification tasks.

#### 2.3.4 Machine Learning Techniques

In general, machine learning [Michalski 1983] subsumes algorithms that approximate an unknown classification function by generalizing from given examples, and then apply this function to classify so far unseen inputs. This section gives an overview of machine learning techniques that are commonly used for training statistical models for NLP applications.

**Supervised Learning** [Vapnik 2000] is machine learning from labeled training data. It can be used for training any of the statistical models described in the previous section. An advantage of supervised learning is that the trainer has tight control of what is learned. A major drawback is the labeled training data supervised learning requires, which is often very expensive or even impossible to obtain in all but the most standard NLP task on the most standard sorts of texts.

**Semi-supervised Learning** [Oneill 1978] overcomes the training data bottleneck by using only partially labeled data for training. In particular, a small amount of labeled training data is supplemented with a large amount of unlabeled training data. Two special kinds of semi-supervised learning are worth mentioning:

- **Active Learning** [Lewis 1994, Cohn 1994] involves a human user whom the learning component can ask to provide labels selected inputs. This technique is frequently used for training SVM based classifiers; the inputs that the SVM presents to the user for labeling are the ones whose label provides the highest information gain, usually the ones closest to the dividing hyperplane.
- **Co-Training** [Blum 1998] combines two classifiers that operate on two different (ideally, conditionally independent) subsets of features of the input data. First, both are trained on the available labeled training data. Then, each classifier trains the other with previously unlabeled examples that it has classified with high confidence.

A major advantage of semi-supervised learning is the drastically reduced amount of labeled training data required, in comparison to supervised learning. A major drawback is that in domain specific NLP tasks the required large amount of supplementary unlabeled training data may not be available. Even if large amount of raw OCR output are available, this can be assumed to be little helpful, as training results from data that contains considerable OCR noise are more than doubtful to be of any use.

**Unsupervised Learning** [Duda 2001] completely overcomes the need for labeled training data. Its primary application area is data mining, in particular clustering. A specific kind of unsupervised learning used in NLP is bootstrapping, which, for

instance, is able to train rule or HMM-based NER components based on very large amounts of unlabeled training data [Cucerzan 1999, Niu 2003]. Bootstrapping is not completely unsupervised, however, as it requires “a short list (order of one hundred) of unambiguous examples (seeds)” [Cucerzan 1999], which is equivalent to all occurrences of the seeds being labeled. Nevertheless, this is different from semi-supervised learning in that it uses one corpus of partially labeled training data instead of two distinct corpora of labeled and unlabeled training data, respectively. A major disadvantage of bootstrapping in domain specific settings is that the required large amounts of training data may not be available, at least not in the required quality (see above). In addition, good seeds are crucial, but may be hard to identify in such a setting.

**Reinforcement Learning** [Kaelbling 1996] is often used for training neural networks (see previous section) incrementally as it interacts with an environment. For each decision the neural network makes based on the observed state of its environment, it receives a reward or penalty (negative reward) from the environment in return. Based on this, reinforcement learning algorithms adjust the decision process with the goal of maximizing the overall reward. Due to its incremental nature, reinforcement learning does not distinguish a training phase from a test or application phase. As this distinction is common in evaluations for classical NLP tasks, reinforcement learning has not been used in this domain so far. This to happen would require NLP evaluations to provide rewards for correct classifications and penalties for incorrect ones during a run on the test set, which has yet to happen.

**Online Learning** [Freund 1999] is a common approach to incrementally training specific kinds of neural networks (perceptrons [Rosenblatt 1958] and winnows [Littlestone 1988], in particular) that classify a series of inputs one at a time, with the actual label of each input becoming available before processing the next input. Similar to reinforcement learning (see above), online learning works incrementally without distinguishing a training phase from a test or application phase. Only recently, perceptrons have been used successfully in NLP, namely for POS tagging and NP chunking [Collins 2002].

## 2.4 Machine Reasoning

A machine reasoner is a software component that infers new facts as logical consequences from a set of asserted facts, also referred to as axioms. The inference rules are commonly specified by means of an ontology language (e.g. OWL [OWL]), and often a description language. Many reasoners use first-order predicate logic and infer facts by forward chaining [Bledsoe 1973] and backward chaining [Newell 1959]. The former derives new facts from the given ones until the sought goal fact(s) are reached. The latter starts from the sought goal fact(s) and tries to find given facts that support them. Reasoners can also identify inconsistencies in a set of given facts.

A common input format for reasoners is RDF [RDF], with one of the various dialects of OWL as a constraint language. The common representation of RDF in semantic web applications is RDF/XML. Example 2.6 shows the RDF/XML representation of the data from Example 2.4, with `rdf` being the XML namespace for RDF, and `dc` being the one for DublinCore (see Section 2.2.1). In this format, a reasoner can add the data to its fact base.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="uri:isbn:1-56592-235-2">
    <dc:title>My Life and Times</dc:title>
    <dc:creator>Paul McCartney</dc:creator>
    <dc:publisher>McMillin Publishing</dc:publisher>
    <dc:date>McMillin Publishing</dc:date>
  </rdf:Description>
</rdf:RDF>
```

**Example 2.6: Data from Example 2.4 represented in RDF/XML**



### 3 Basic Assumptions & Requirements

This chapter introduces the basic assumption this work is based on, as well as the basis requirements. The assumptions are more concerned with the limitations of existing technology, its human users, and digitized legacy documents that have to be addressed than with favorable conditions that simplify the task.

#### 3.1 Assumptions Regarding Existing Algorithms & Technology

This section covers general technological limitations of NLP, not limitations of specific existing work; the latter is covered in the chapter on Related Work.

**Assumption T1.** Despite all research, NLP algorithms will never work 100% accurate even on favorable input in the general case. Therefore, the output of all NLP algorithms has to be considered potentially erroneous.

The rationale behind Assumption T1 is that the complexity and ambiguity of natural language *semantics* is far beyond what computers are able to process deterministically: Think of the famous “Time flies like an arrow, but fruit flies like a banana” example. This simple sentence highly intuitively illustrates both the semantic homonymy of natural language and the great deal of intuition the human brain uses in making sense of it anyways. Even whether or not natural language *syntax* is context-free and thus processible in polynomial time is highly controversial among language theorists [Shieber 1985], with a tendency towards non-context-freeness.

The best results for common NLP tasks furnish additional proof: For NER, the highest f-scores reported for the MUC-7 NER task [Chinchor 1997] are around 97% , which corresponds to an accuracy around 95% at best. The best published results for NP-Chunking are in equal echelons [Ngai 2000]. For POS-Tagging, the highest accuracy achieved so far is 97% [Brants 2000].

Most of the systems that achieved these results had been trained not only for one specific sort of text (usually newswire), but even for a rather narrow genre, with results worsening drastically when tested even on text from a related genre [Chieu2002]. Even though domain specific digitization and knowledge extraction efforts can be considered equally genre-specific, the task is harder in the latter. This is due to the wide diversity of language, layout, and typesetting styles found across legacy documents even from narrow domains, e.g., the ant fauna of Madagascar, not to mention documents written in different languages.

**Assumption T2.** NLP results worsen if the input data contains errors; i.e. errors propagate in multi-step processes.

Assumption T2 puts the results mentioned above into the context of multi-step processes, which apply multiple NLP techniques sequentially, each one possibly building on the results of the previous ones. Simple examples of such processes are NER and NP-Chunking with POS tags as evidence, the latter being generated by a

POS-Tagger before NER or chunking, respectively. At least for NP-Chunking, the best reported results – f-score about 95% – have been achieved with error-free POS tags. Two common examples of more complex processes are parsing and information extraction, which both often involve POS-Tagging, Chunking, and grammatical analysis, and in the case of information extraction also co-reference resolution. The best reported results for these two tasks are way worse than the ones for the individual parts, around 88% for parsing [Li 2001], and around 72% for information extraction [Marsh 1998]. This clearly shows that the results of later steps suffer from errors in the results of earlier ones.

### 3.2 Assumptions Regarding Human Users

This section covers the assumptions regarding the attitude, motivation, and limitations of human users who contribute to digitization and knowledge extraction efforts.

**Assumption U1.** Users are generally benevolent, not malicious or even colluding to purposely compromise data or break software.

Assumption U1 trivially holds for domain experts who extract knowledge from documents that belong to their own domain. This is because their motivation usually is to make that very extracted knowledge available for themselves and their peers to benefit from machine processing it. Consequently, domain experts would only harm themselves by purposely introducing errors in their data. On the other hand, Assumption U1 does not necessarily hold – and will not be assumed to hold – for contributors who have no direct interest in the data, but contribute for other reasons. Errors users make in violation of Assumption U1 are referred to as **cheating errors**.

**Assumption U2.** Technical complexity scares users with low computer knowledge, e.g. domain experts.

Assumption U2 accounts for the natural fear of unknown complex things that human beings tend to have when faced with technology they are not proficient in. In particular, domain experts cannot be assumed to have profound knowledge of XML or XML Schema, let alone NLP, using NLP tools from the command line with complex parameter settings, or machine learning. And having to learn all these things before being able to participate in a digitization and knowledge extraction effort puts up a high psychological barrier.

**Assumption U3.** Users can and do make occasional mistakes.

Assumption U3 accounts for the natural sloppiness of human beings, and with the fact that focus can fade over time, at least temporarily. Thus, supplementing Assumption U2, even with technology or GUIs users are familiar with, occasional mistakes cannot be ruled out; why else do we need spell checking in text processors? Errors users make as a consequence of Assumption U3 are referred to as **accidental errors**.

**Assumption U4.** Having to do routine tasks repeatedly deters users degrades their motivation.

Assumption U4 accounts for the natural reaction of humans to cumbersome repetitive tasks, namely that a user's motivation wears off when he is faced with the same low-level task over and over again.

**Assumption U5.** Users want to work when they have the time to do so.

Assumption U5 accounts for the fact that domain experts are often academics, who often have a rather busy schedule. This means that there is a good chance that they want to work on the road, e.g. on planes or trains, etc.

### 3.3 Assumptions Regarding Data

This section covers the assumptions regarding the sort and quality of both input and output data of semantic markup efforts. The assumptions about the input data are more concerned with complications and limitations to deal with than with favorable conditions. On the other hand, the ones regarding the output data specify the requirements to meet for semantic markup efforts to be successful.

**Assumption I1.** Digitized legacy documents are likely to contain OCR noise.

The rationale behind Assumption I1 is to explicitly state that OCR, as an application of artificial intelligence, is subject to the same limitations as specified for NLP technology in Assumption T1. The distinction is because the latter is a topic of this work, while OCR is not, and therefore the limitations of its output are to be treated as a given complication, not as a research challenge.

In addition, OCR is more challenging on legacy data and thus likely to produce noisier results, for two reasons: (a) due to sometimes inferior quality of both print (e.g. uneven distribution of ink, which may result in incomplete recognition) and paper (e.g. small stains, which are easily mistaken for punctuation), and (b) due to the traces the centuries left on the printed documents, like faded ink, or even marks and notes people wrote on the pages.

**Assumption I2.** The text of digitized legacy documents may not be in its logical reading order, paragraphs and sentences may be interrupted, and some phrases may not be part of the original text at all.

Assumption I2 accounts for the fact that digitized legacy documents are in print layout. This means (a) that the text may not be in its logical reading order, as some parts are positioned due to layout considerations, e.g. footnotes next to page boundaries. Worse, the logical paragraphs and sentences may flow across page breaks (possibly with footnotes) or around figures and their respective captions. It further means (b) that the documents likely contain parts that do not actually belong to the original text, but were added during print, e.g. page headers and page numbers. Print layout finally means (c) that words may be hyphenated, possibly right where the text flows across page breaks or figure captions.

**Assumption I3.** Digitized legacy documents do contain punctuation and capitalization information, even though it may be noisy.

Punctuation and capitalization information is valuable evidence in many NLP tasks, and its absence complicates matters considerably. Assumption I3 makes explicit that this valuable information is assumed to be available, even though not completely reliable as a consequence of Assumption I1. This is not too much of a restriction of the generality of this work, however, as the subject of research is digitized printed text, for which Assumption I3 almost trivially holds. And even if it does not hold, this merely complicates individual NLP tasks, but does not otherwise affect the overall effort.

**Requirement O1.** Semantic markup has to be highly accurate to be meaningful and useful.

The rationale behind Requirement O1 is that the facts that semantic markup makes explicit and machine-understandable have to be exactly the ones that the original author described in textual form for a human audience. Otherwise, especially the results of machine reasoning (see Section 2.4) are likely to turn out arbitrary: In a derivation chain of, say, 20 facts, one wrong fact, i.e., one not supported in the original text, is sufficient for the overall derivation to be arbitrary or outright wrong. But errors are undesirable in less sophisticated applications as well: Suppose an application that allows users to browse a document collection using links that are generated from semantic markup: The links would outright emphasize potential errors for users to see, beside leading to arbitrary content, which may both alienate users.

Especially in the digitization of scientific documents, it is crucial for Requirement O1 to hold: It is desirable to utilize the extracted facts in fact-based systems that support, for instance, surgery teams by suggesting adequate countermeasures to take in order to save a patient who shows a specific combination of symptoms. Likewise, such systems might support disaster response teams, e.g. by proposing reactions to specific chemical hazards in a specific environment, or environmental studies, etc.

Particularly in scientific applications, a further problem that arises from violations of Requirement O1 is that facts from different documents may be combined in incorrect ways, which can be rather debilitating because virtually all publications build upon or distinguish themselves from prior ones.

## 4 Example Scenarios

This chapter introduces two example scenarios, which will serve as an illustration for the further considerations. Each scenario has a specific setting, specific source documents, and a specific target markup. The latter describes which semantic markup was to be generated.

The first scenario is the generation of semantic markup for digitized legacy publications from the biosystematics domain. This scenario arises from the fact that the research presented in this thesis was conducted within a joint computer science and biodiversity project. Several of its sub-projects were the subjects of the field studies presented in later chapters. It will be the main example.

The second scenario is the generation of semantic markup for digitized cooking books, pasta recipes in particular. This scenario was created for controlled laboratory experiments whose participants lacked the biological domain expertise required in the first scenario. The choice fell on pasta recipes because they are commonly understandable, yet rich in structure and details, and adding semantic markup is intuitively beneficial, e.g. for auto-generating grocery lists.

### 4.1 The Biosystematics Scenario

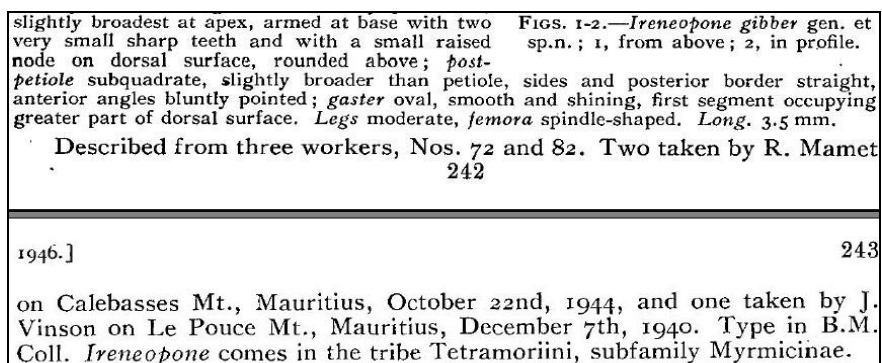
The field of biosystematics deals with the description of the species of the world, and with their organization and categorization into the taxonomic hierarchy, which, in bottom-up order, consists of species, genera, families, orders, classes, and kingdoms, to name only the major ones.

Biosystematics started in the 18<sup>th</sup> century, when Swedish natural scientist Carolus Linneaus first published his seminal books *Systema Naturae* (Latin for *The System of Nature*) [Linneaus 1735] and *Species Plantarum* (Latin for *The Species of Plants*) [Linneaus 1753]. To this day, these two books are the basis of systematic zoology and systematic botany, respectively, and the whole network of bibliographic references in either discipline is more or less rooted at them. Both sciences experienced an enormous boost in the 19<sup>th</sup> and early 20<sup>th</sup> century, when explorers from the colonial powers of the time traveled the world and brought back specimens of animals and plants from around the globe. In the end of the 20<sup>th</sup> century, genetic analysis revolutionized the methodology applied in biosystematics and brought up enormous amounts new data.

As ecology becomes more and more important, especially as a means of assessing the impact that mankind has on nature, there is a rapidly increasing interest in all the invaluable knowledge that explorers gathered on the state of nature 100 years ago, among others as a basis for comparison to the state of today. Presently, this knowledge is mostly stored in libraries as ink on paper. Only in the last decade, efforts like the Biodiversity Heritage Library (BHL) [BHL] or AntBase [Agosti 2005] have started digitizing and OCR-ing the documents to make them available in digital form. However, they do now go beyond OCR at this time.

Figure 4.1 shows a small excerpt from one of the digitized documents, namely from [Donisthorpe 1946]. The figure contains several examples of the complications that arise from print layout:

1. A page break (the gray bar) is embedded in a paragraph, splitting the sentence “Described from three workers, Nos. 72 and 82. Two taken by R. Mamet on Calebasses Mt., Mauritius, October 22nd, 1944, [...]” right between the words “Mamet” and “on”.
2. This page break comes with the page number “242” above and the page header “1946.] 243” below it, the latter comprising the year of publication and the next page number.
3. In the top right corner sits the caption of a figure, namely “FIGS. 1 - 2. - *Ireneopone gibber* gen. et sp. n.; 1, from above; 2, in profile.”, with the main text flowing right around it.
4. To the lower left of the caption, the word “postpetiole”<sup>4</sup> is hyphenated as “post-<linebreak>petiole”.



**Figure 4.1: Excerpt from scanned document**

Clearly, the page break (1) and its surrounding layout artifacts (2) have to be removed in order to restore the logical text flow. Furthermore, a quite common OCR error with captions that have the main text flowing around them (3) is that they are merged in a line together with the main text, alienating both, as shown in Example 4.1 in what might be HTML formatted OCR output (words in middle of line omitted for clarity). Finally, the morphological term “postpetiole” (4) has to be de-hyphenated.

```
<p>[...]<br>
  slightly [...] with two      FIGS. 1-2 [...] gen. et<br>
  very small [...] raised     sp.n.; [...] in profile.<br>
  node on dorsal [...] post-</p>
<p>petiole subquadrate, [...] border straight,<br>
  [...]</p>
```

**Example 4.1: Main text and caption mixed up (hypothetical)**

<sup>4</sup> The postpetiole (roughly) is the rearmost of two parts that make up the slender stick connecting the main body and the abdomen. A more accurate description would have to contain a lot more zoological terms and thus would contribute little to common understandability.

Example 4.2 shows the whole excerpt after OCR, augmented with related text further up the page is required for explanations below. The error shown in Example 4.1 is not repeated, the caption being embedded between two main text paragraphs. This is problematic as well, as the caption is located in between the two parts of the hyphenated word “*postpetiole*”.

```

<p>Ireneopone gibber, sp. n.</p>
<p>Reddish brown, shining, gaster darker.</p>
<p>Head subrectangular, longer than broad, posterior angles
  rounded, posterior border and<br>
  cheeks straight, covered with small, shallow, scattered
  punctures which bear a micro-<br>
  scopical decumbent yellow hair; mandibles power-<br>
  [...]<br>
  slightly broadest at apex, armed at base with two<br>
  very small sharp teeth and with a small raised<br>
  node on dorsal surface, rounded above; post-</p>
<p>FIGS. 1 - 2. - Ireneopone gibber gen. et<br>
  sp. n.; 1, from above; 2, in profile.</p>
<p>petiole subquadrate, slightly broader than petiole,
  sides and posterior border straight,<br>
  anterior angles bluntly pointed; gaster oval, smooth and
  shining, first segment occupying<br>
  greater part of dorsal surface. Legs moderate, femora
  spindle-shaped. Long. 3.5 mm.</p>
<p>Described from three workers, Nos. 72 and 82. Two taken
  by R. Mamet</p>
<p>242</p>
<hr>
<p>1946.] 243</p>
<p>on Calebasses Mt., Mauritius, October 22nd, 1944, and
  one taken by J.<br>
  Vinson on Le Pouce Mt., Mauritius, December 7th, 1940.
  Type in B. M.<br>
  Coll. Ireneopone comes in the tribe Tetramoriini,
  subfamily Myrmicinae.</p>

```

**Example 4.2: OCR output for Figure 4.1 and page above it**

To extract the actual knowledge and data from the OCR output is the subject of smaller projects. One of them, joint DFG/NSF grant *BIB47 MGuv 01-01 (Development of New Digital Library Applications in the Context of a basic Ontology for Biosystematics Information Using the Literature of Entomology (Ants))*, and its NPO successor Plazi [Plazi] are the background of much of the research presented in this work. In particular, several small (sub-) projects have generated semantic markup for distinctive bodies of literature from the ant domain. The research results were put to use in these projects, and evaluated by means of accompanying field studies. In total, these projects generated semantic markup for some 5.000 document pages, which are in six different languages, namely English, French, German, Portuguese, Italian, and Latin.

The semantic markup generated in the markup projects covers both the documents' structure and the semantic details. In particular, the structure comprises taxonomic treatments, their sub sections, and paragraphs. **Treatments** are a specific kind of section that is specific to biosystematics: The subject of a taxonomic treatment is exactly one taxon and thus a specific entity, most commonly a species, but also lower ones, e.g. subspecies, or higher ones, e.g. genera or families. All data given in a treatment refers to this entity. The individual subsections of a treatment cover one aspect of the taxon each: The taxon is given in the **nomenclature** subsection, usually right at beginning of the treatment. There may further be a **morphological description** of the taxon, and where individual specimens have been collected, when and by whom, see Table 4.1 for a comprehensive overview. The semantic details comprise the names of taxa and locations where specimens have been collected. Furthermore, taxon names are disambiguated with their Life Science Identifiers (so-called LSIDs, [Brazma 2006]), a special kind of URIs for taxon names, and location names are geo-referenced for the same purpose.

Type	Primary Content
Nomenclature	The taxon the treatment refers to, plus other nomenclatorial acts, e.g. the synonymization of two taxonomic names that were found to actually refer to the same taxon
Description	Morphological description of a taxon
Materials Examined	Locations where the specimens described in Description have been collected, when they were collected, and by whom
Biology	Behavior of taxon, interactions with environment, nesting preferences
Diagnosis	Explanation of differences of taxon to related taxons, plus identifying features.
Discussion	Explanation why these differences justify making them a taxon of their own
Distribution	Summary and distribution pattern of a taxon (e.g., 'throughout southern Africa'), based on material presented in Materials Examined
Etymology	Etymology (origin) of the taxon name

**Table 4.1: Types of subsections in treatments**

Example 4.3 shows the same text as Example 4.2, after semantic markup. The `tax` namespace points to TaxonX [Catapano 2006], a dedicated XML Schema for biosystematics documents; the `dwc` namespace points to Darwin Core [DarwinCore], an XML Schema that provides elements for semantic details used in biodiversity. In particular, the following has happened on the way from Example 4.2 to Example 4.3:

1. The caption that was embedded in the logical main text paragraph has been relocated so it does not interrupt the main text any more, in particular not the word "*postpetiole*", facilitating its de-hyphenation.
2. The layout artifacts (page number and page header) have been removed.



```

<tax:treatment xmlns:tax="..." xmlns:dwc="...">
  <tax:nomenclature>
    <tax:name>
      <tax:xid source="HNS" identifier="urn:lsid:...:29577"/>
      <tax:xmldata>
        <dwc:Genus>Ireneopone</dwc:Genus>
        <dwc:Species>gibber</dwc:Species>
      </tax:xmldata>
      Ireneopone gibber
    </tax:name>
    <tax:status>sp. n.</tax:status>
  </tax:nomenclature>
  <tax:div type="description">
    <tax:p>Reddish brown, shining, gaster darker.</tax:p>
    <tax:p>Head subrectangular, longer than broad, [...]
    [...] femora spindle-shaped. Long. 3.5 mm.</tax:p>
    <tax:p>FIGS. 1 - 2. - Ireneopone gibber gen. et sp. n.;
    1, from above; 2, in profile.</tax:p>
  </tax:div>
  <tax:div type="materials_examined">
    <tax:p>Described from three workers, Nos. 72 and 82.
    Two taken by R. Mamet <tax:pb n="243"> on
    <tax:locatity>
      <tax:xmldata>
        <dwc:Longitude>57.57056</dwc:Longitude>
        <dwc:Latitude>-20.13361</dwc:Latitude>
      </tax:xmldata>
      Calebasses Mt.
    </tax:locatity>
    , Mauritius, October 22nd, 1944, and one taken by J.
    Vinson on
    <tax:locatity>
      <tax:xmldata>
        <dwc:Longitude>57.52222</dwc:Longitude>
        <dwc:Latitude>-20.195</dwc:Latitude>
      </tax:xmldata>
      Le Pouce Mt.
    </tax:locatity>
    , Mauritius, December 7th, 1940.
    [...]</tax:p>
  </tax:div>
</tax:treatment>

```

**Example 4.3: OCR output from Example 4.2 after semantic markup**

3. The paragraphs are marked for logical document structure now, no more for layout consideration. The `tax:pb` element marks where a page break was, for bibliographic use.

4. All other formerly hyphenated words have been de-hyphenated as well.
5. The taxon name "*Ireneopone gibber*" has been marked, parsed into genus "*Ireneopone*" and species "*gibber*", and has had its LSID "*urn:lsid:....:29577*" attached to it. The latter are imported from a respective external authority.
6. The two locations "*Calebasses Mt.*" and "*Le Pouce Mt.*" have been marked and geo-referenced with their respective longitudes and latitudes. The latter have are imported from an external geo gazetteer<sup>5</sup>.
7. The whole part of the underlying document that refers to the taxon "*Ireneopone gibber*" has been marked as one treatment.
8. The treatment is structured into nomenclature (specifies the taxon the treatment refers to), description (morphological description of the taxon), and materials examined (locations where the specimens that the description is based on have been collected, when, by whom, etc., only location markup shown for clarity).

This markup now allows extracting, among others, pairs of a taxon name and a location where specimens of the taxon have been collected, so-called occurrence records. A set of occurrence records for a given taxon, in turn, facilitates plotting a distribution map of that taxon. The latter can also be time-dependent if dates are given and marked (not shown in Example 4.3 for clarity), based on when the specimens have been collected. This, finally, can indicate how the dispersal of a taxon changed over time, which can form the basis for ecological conclusions.

## 4.2 The Pasta Recipe Scenario

Verifying the results from the field studies under laboratory conditions by means of controlled experiments [Tichy 2000] required another application scenario. This is because thoroughly understanding biosystematics documents even in their basic concepts requires considerable domain knowledge, which drastically restricts the pool of potential participants and heavily influences experimental results. The major design goals of the alternative scenario were the following: (a) common understandability of the general subject and intuitive benefit from semantic markup, and (b) documents with a structure and semantic details that somewhat resemble their counterparts in biosystematics documents.

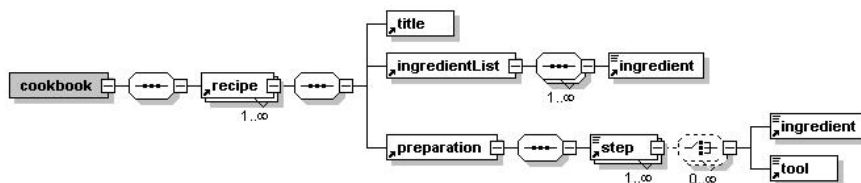


Figure 4.2: The XML Schema for the pasta recipes

<sup>5</sup> Note that unlike LSIDs, geographical coordinates naturally come with a certain numerical accuracy and thus bear some degree of fuzziness. They consequently do not fully qualify as unique identifiers, but they are the closest to the latter available for disambiguating locations. Therefore, they are used in the role of respective identifiers, which works well in practice.

The choice fell on pasta recipes. They are commonly understandable on the one hand, but on the other hand resemble biosystematics publications to some degree. In particular, the structure comprises recipes that refer to an individual dish, and sub sections like ingredient list, step-by-step preparation, or wine recommendations. Semantic details are ingredients and cooking tools. Figure 4.2 displays the XML Schema against which the documents have to be valid when the markup process is complete.

In addition, cooking is a scenario that many people do not only understand, but also have a relation to. This increases the pool of possible participants for the experiments, and it facilitates some intuitive and commonly understandable examples for the benefit of semantically marked documents, like the automated generation of grocery lists, or the filtering of recipes according to dietary restrictions or preferences.

The actual documents are a compilation of individual recipes from an online cooking community, converted into printed form and re-OCR-ed to resemble the starting documents of the biosystematics scenario. This approach yields the further benefit that the original recipes easily convert into gold standard documents that allow for assessing the performance of experiment participants. The latter is necessary because Assumption U1 cannot be expected to hold in this scenario, as participants do not actually benefit from the work they do, as opposed to the biologists from the field studies.



## **5 Basic Approach & Research Problems**

In general, there are two ways of generating of semantic markup for digitized legacy documents, namely automated generation by means of NLP and manual generation by users. This chapter first discusses their individual advantages and drawbacks. It then outlines a hybrid approach to semantic markup generation, which utilizes the advantages of both of them to overcome either one's drawbacks. The final two sections highlight the fundamental research problems to solve for this hybrid approach to work and formulate design goals to address these problems.

### **5.1 Automated Semantic Markup Generation**

NLP provides powerful techniques for the automated generation of semantic markup. It requires very little manual effort from users, namely only for feeding the input documents to the NLP tool(s). However, NLP is not sufficiently accurate to produce output that complies with Requirement O1 in the general case. This holds for individual parts of semantic markup, e.g. named entities, according to Assumption T1, and even more so for comprehensive semantic markup as a whole (as specified in Definition 2.1), according to Assumption T2.

### **5.2 Manual Semantic Markup Generation**

Their expert judgment enables human users to manually create highly accurate semantic markup. However, the manual effort for such an endeavor is prohibitive for all but negligible amounts of data. The main reason for this is the sheer amount of markup that users have to create. In addition, the highly repetitive work, e.g. marking each and every single named entity by hand, is likely to deteriorate both users' motivation and concentration (Assumption U4). The latter, in turn, likely results in a higher amount of errors (Assumption U3), and consequently also has a negative impact on data quality.

### **5.3 Discussion & Synthesis**

What follows from the two previous sections is that the generation of high quality semantic markup for a significant number of documents is not possible with either approach, neither purely automatically nor purely manually.

A closer look at which cases are problematic in either approach, however, reveals a third option. In particular, NLP is rather good at handling common standard cases, which are unambiguous and comply with common rules; its problems are the non-standard special cases that it encounters occasionally. On the other hand, the common standard cases are the ones that are repetitive, tedious, and boring for human users;

the non-standard special cases not so much. This makes the following hybrid approach promising:

First, let NLP generate semantic markup automatically, which is highly likely to mark the standard cases correctly and leave errors mostly with the non-standard ones. Second, have a human user correct the auto-generated markup; all the user has to do is manually correct the errors the preceding NLP run left, which requires no action in the 90 – 95% of the cases already marked correctly, and thus strongly reduces manual effort. This induces the following preliminary definition:

**Definition 5.1p:** Semi-automated generation of semantic markup is a process, in which first NLP algorithms generate semantic markup, which is then corrected by a human user.

What may be perceived as a very crude and basic version of this approach has been used in to create and correct POS tags for a corpus in the Penn TreeBank project [Marcus 1994], some other linguistic corpus creation projects afterward [Chinchor 1997, Fellbaum 1998, Kim 2003], or projects that extracted meta data from documents [Tonkin 2008, Lu 2008]. However, these projects worked strictly on the sentence level and below, used a single cycle of NLP application and correction, generated rather shallow markup, and used documents the size of individual abstracts. They never defined or even studied the approach in general, and much less its applicability to the multi-step generation of complex semantic markup in large digitized documents like the ones from the biosystematics scenario.

As already discussed for Assumption T2, errors likely propagate when multiple NLP algorithms build upon each other's results. This results in a relatively high rate of errors to correct in the output of multi-step processes, e.g. information extraction. In particular, the errors of each step incur further errors in subsequent steps. The only way to counter error propagation is to correct errors before proceeding to the next step. Therefore, each individual step of a multi-step semi-automated semantic markup generation process has to consist of markup generation and manual correction. This induces the following definition:

**Definition 5.1:** Semi-automated generation of semantic markup is a multi-step process in which each step consists of two phases: First, in the **automated phase**, NLP algorithms generates semantic markup, and then, in the **correction phase**, a human user corrects this markup. Only after the correction phase, the process proceeds to the next step.

On purpose, Definition 5.1 does not limit the number of NLP algorithms that run in the automated phase of a step. The rationale is that the output of an NLP algorithm is corrected before it becomes the input of another NLP algorithm to prevent error propagation, and the latest before the end of the whole markup process to prevent errors from becoming part of the final result. In other words, the definition does not require errors to be corrected immediately after they occur, but only before they affect the performance of a subsequent NLP algorithm or the quality of the final result.

An immediate consequence of this is that two NLP algorithms that are agnostic of each other both in input and output can be bundled in one step. There are some special cases in which it even makes sense to bundle multiple NLP algorithms in one step even though they build upon each other's results, namely when only the last one in a

sequence of algorithms contributes to the final output of the process. An example of such a case can be POS tags, which may well be created as a basis for chunking and parsing, but not have a role in any further steps, in particular if the latter deal with higher levels of document structure. In such cases, it may make sense not to correct the POS tags, but only the chunking result because the overall effort is lower. However, this decision depends not only on the involved NLP algorithms, but also on their performance on specific input documents (layout peculiarities, etc.), and has to be decided on a case-by-case basis.

## 5.4 Research Problems

Generating semantic markup in a multi-step interactive process raises many questions, especially with regard to reducing the human users' manual correction effort as far as possible. Considering how fast modern computers have become today, waiting for NLP algorithms to finish in the automated phase of a step likely accounts only for a marginal fraction of the time a user spends with semi-automated markup generation; checking and correcting NLP results in the correction phase accounts for the lion's share. Therefore, the main focus is on reducing the effort users spend on manual corrections. There are several dimensions to this problem:

**Problem P1:** How to optimally support users in generating semantic markup, especially in manual correction? This is, what does an application for semantic markup generation have to provide users with to simplify their task as far as possible?

While domain experts are profound in their particular domain, they cannot generally be assumed to be familiar with XML, and much less with NLP. Having to deal with the peculiarities of both – e.g. editing the strict and delicate syntax of the former on the plain text level, or explicitly running the latter from the command line, possibly with complex parameter settings – is likely to appear relatively complex and scary to domain experts, besides being error prone. This complexity and fragility is likely to deteriorate their motivation to participate in semantic markup efforts (Assumption U2). To counter this, an application that supports domain experts in generating semantic markup needs to encapsulate and/or hide complexity as far as possible and provide correction facilities that are more forgiving than text level XML editing.

**Problem P2:** How to optimize individual NLP algorithms and their respective implementations so their output is the least effort to correct?

NLP algorithms are close to never 100% accurate (Assumption T1). The errors they make are not random, however. On the contrary, there are quite specific cases that incur errors for most NLP algorithms, be it due to specific weaknesses in the rules they use or due to specific peculiarities the data they were trained on. In either case, it is tedious for users to correct the same sorts of errors time and again (Assumption U4). Thus, it is essential for large-scale semantic markup generation that NLP algorithms do not repeat errors that users have corrected several times before.

Furthermore, it is not clear in general whether or not all possible errors a given NLP algorithm can make cause the same correction effort for users.

**Problem P3:** Especially for digitized legacy documents, semantic markup generation is a complex process: Besides generating structural and detail markup, it has to deal with the artifacts that originate from print layout and OCR. How to arrange the individual markup generation steps so the overall correction effort is minimized?

Suppose a semantic markup generation process marks sections, paragraphs, and named entities. A top-down approach would first mark the sections, then the paragraphs. This is not optimal, however, as section boundaries always coincide with paragraph boundaries, but not the other way around. Thus if paragraph boundaries are correct, sections can be marked as groupings of consecutive paragraphs, whereas section boundaries would be of limited help in marking paragraphs. On the other hand, a bottom-up approach would first mark named entities, and then paragraphs. This is not optimal either, as paragraph boundaries provide natural boundaries to named entities and thus would be helpful to be given, whereas named entities provide little help in identifying paragraph boundaries. As shown in a rather generic and simple example, both top-down and bottom-up generation of semantic markup are far less than optimal in the general case. Determining the optimal order of steps in a semantic markup generation process requires more sophisticated means, which are more process specific and consider the actual NLP algorithms employed in the individual steps.

**Problem P4:** How to support users in performing a complex, multi-step markup generation process? This is, how to enforce the optimized order of the steps, and how to make sure users do not miss any errors in the correction phase of each step, so to prevent error propagation?

Once a complex semantic markup generation process is optimized in response to Problem P3, its individual steps can rely on that any preceding steps have been executed and that their respective results have been corrected. However, once steps are ordered in this way, and maybe NLP algorithms are in use that explicitly exploit the results of previous steps, it becomes essential for the steps to be executed in their intended order, and for all errors to be corrected before proceeding from one step to the next. If the user fails to correct some errors in the correction phase of a given step, these errors propagate to the next step, causing more errors (Assumption T2). If a given step is executed prematurely, or not at all, this may cause a considerable number of needless errors as well: In the former case, the input of the step is suboptimal, and in the latter, so is the input of subsequent steps. Despite all benevolence, users cannot be assumed to spot each and every error in every step for every document, and they might accidentally deviate from the optimal ordering of steps from time to time (Assumption U3). This demands mechanisms that guide users through semantic markup generation processes step-by-step, circumnavigating both sorts of problems.

**Problem P5:** Quite often, several steps of a semantic markup generation process require users to have considerable domain knowledge, e.g. disambiguating



named entities, so users have to be domain experts. However, dealing with print layout and OCR artifacts in digitized legacy documents does not take a domain expert. So, how to relieve domain experts from these tasks?

Suppose a small number of domain experts work on a rather large body of legacy documents. Even though they have a proper interest in the data they process and extract knowledge from (Assumption U1), cleaning up print layout and OCR errors might occur increasingly repetitive and tedious to them over time, hampering the whole effort (Assumption U4). This demands a solution that relieves domain experts from cleanup – or generic tasks in general – and lets them concentrate on steps they need their expertise for. Put the other way around, it demands alternative ways of dealing with non-expert steps, e.g. delegating them to non-expert contributors. The latter, in turn, do not necessarily have a proper interest in the data they process, so Assumption U1 cannot be assumed to hold in such a scenario, which thus requires other means of motivating users and ensuring data quality.

## 5.5 Design Goals

The previous section has pointed out five central problems to solve in order to facilitate large-scale generation of high quality semantic markup for digitized legacy documents by means of the semi-automated approach introduced in Section 5.3. This section takes a more constructive point of view. In particular, it formulates design and optimization goals for applications and algorithms that implement the semi-automated approach.

**Goal 1:** Provide an intuitive and familiar-looking user interface that hides the complexity of editing XML on the character level and encapsulates parameterized invocations to NLP algorithms.

The aim of Goal 1 is to address Problem 1 on the level of the general user interface, like an application’s main window. It makes the application look less scary and shields the fragile XML syntax, helping users with both.

**Goal 2:** Provide flexible views on XML documents that in the correction phase of each step provide the user with exactly the information and editing options required for the correction task at hand.

Goal 2 complements Goal 1 on the level of specialized correction facilities for the results of individual NLP algorithms, also addressing solving Problem 1. Such specialized views further reduce complexity and help the user focus on what he needs to do.

**Goal 3:** Optimize NLP algorithms so their errors are the least effort to correct.

Addressing Problem 2, the rationale behind Goal 3 is that while NLP algorithms will always produce *some* errors (Assumption U1), it may be possible to influence *which* errors they make, which opens up possibilities for tuning. Goal 3 consists of two parts: First, to find out which combination of errors (e.g. false positives and false negatives in NER) are the easiest to correct. This combination may not be the same across all NLP algorithms involved in a semantic markup process, so this first partial

goal may have different answers for individual groups of algorithms. Second, tune NLP algorithms to produce errors in the optimized combination wherever possible.

**Goal 4:** Have NLP algorithms learn from the corrections users make to their output so errors do not repeat.

Goal 4 addresses Problem 2, aiming to save users the tedious burden of correcting the same errors over and over again, which annoys them and deteriorates their motivation (Assumption U4). Continuous learning also likely reduces the overall number of errors, and thus the correction effort, because NLP algorithms constantly learn previously unknown cases or even classes of cases, depending on if and how they generalize user-labeled input.

Active Learning provides interesting algorithms for achieving Goal 4. A difference is that while Active Learning chooses one instance at a time to be labeled, users generating semantic markup simply correct all errors at once. This results in a varying number of instances a learning algorithm obtains the correct label for in a round of learning, namely all the erroneous ones, and the learning algorithm does not get to explicitly choose the instances for the user to label.

On the other hand, as the users correct all errors that a given NLP algorithm left behind, a document becomes fully labeled data afterward; this facilitates the use of Supervised Learning algorithms at the end of correction.

**Goal 5:** Find rules or guidelines for arranging the steps of a complex markup process in such a way that the overall correction effort is reduced as far as possible.

Striving a solution to Problem 3, Goal 5 considers a semantic markup generation process as a whole and aims at overall optimization. This may not even mean to reduce the overall number of errors, as the errors of different NLP algorithms may vary widely in the effort they require to correct them. The guidelines should consider the specificities of the individual NLP algorithms involved in a given markup process, their interdependencies in this process, and also possible peculiarities of the data the process is intended to generate semantic markup for.

**Goal 6:** Make semantic markup applications sufficiently flexible to allow for implementing a given semantic markup process in its optimized order of steps.

Goal 6 complements Goal 5 on the technical level; it basically emphasizes that solutions to Problem 3 found in Goal 5 should be easy to implement or deploy.

**Goal 7:** Provide users with a guidance mechanism that assists them in performing complex multi-step semantic markup processes.

The mechanism specified in Goal 7 addresses Problem 4. It has two particular responsibilities: To make sure the individual steps of the markup process are executed in their intended (optimized by means of the guidelines from Goal 5) order, and to make sure users do not proceed to the next step before having corrected all errors in the current one.

**Goal 8:** A semantic markup application must be sufficiently slim to run on a standalone desktop or laptop computer.

Goal 8 is not motivated by the problems identified in the previous section, but by the observation that domain experts tend to have a rather tight schedule. Thus, they should be able to work on semantic markup generation when they have the time to, e.g. on the road, where access to the World Wide Web (or a network in general) is not a given. Goal 8 particularly implies that web service calls should be reduced to a minimum, so the implementations of NLP algorithms should not require several gigabytes to main memory to run, e.g. to accommodate enormous statistical models.

**Goal 9:** Provide infrastructure for distributing the individual steps of a given semantic markup process across multiple users. In particular, provide mechanisms that reduce the workload of domain experts to steps they need their special knowledge for, delegating the remaining steps to non-experts.

The rationale behind Goal 9 is to address Problem 5 by involving more people in semantic markup generation projects to increase throughput. As domain experts are scarce, this requires the involvement of non-experts. The latter, however, is not feasible if every single participant is to perform the whole markup process because some steps require expert knowledge. A coordinating infrastructure that passes each document through several hands in the course of the markup process would solve this. In particular, it facilitates to restrict the experts' involvement to the correction phases of the steps they need their special knowledge for, while non-experts take over the correction phases of the remaining steps.

**Goal 10:** Find means of motivating users and ensuring data quality in a non-expert environment.

Goal 10 is motivated by an immediate consequence of Goal 9: As non-experts do not have a proper interest in the documents they are intended to work on, Assumption U1 cannot be assumed to hold for non-experts. Especially when motivated by monetary rewards, non-experts cannot be assumed to generally care about data quality if the latter does have an immediate impact on the reward [Eckert 2010]. This necessitates data quality enforcement mechanisms that are integrated with the reward mechanism.

And even with benevolent non-experts who contribute mainly out of idealism, data quality enforcement still is an issue: In the general case it is impossible to thoroughly train a large number of non-expert contributors, who might even come from a loose community that is distributed all over the internet. Thus their corrections are not of reliably high quality, or at least cannot be assumed to be in the general case, so data quality enforcement remains essential.



## 6 Related Work

This chapter discusses existing approaches and solutions to Problems 1-5 and Goals 1-10 and evaluates them with regard to their applicability in semantic markup generation.

### 6.1 Markup Visualization

This section discusses the different existing approaches to XML visualization in general, not necessarily in combination with NLP, to give a comprehensive overview of possible approaches to the visualization aspects of Goal 1 and Goal 2.

#### 6.1.1 Syntax Highlighting

Syntax highlighting is a visualization feature frequently found across many advanced text editors like UltraEdit [UltraEdit], Notepad++ [Notepad++], or Emacs [Emacs]. It helps users to understand all sorts of text based data, be it source code of some programming language or XML data. For the latter, syntax highlighting is also abundant in purpose-built XML editors like XML Spy [XmlSpy] or `<oXygen/>` [oXygen].

```
<paragraph>
  <sentence>
    <subject>
      <person>
        <firstName>Paul</firstName>
        <lastName>McCartney</lastName>
      </person>
    </subject>
    <predicate baseForm="write" tense="past">wrote</predicate>
    his
    <object>
      <workOfArt type="book" title="My Life and Times">autobiography <quot;My Life and Times</workOfArt>
    </object>
    <adverbial type="time" detailType="year" value="1998">in 1998</adverbial>.
  </sentence>
  <sentence>
    <subject>
      <organization type="company">McMillin Publishing</organization>
    </subject>
    <predicate baseForm="publish" tense="past">published</predicate>
    <object>
      <coreference target="preceding::workOfArt[@type='book'][1]">the book</coreference>
    </object>
    <adverbial type="time">
      <coreference target="preceding::adverbial[@type='time'][1]">the same year</coreference>
    </adverbial>
    (
      <attributive target="..object" type="ISBN">ISBN 1-56592-235-2</attributive>
    ),
  </sentence>
</paragraph>
```

Figure 6.1: The document from Example 2.3.2 with XML syntax highlighting<sup>6</sup>

Syntax highlighting displays different parts of text data in different fonts, mainly differentiating font color and plain vs. bold vs. italicized style; variations in font face

---

<sup>6</sup> taken in Notepad++ in XML mode

are less common. The intention is to give users visual clues towards the function of individual parts of text data, for instance the distinction between tags, tag attributes, and textual content in XML. Figure 6.1 displays the semantic XML document from Example 2.3.2 with XML syntax highlighting.

A drawback displaying the full XML syntax in semantic markup, especially on the detail level, is that it renders the document text very hard to read, as it is scattered between a multitude of XML tags. The tags account for a large fraction of the displayed characters, about 85% of the non-whitespace characters in Figure 6.6, for instance. Syntax highlighting can hardly alleviate this.

### 6.1.2 Code Folding

Code Folding [Mössenböck 1996] strives to render XML documents (and source code as well) easier to review by folding away parts that are not needed at a given point. However, this approach is ill suited for alleviating the problems with full XML display and syntax highlighting. Suppose wanting to display only those XML elements in Example 2.3.2 that belong to the parsing result, i.e., the subject, predicate, object, adverbial, and attributive elements in this case. This requires hiding the NE and co-reference tags, i.e. the `person`, `workOfArt`, `organization`, and `coreference` elements. Figure 6.2 shows the document from Example 2.3.2 with these tags folded away. The problem is obvious: Code folding does not only hide tags that are not required at a given point, but also their textual content, rendering the document text essentially unreadable.

```

<paragraph>
  <sentence>
    <subject>
      <predicate baseForm="write" tense="past">wrote</predicate>
      his
    <object>
      <adverbial type="time" detailType="year" value="1998">in 1998</adverbial>.
    </sentence>
  <sentence>
    <subject>
      <predicate baseForm="publish" tense="past">published</predicate>
      <object>
        <adverbial type="time">
          (
            <attributive target=" ../object" type="ISBN">ISBN 1-56592-235-2</attributive>
          ).
        </adverbial>
      </sentence>
    </paragraph>

```


Figure 6.2: The document from Example 2.3.2 with elements folded away<sup>7</sup>

### 6.1.3 Semantic Coloring

Semantic coloring is a visualization technique often found in NLP applications (see Section 6.2). It does not display XML tags, but only the plain document text, and

<sup>7</sup> taken in Notepad++ in XML mode

colors the background based on the existing markup. Figure 6.3 shows the semantic XML document from Example 2.3.2 with semantic coloring.



Paul McCartney wrote his autobiography "My Life and Times" in 1998. McMillin Publishing published the book the same year (ISBN 1-56592-235-2).

**Figure 6.3: The document from Example 2.3.2 with Semantic Coloring**

As opposed to XML syntax highlighting, semantic coloring keeps the document text compact and thus easily readable, as it only modifies the background colors. In addition, the coloring scheme can also reflect the actual semantics, i.e. the names of semantic XML elements.

A severe drawback of semantic coloring is, however, that it cannot simultaneously display multiple tags that mark different semantic aspects of a given phrase, e.g. the subject and the person tag around “*Paul McCartney*” or the object and the coreference tag around “*the book*”. Furthermore, it can also show the attributes of the tags, e.g. the target attribute of the coreference tag around “*the book*”, which is highly important for specifying that “*the book*” actually refers to “*autobiography 'My Life and Time'*” in the preceding sentence. An additional drawback is that semantic coloring hardly suited for displaying structural markup.

To handle nested elements, semantic coloring is often paired with a control widget through which a user can activate and deactivate highlighting for individual element names – this is the case in all the NLP frameworks that use this visualization technique (see Section 6.2). The attributes of XML elements are usually displayed in yet another interface widget. This can and often does result in a rather complex arrangement of widgets in user interface.

#### 6.1.4 Summary

Both semantic coloring and XML syntax highlighting are promising approaches to the visualization aspects of Goal 1 and Goal 2. However, both have severe drawbacks: The former cannot display structural markup in an easy-to-overview fashion and cannot simultaneously display multiple levels of complex nested detail markup at all. The latter, in turn, intermixes the document text with lots of XML tags, rendering it hardly readable, especially with complex nested detail markup; code folding cannot alleviate this without hiding parts of the document text, which completely deteriorates readability.

## 6.2 Existing Applications & Frameworks

This section reviews existing applications and frameworks, both text and XML editors and NLP workbenches, with an emphasis on their NLP functionality and their facilities for markup visualization and correction, as well as the flexibility of their NLP features.

### 6.2.1 Text and XML Editors

General-purpose **text editors** like UltraEdit [UltraEdit] or Notepad++ [Notepad++] are powerful editors for all kinds of text-based data, e.g., plain text, XML, or a

multitude of programming languages. Many of these editors natively provide syntax highlighting and code folding for XML and for common programming and script languages. Some also support recording macros for frequently used editing steps, and for including external components.

On the other hand, for most text editors XML is one data format among many, so they do not provide any special support for XML editing beyond the visualization features already mentioned, e.g., some sort of support for inserting XML tags. This shortcoming renders the latter unnecessarily cumbersome, since the functional parts of XML tags can be inserted automatically.

Specialized **XML editors**, like the widely used XML Spy (XmlSpy) and <oxygen/> (Oxygen), are explicitly built to support handling existing XML data and related techniques like DTDs and XML Schemas for validation, the XPath and XQuery query languages, XSLT, etc. Given an XML Schema, they also provide some automation in creating new tags, e.g. wrapping a selected piece of text in some element that is permitted according to the schema. However, this sort of tagging support forces markup to be created in a strict top-down sequence, i.e. first the structural markup and only then the details. In addition, the tagging feature often does not guard the XML document's wellformedness in that it often does not prevent generating a new tag in the middle of an existing one; this happens if a user's text selection is off by a few characters when using the feature. Once such an editing error happens, the document is no longer well-formed, which often disables all user support, including syntax highlighting, leaving the user alone with nothing but a plain text view. There is rarely any support for modifying or removing existing XML tags.

Aside from their native support of query and transformation languages, XML editors rarely provide any mechanisms for automated changes to XML documents, neither to the content nor to the markup. They are not designed to apply NLP, either, because it is rarely used in handling data centric XML documents, the main focus of XML editors.

### 6.2.2 GATE

The General Architecture for Text Engineering (GATE) [GATE] is an NLP workbench developed by the University of Sheffield. It offers a wide variety of functions, among others text tokenization, POS tagging, noun-phrase and verb-phrase chunking, NER, semantic parsing, and co-reference resolution. The former four build upon each other and are the basis for the latter three, which are interesting for the generation of detail-level semantic markup. The GATE framework also includes Apache Lucene [Apache Lucene] as a basis for information retrieval, and a GUI for visualizing NLP results by means of semantic coloring. GATE further provides powerful facilities for developing new NLP components, including a bootstrapping tool and an editor for JAPE (Java Annotation Patterns Engine) grammars. These grammars are essentially regular expression patterns that are matched against existing markup (instead of a plain character sequence) and output new markup in case of a match. Individual NLP components can be arranged into a pipeline in user-definable order, so to execute them in sequence; thus GATE effectively achieves Goal 6.

However, the purpose of GATE is NLP research and automated evaluation rather than the generation of comprehensive semantic markup. GATE readily provides an AnnotationDiff tool for computing f-Scores from the results achieved with test



corpora in evaluation tasks. On the other hand, it lacks any facility for manually editing the document text or the generated markup, not at the end of its pipeline, and much less in between, rendering it completely unsuited for the interactive approach taken in this work. In addition, GATE's semantic coloring visualization features (see Figure 6.4, semantic coloring control widget on the right) are suited for relatively small documents and for detail-level markup only, as found in NLP evaluation tasks, but not for large documents and not for structural markup.

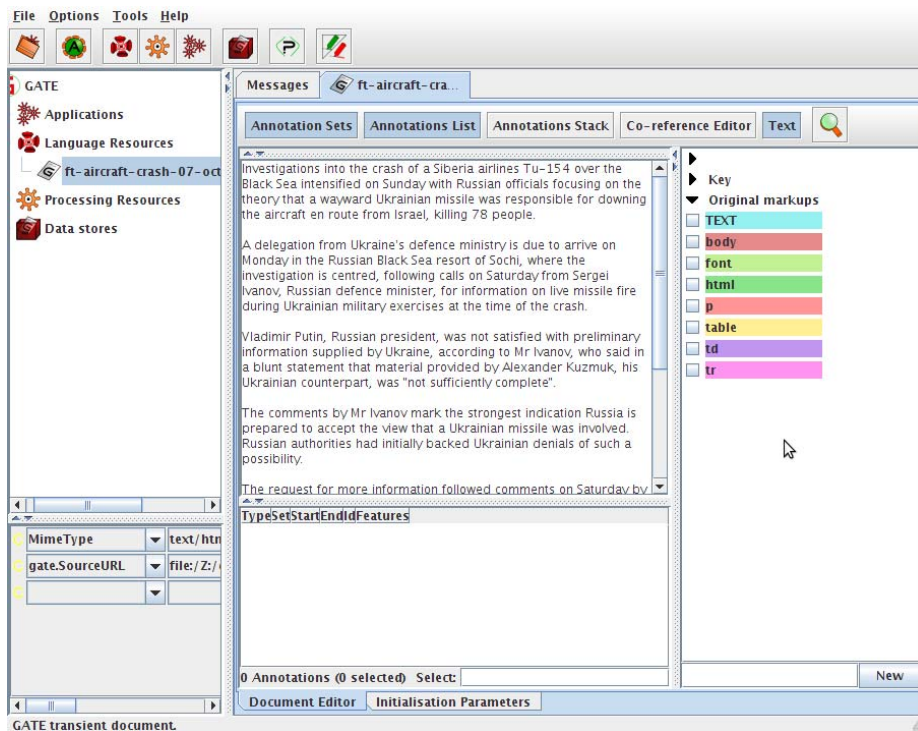


Figure 6.4: The GATE user interface<sup>8</sup>

### 6.2.3 Knowtator

Knowtator [Knowtator] is an NLP plug-in for Stanford University's widely used ontology editor Protégé [Protégé], developed by University of Colorado Health Sciences Center. It uses the Protégé's knowledge representation to specify annotation schemas. Furthermore, Knowtator can integrate existing Protégé ontologies in its NLP components, and it can feed back extracted data into these ontologies. The Knowtator user interface (see Figure 6.5, semantic coloring control widget on the upper left) provides semantic coloring visualization of generated markup and also features manual correction facilities. However, Knowtator suffers from all the shortcomings that come with semantic coloring; namely, it is problematic for large documents and for correcting structural markup. In addition, the user interface as a whole is rather

<sup>8</sup> Source: <http://gate.ac.uk/sale/tao/index.html>

complex, as it consists of many different views and input fields that are displayed alongside one another.

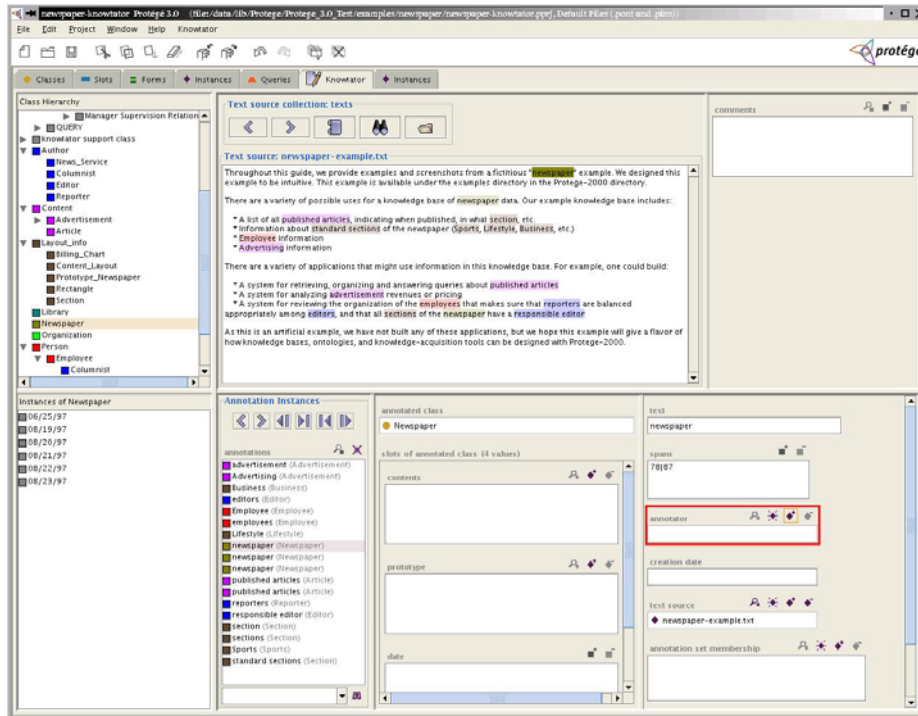


Figure 6.5: The Knowtator user interface<sup>9</sup>

## 6.2.4 WordFreak

WordFreak [WordFreak] is a linguistic markup tool developed by the University of Pennsylvania. It supports both human and automatic annotation of linguistic data, and it can actively learn from manual corrections of auto-generated markup. Like GATE, however, its NLP functionality and visualization capabilities are restricted to relatively small documents and detail-level markup like POS tags, noun-phrase and verb-phrase chunking, NER, or parsing. Figure 6.6 and 6.7 show two task specific layouts of WordFreak's user interface, namely the one for NER correction in Figure 6.6 and the one for POS tags and chunking in Figure 6.7. A special feature of WordFreak is that the user interface always consists of two frames, one for visualization (on the left in both figures), and the other one for task specific input (on the right in both figures). This sort of user interface is hardly suited for overviewing structural markup in large documents. In addition, judging from the functionality it provides, WordFreak is primarily intended for use by (computational) linguists rather than domain experts like biologists or historians.

<sup>9</sup> Source: <http://knowtator.sourceforge.net/tour.shtml>

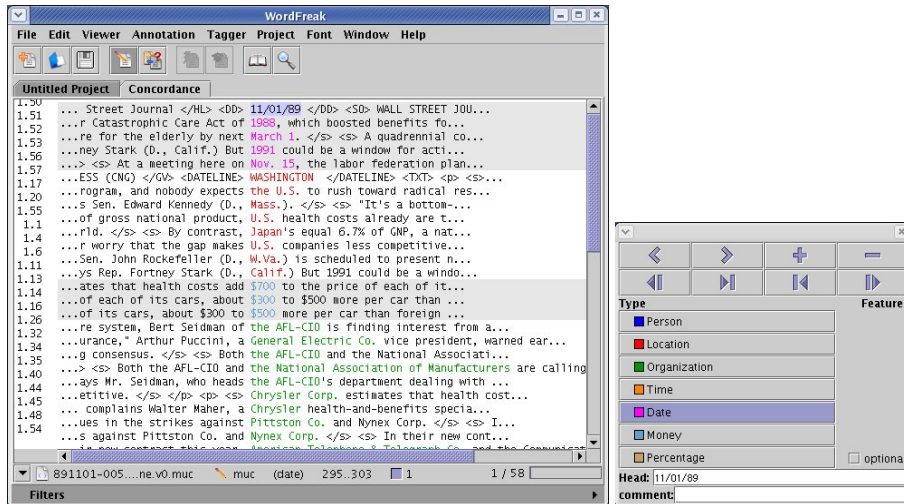


Figure 6.6: WordFreak user interface for NER<sup>10</sup>

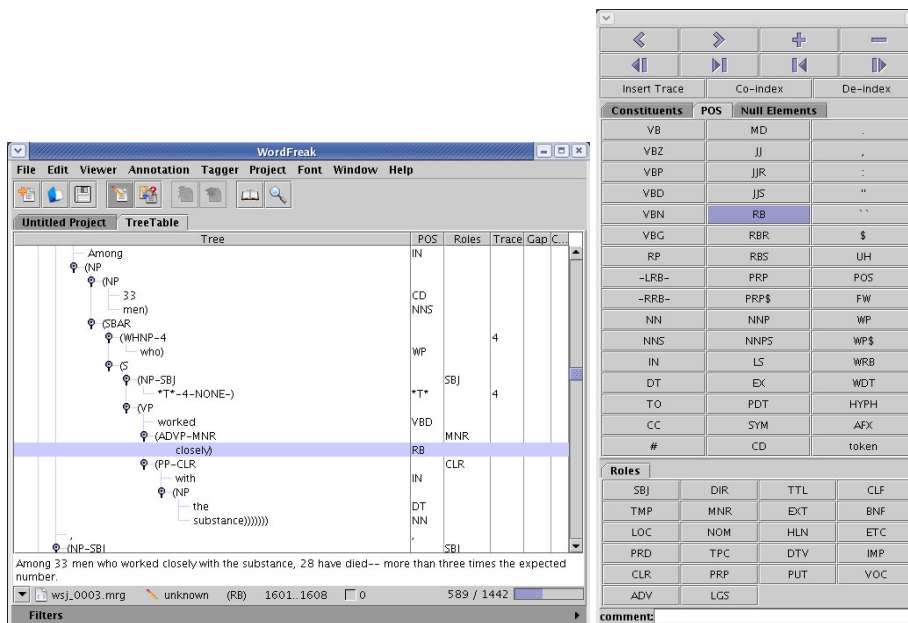


Figure 6.7: WordFreak user interface for POS and Chunking<sup>9</sup>

## 6.2.5 Summary

Existing text and XML editors provide powerful features for visualizing both structural and detail markup, with an abundance of the latter rendering the document

<sup>10</sup> Source: <http://wordfreak.sourceforge.net/screenshots.html>

text close to unreadable, however. They offer little support for manual editing, and no NLP integration at all.

Existing NLP workbenches and applications are primarily suited for automated NLP execution and subsequent visualization of the results. Facilities for manual corrections are rudimentary or not available at all, and can be used only after the whole sequence of NLP algorithms has run. Intermediate corrections are not intended, and thus no means of preventing the propagation of NLP errors. In addition, correction features, if available, are restricted to corrections to the markup; there is no way of correcting the actual documents text, e.g. to deal with hyphenation or OCR errors in digitized legacy documents. None of the discussed applications is suited to generating, much less to displaying or correcting structural markup in large documents. One reason for this common shortcoming likely is that all of them are primarily designed to process texts the size of abstracts or individual newswire articles, the documents usually used in NLP evaluation tasks. Furthermore, all three applications encapsulate the complexity of NLP only to a certain degree, as their visualization facilities display a fair share of it in their layout alone.

The user-configurable pipelining of NLP components implemented in GATE provides an interesting approach to Goal 6. However, how to integrate the pipelining approach with manual corrections between individual steps remains an open question. The configurable semantic coloring visualization of detail-level markup is an alternative to displaying the whole XML syntax, and thus an interesting approach to Goal 1 and Goal 2, but it is rather ill suited for visualizing structural-level markup. It remains an open question how to integrate XML syntax highlighting for structural-level markup with semantic coloring for detail-level markup, and how to integrate this with a user interface that is easy to grasp, yet allows for manual corrections to the markup on either level.

### 6.3 Specialized Editing Views

As seen in the last section, flexible views on an XML document that optimally support a user in correction the result of a specific NLP algorithm are not readily available. An alternative approach to Goal 2 is to use XML transformation languages like XSLT [XSLT] or XQuery [XQuery] to project out the markup a user does not need to see for a specific correction.

However, even in the world of relational databases, where views are an integral and widely used concept, view based updates to the underlying base relations (also known as the “view update problem”) are subject to theoretical restrictions [Bancilhon 1981]. In the XML world, transformations via XSLT [XSLT] or XQuery [XQuery] usually produce new documents that are independent of the original ones. Actual views that change if the underlying data changes are unknown in practice, not to mention views that write updates through to their underlying documents. In addition, there is no standardized and widely used language for updating XML documents so far; the XQuery Update Facility [XQueryUpdate] still has the status of a candidate recommendation, as of 2011. Updatable views of XML document are thus far from becoming applicable in practice, and consequently views based on XML querying or transformation languages are not a realistic approach to Goal 2.

## 6.4 Natural Language Processing Implementations & Models

This section first introduces several different representations of document text and associated semantic markup that are used by existing implementations of various NLP algorithms. Second, it reviews a broad variety of existing NLP implementations, with a focus on how well they are suited for interactive semantic markup generation, i.e., how well they achieve Goal 3 and Goal 4. A review of incremental machine learning techniques as possible approaches to Goal 4 closes this section.

### 6.4.1 Data Models

Across the wide variety of implementations of NLP algorithms, several different representations of a document's text and its associated semantic markup are used. These representations vary widely in complexity and expressiveness. Although this was never one of their design goals, some representations can also handle basic edits to the document text.

**Slash-Tag.** A very early representation, and arguably the most primitive one, is to append a tag to each token of the document text, separated by a forward slash, as used in MUC-1 [Grishman 1996]. Example 6.1 shows Example 2.3.2 with NE tags in slash-tag format, tags in bold for readability. The tags that start a named entity start with an "S" (for start), followed by an indicator for the type of named entity; the "SP" appended to "Paul", for instance, reads "start person", "SWoA" reads "start work of art", "SO" reads "start organization", and "SD" reads "start date". Tokens that continue a named entity are tagged with a "C", for "continue"; the "C" tag of "McCartney", for instance, indicates that this word continues the person that started with "Paul". Tokens that do not belong to a named entity are tagged with an "O", for "other".

```
Paul/SP McCartney/C wrote/O autobiography/O "/O My/SWoA  
Life/C and/C Times/C "/O in/O 1998/SD ./O McMillin/SO  
Publishing/C published/O the/O book/O the/O same/O year/O  
(/O ISBN/O 1-56592-235-2/O )/O ./O
```

#### Example 6.1: Slash-Tag notation for named entities

This format is capable of representing only one layer of semantic tags; already the generation of another layer of tags would be more hampered than helped by the tags of the first layer, which are intermixed with the document text. The Slash-Tag representation is almost exclusively found in NLP evaluations, namely for POS-Tagging, NER, or NP-Chunking.

**Tag-Array.** The Tag-Array data model tokenizes the document text and represents it as an array of strings. Each layer of semantic markup is represented as an additional string array that holds the layer's tags. Example 6.2 shows Example 6.1 in Tag-Array format; the meanings of the tags are the same as above, "|" characters separate the individual array entries.

```

{Paul | McCartney | wrote | autobiography | " | My | Life |
{SP | C | O | O | O | SWoA | C |
and | Times | " | in | 1998 | . | McMillin | Publishing |
C | C | O | O | SD | O | SO | C |
published | the | book | the | same | year | ( | ISBN |
O | O | O | O | O | O | O | O |
1-56592-235-2 | ) | .}
O | O | O}

```

### Example 6.2: Tag-Array format for named entities

This format can represent any number of layers of semantic markup. However, when multiple layers are generated, it is hard to keep track of which tag array represents which layer, as the individual arrays do not bear markers of any sort. Furthermore, each attribute for each layer of markup requires an additional array to represent it, e.g. the essential target attribute of the `coreference` tags in Example 2.3.2, further increasing the danger of confusing the individual arrays' meanings. To give another example, the semantic markup of Example 2.3.2 in its entirety requires no less than 17 tag arrays to represent, not counting auxiliary tag layers that are not part of the XML, e.g. POS tags.

**Spans.** The span representation regards the document text as a plain continuous sequence of characters. Semantic markup is represented by means of marker objects, which reference the text through offsets, i.e., the indices of the characters where the marked piece of text starts and ends. These marker objects further have a type that indicates as what the referenced piece of text is marked. In many implementations, the spans can also bear attributes, sometimes also referred to as **features**. The spans themselves are often referred to as **annotations**, e.g. in the data model underlying GATE (see Section 6.2.2). Most implementations bundle the document text and the associated spans in a dedicated document object. Example 6.3 shows the span notation for the named entities in Example 2.3.2, attributes in brackets; the character sequence marked by the spans is given in italics for clarity.

```

Paul McCartney wrote autobiography "My Life and Times" in
1998. McMillin Publishing published the book the same year
(ISBN 1-56592-235-2).
span spanType=person startOffset=0 endOffset=14
()
Paul McCartney
span spanType=workOfArt startOffset=36 endOffset=53
(type="book" title="My Life and Times")
My Life and Times
span spanType=date startOffset=58 endOffset=62
(type="time" detailType="year" value="1998")
1998
span spanType=organization startOffset=64 endOffset=83
(type="company")
McMillin Publishing

```

### Example 6.3: Spans marking named entities

The span data model is highly versatile and expressive, and many existing NLP implementations use it. Spans whose types come from the ENAMEX tag set were the result submission format for MUC-7 [Chinchor 1997].

**Text Editing.** Neither of the tree data models presented here was designed to handle editing operations to the document text. Nevertheless, some can handle some updates without destroying the semantic markup. In particular, the Slash-Tag notation can handle updates to the text rather well because the tags are embedded and therefore are not dislodged by changes. The Tag-Array representation can handle updates to individual tokens as long as the number of tokens does not change; thus correcting a character-level OCR error would not be a problem, but de-hyphenating a word would be, as the latter merges two tokens into one. Spans are rather sensitive to updates to the text because of their offsets. The latter get dislodged even if the text the spans point to changes by only one character. Thus, the span data model can only handle one-for-one character substitutions in the general case.

#### 6.4.2 NLP Implementations

There is a multitude of existing implementations of NLP algorithms available; some of them provide a single component that implements a single algorithm, while others are collections of such components. In the latter, the individual implementations often build on the same data model for easier combination. This section reviews a selection of them. All of the NLP implementations presented here are pure components that are intended to be integrated in some application; some have primitive command line interfaces; however, all lack any sort of user interface or facilities for manual corrections.

**OpenNLP** [OpenNLP] is an umbrella project that hosts multiple smaller, mostly open-source projects that develop NLP tools. These tools are highly heterogeneous with regard to purpose, programming platform, and quality. Among others, the functionality provided comprises text tokenization, POS tagging, noun-phrase and verb-phrase chunking, NER, and semantic parsing. The former four build upon each other (in the order they are enumerated) and form the basis for the latter two. Most implementations in the OpenNLP suite share a common data model, namely a variation of the Tag-Array model; some can provide their output in the Span data model as well, and some exclusively use the latter representation for their output; the input always has to be in Tag-Array representation, however. None of the NLP implementations included in OpenNLP provide the functionality to learn from users' corrections, and none have any options to influence the type of errors that occur.

**LingPipe** [LingPipe] is a commercial NLP suite developed by alias-i. Apart from tokenization, which is rule based, almost all the analysis functions are based on statistical models, Hidden Markov Models, in particular. Thus, training LingPipe components for new domains or domain specific tasks requires pre-annotated data for supervised learning. Once a model is generated from training data, it can be used to generate semantic markup for documents. The underlying rationale of LingPipe is the ability to train and apply models for a wide range of purposes. Models for POS tagging and NER are readily available. LingPipe uses a variation of the Tag-Array data representation. Most LingPipe components can be trained incrementally, e.g. from user corrections, an approach its producer alias-i calls "learn-a-little, tag-a-little", which is interesting with regard to Goal 4. However, as LingPipe is

commercial, there is no information on how exactly the incremental training works behind the scenes. Furthermore, none of the LingPipe components provide any means of influencing what types of errors occur.

**StanfordNLP** [StanfordNLP] is an open-source suite of NLP implementations provided by the NLP Group of Stanford University. Among others, its functionality comprises POS tagging and NER. Both use statistical models, the POS tagger a maximum entropy model, and the NE tagger a Conditional Markov Model. The predominant data model throughout StanfordNLP is a hybrid of the Tag-Array representation and the Span representation. The available implementations are pure components for integration in other software, though some come with wrappers that provide command line interfaces. For most components, the StanfordNLP package also provides facilities for training new models, but there are none for incrementally training existing ones. There are no options or parameters that would allow to take influence on the kinds of errors that occur.

**LINNAEUS** [Gerner 2010] is an open source NER system specifically built to extract taxonomic names from biomedical literature. It uses a gazetteer list & rule approach with handcrafted rules. LINNAEUS provides no functionality to feed back manual corrections into its underlying data, and no means to influence the kind of errors that occur.

### 6.4.3 Incremental Learning Techniques

To achieve Goal 4, NLP algorithms used in interactive semantic markup generation need to learn from corrections that users make to their output. This section revisits the two approaches to incremental learning from Section 2.3.4 that are promising in this respect.

**Active Learning** is inherently designed to work interactively, with a user training an NLP component by giving it explicit feedback. In particular, the NLP component to be trained picks instances for the user to label, usually those instances that most improve its statistical model. These instances are dubbed to provide the highest **information gain**.

However, in interactive semantic markup generation, NLP components do not get to pick the instances for the user to label, but the user simply corrects all errors. Furthermore, active learning is designed to decrease the required amount of pre-labeled training data, not to indefinitely continue training during application. In particular, in early stages of training, when the statistical model being trained is not yet well developed, the number of errors can be expected to be rather high, burdening the correcting user with considerable effort. The alternative is extensive training ahead of deployment, possibly with pre-labeled data; however, a sufficient amount of labeled training data may not be readily available and generates high cost and effort to create, and even unlabeled training data may not be available in sufficient amounts.

A more theoretical problem of active learning in NLP algorithms is the notion of information gain. While it computes easily for SVMs, which active learning is mostly used to train, it is unclear how to define information gain in HMMs or CRFs, which are the basis of many NLP algorithms. However, computing information gain is necessary only for picking the instances for the correcting user to label, which has no relevance in semantic markup generation, as the user corrects all errors anyways.



**Online Learning** is specifically designed not to distinguish a training phase from an application phase, and thus for the training to continue indefinitely during application. It is thus a very promising approach to Goal 4. However, online learning has so far been used mostly for training perceptrons for time series prediction, e.g. for stock prices, and has seen no use in NLP. Consequently, there are no known figures about its expectable performance in this area. In addition, online learning works in an instance-by-instance fashion: The perceptron predicts the next value in a time series, then the actual value becomes available and is fed back into the perceptron before it makes the next prediction. This is fairly different from interactive semantic markup generation, where an NLP component first processes a whole document, and only then the user corrects all errors at once, thus possibly providing the correct output for many instances at the same time. Furthermore, it is not clear how to use online learning for incrementally training SVMs, HMMs, CRFs, or rule- and gazetteer-based systems.

## **6.5 Process Control Mechanisms**

There are several approaches Goal 7, namely to provide users with automated guidance through complex multi-step semantic markup generation processes. Most prominent in this area are workflow management systems (WfMS), but there are also mechanisms that are specifically designed to assist in the generation of semantic markup in corpus creation projects.

### **6.5.1 Workflow Management Systems**

Workflow management systems [Van Der Aalst 2003, 2004] are widely used; they control everything from online purchases to logistics to the compilation of conference proceedings [Mülle 2006]. To be executable in a WfMS, a workflow first has to be modeled in a respective modeling language. The most common of these languages is the Business Process Modeling Notation (BPMN) [White 2004], but there are others as well, many of them academic or experimental, e.g. YAWL [Van Der Aalst 2006]. After modeling, workflows are translated into an execution language to run in a WfMS, most commonly the Business Process Execution Language (BPEL) [BPEL].

Most workflow modeling languages share Petri Nets [Petri1962] or extensions thereof as their common theoretical basis. As a consequence, the transitions between activities are mostly well-defined and take place within a closed domain of states, e.g., who is next to take action in an editorial process. The activity or activities just completed, together with their result(s), determines which activity or activities are next. The state of a data item in the workflow therefore solely depends on the which activities have been completed and with what result. In general, workflow execution in a WfMS is process driven. In a semantic markup generation process, on the other hand, the state of a document can change almost arbitrarily, as users can freely edit the document in the correction phase of each step. In particular, they can corrupt or undo the results of steps already completed. Such arbitrary transitions do not model well in workflows modeling languages. It seems more promising and practical to execute a semantic markup generation process in a purely data driven fashion. This is,

to determine the state of a document in a markup process solely from the markup that already exists in the document and on the errors in this markup.

### 6.5.2 Annotation Control Mechanisms

Annotated corpora have been created in multiple domains, e.g. to provide training and evaluation data for NLP algorithms or input for document analysis. According to literature [Chinchor 1997, Fellbaum 1998, Kim 2003, Marcus 1994], most of these projects use a very primitive version of the interactive approach: First, NLP tools process the document text, and second, computer linguists manually correct the NLP result, up to a 98% level of inter-annotator agreement in the case of Marcus [1994]. This somewhat corresponds to manual correction at the end of an NLP pipeline. However, the control mechanisms and user interfaces used for correction are very limited in scope – for instance an Emacs plug-in that highlighted words for which annotators did not agree on the POS tag [Marcus 1994], or the highly specialized user interface of WordFreak. They are not intended and not suited to guide users through an entire semantic markup generation process, especially not domain experts like biologists or historians. To the contrary, they usually are special-purpose implementations for specific types of detail-level markup elements, and they are useful only for computer linguists. Furthermore, components built for a special purpose tend to be only very little generic, if at all, so changes to a semantic markup generation process might well require implementation level changes to components that support users in correction. Finally, the approach of finding potential errors by means of disagreement between correctors is questionable: As a consequence of Assumption U3, it may well happen that all correctors involved initially miss the same tricky errors. Thus these errors never get to the attention of the agreement-based highlighting mechanism, and no corrector has his attention drawn to them.

Another difference to the background of this work lies in the complexity of the task: The annotations<sup>11</sup> generated in the aforementioned projects only cover very low levels of the document structure, i.e., POS tags and sentence structure [Marcus 1994], word sense [Fellbaum 1998], named entities [Chinchor 1997], and domain-specific terms [Kim 2003]. They do not cover higher levels of the structure, like sections, subsections, or even paragraphs, not to mention cleaning up print layout. This is because these projects worked on small clean pieces of text that are atomic semantic units, e.g., the MEDLINE abstracts [MEDLINE] Kim used for the GENIA corpus [Kim 2003]. In the biodiversity scenario, in contrast, the documents consist of multiple (up to over 100) treatments, and the markup covers both them and their subsections. This renders semantic markup generation a lot more complex. In addition, as opposed to uniform and clean documents like MEDLINE abstracts, the documents in the biodiversity scenario are diverse, both in print layout that varied over time and in language, and they contain layout artifacts like page titles and OCR errors, which require extensive cleanup. Furthermore, the annotations they generated do not provide the semantic disambiguation information like geo-references for locations, which are essential for the extraction of useful data from the documents.

---

<sup>11</sup> Annotations are markers pointing to small pieces of text (individual words or phrases) in this context, with no specific representation. As opposed to this, XML markup can also express document structure.

## 6.6 Crowdsourcing

Recently, crowdsourcing has become a very popular approach to data processing tasks that require human input to improve data quality. In particular, crowdsourcing means to distribute individual small pieces of a large data processing task to a large number of users who make small contributions to the solution, usually over the Internet. This approach is highly promising for Goal 9, namely to crowdsource all the correction phase of steps of a semantic markup generation process that do require expert knowledge. This section discusses several prominent crowdsourcing platforms and projects, their advantages and disadvantages, and the experiences they gathered.

### 6.6.1 Amazon Mechanical Turk

The Amazon Mechanical Turk (AMT) [AMT] is a commercial crowdsourcing platform provided by e-commerce company amazon.com. Its name derives from an 18<sup>th</sup> century chess-playing automaton that had a human operator hidden inside of it. To observers, it appeared as if the automaton was playing chess, while it actually was a human being. The same way, task uploaded to the AMT are apparently solved by these web services, while actually human contributors solve it behind the scenes; the web services act as a mere broker between the providers of the tasks and the humans contributors solving them. The reward contributors get for their work is strictly monetary. Due to the monetary reward system, contributing users cannot generally be assumed to be benevolent and motivated by interest in the task, so Assumption U1 does not hold here. AMT therefore offers task providers the option to refuse answers to tasks, denying the contributing user his reward and damaging his reputation. However, it is highly dependent on the individual task whether or not inferior answers are easy to detect automatically; if not, enforcing data quality may require considerable effort for reviewing the answers.

A further technical / legal problem is that only organizations based in the United States of America can have tasks completed by the AMT, though users from around the globe can contribute to solving them.

Eckert et al. [2010] used AMT to arrange terms into a concept hierarchy, having contributing users compare terms pair wise with regard to relatedness and relative generality, i.e. which of the terms was more specific or more general than the other. With some filtering (see next section), the quality of the data obtained from the contributions was comparable to a concept hierarchy constructed from the same terms by domain experts. Snow et al. [2008] report similar results for detail level NLP tasks like word sense disambiguation. However, it is unclear how these results translate to correcting NLP results in a semantic markup generation process because the number of parameters and degrees of freedom in these corrections may exceed the corresponding numbers for detail level NLP tasks by one or more orders of magnitude. Think of page structuring, i.e. identifying text blocks (in the sense of logical paragraphs) in OCR output and figuring out which ones of them belong to the document's main text, and which ones are captions, footnotes, page headings, etc. This is far more complex than specifying the relatedness of two terms on a 0-4 scale or their relative generality as one out of four categories.

### 6.6.2 Competitive Games

There have been several attempts to turn crowdsourced task into online games and have users compete for correct answers, with no motivation but the competition itself. Von Ahn successfully used this approach for image labeling in his **ESP Game** [Von Ahn 2006]. Siorpaes' **OntoGame** [Siorpaes 2007] showed that the gaming approach also works well for ontology construction and alignment, and for named entity disambiguation. The latter, however, was not done from the relatively small amount of context a named entity has in a document, but from a whole Wikipedia<sup>12</sup> article per named entity. Thus, it is questionable if results would be equally good if only the sentence or paragraph containing the named entity was given as the context. Furthermore, for the same reasons as argued above, it is unclear how these results translate to correcting NLP results in a semantic markup generation process, which can be far more complex.

### 6.6.3 Other Crowdsourcing Applications

Apart from competitive games, there are other attempts at crowdsourcing tasks on a volunteer basis, some browser based, some with special-purpose client software. The **GalaxyZoo** [Lintott 2008] project had over a million galaxy images classified into six basic categories by over 10.000 volunteers within a period of less than 200 days. However, a six-way classification again is by far less complex than corrections in a semantic markup generation process. **FoldIt** [Cooper 2010] has shown that crowdsourcing also works for more complex tasks to some degree. In this case it was protein folding, which is basically finding the three dimensional arrangement of a sequence of amino acids that has the lowest energy level, and thus the arrangement such a sequence will naturally assume. In all, the FoldIt project identified the structure for 208 proteins so far. However, of some 5.000 volunteers who contributed to FindIt only some 500 made contributions that were actually useful. This may serve as an indication of how rare good contributions are for complex tasks. While page structuring is surely less complex than protein folding and therefore a higher fraction of useful contributions can be expected, it remains unclear how high this fraction might be for NLP correction tasks like structuring scanned document pages. Furthermore, there are hundreds of thousands of digitized document pages for which semantic markup generation is desirable; processing this large amount of data is impossible to achieve with the very low throughput / high redundancy used in FoldIt. Another distributed data analysis project, if not exactly a crowdsourcing one, was **SETI@home** [Andersen 2002], in which roughly four million users contributed to analyzing radio signals from the depths of space in search for extraterrestrial intelligence. In this project, users contributed computing power rather than actual working time. Nevertheless, SETI@home is worth mentioning here because it shows how many volunteers a project can attract if its background has a sufficiently high appeal or "coolness factor" with the target audience. Arguably, GalaxyZoo and FoldIt also heavily benefit from this effect.

A successful crowdsourcing project related to the digitization of legacy literature is **Distributed Proofreaders** [Newby 2003], a part of Project Gutenberg<sup>13</sup>. The latter

---

<sup>12</sup> <http://www.wikipedia.org>

<sup>13</sup> <http://www.gutenberg.org>

aims at digitizing and thus preserving cultural work like novels and plays, but it does not generate any semantic markup, whatsoever. The sole purpose of Distributed Proofreaders is to correct OCR errors, which it achieves to a high degree by means of considerable redundancy. Tens of thousands of volunteers have managed to proofread the OCR output of more than 18.000 works over a period of roughly eight years, as of July 2010. Apart from the high redundancy, a main reason for this relatively low throughput may well be the user interface, which uses a plain text area on a web site and thus offers little visual or editing support for the users. Another reason may be that the appeal of cultural preservation is way lower than that of searching for extraterrestrial intelligence, at least among people who regularly use computers at times they are free to do volunteer work.

**ReCAPTCHA** [Von Ahn 2008], another crowdsourcing project related to the digitization of legacy documents, has a goal somewhat similar to Distributed Proofreaders, but departs from the volunteer approach. It builds on the CAPTCHA mechanism [VonAhn2003], which was originally designed to tell human users apart from web bots, e.g. in order to deny the latter access to certain parts of web pages. In particular, reCAPTCHA does the same as CAPTCHA, but instead of decrypting text from purpose-generated images, users transcribe two words from OCR output in each CAPTCHA. Thus, reCAPTCHA essentially forces users to contribute, even though in very small doses. ReCAPTCHA is a very popular security mechanism that is integrated in over 40.000 web pages [Von Ahn 2008] by means of JavaScript callbacks and has transcribed 440 million words, as of 2008 [Von Ahn 2008]. While very successful for word-level OCR correction, a mechanism akin to reCAPTCHA is less promising for corrections in a semantic markup generation process, as in the latter work packages are considerably larger and user interaction is considerably more complex than typing a few letters in a text field – again, consider page structuring, as argued above.

## **6.7 User Motivation & Data Quality Enforcement Mechanisms**

As seen in the previous section, ensuring data quality in crowdsourcing applications is difficult. This is especially true if the contributing users have to perform complex tasks, but even if complexity is limited, Assumption U3 cannot be ignored. FoldIt has shown that the more complex the task, the higher the number of accidental errors. In addition, if the contributing users' primary motivation is not interest in the subject, but a monetary reward, as in Amazon Mechanical Turk, Assumption U1 cannot be assumed to hold. This means that users might not bother investing the effort to complete their task to their best knowledge, but simply submit random answers to rake in the reward for free. Previous crowdsourcing and community-based projects have taken different measures to counter both accidental and cheating errors.

The individual error prevention and cheating deterrence mechanisms presented in this section will be formalized and subjected to in-depth mathematical analyses of their effectiveness in Chapter 10.

### **6.7.1 Countering Accidental Errors**

Accidental errors arise if users get distracted or do not pay sufficient attention when performing their assigned tasks, which can always happen according to Assumption

U3, or if they are generally not up to the work they do. Resulting errors are presumably rather random. The measure commonly employed to counter them is redundancy, i.e. having multiple contributing users perform the same task and combining their answers, e.g. by means of a majority vote. This approach was or is successfully used to varying extent in ESP Game, OntoGame, GalaxyZoo, FoldIt, Distributed Proofreaders, and reCAPTCHA, and also by Eckert [2010] and Snow [2008]. The level of redundancy is somewhat correlated to both the size and the complexity of the task contributing users perform, and on their expected motivation. Labeling tasks that consist of categorical decisions with a limited number of alternatives use moderate redundancy, assigning any given task to around 5 users; ESP Game, OntoGame, GalaxyZoo, Eckert [2010] and Snow [2008] all fall into this category. As opposed to this, the level of redundancy in large or highly complex tasks with many degrees of freedom in the answers is relatively high, as in FoldIt and Distributed Proofreaders, incurring a relatively low throughput. ReCAPTCHA is a different case: Although entering plain text as a transcript of some image provides many degrees of freedom, the size of the transcription task is very small (two words), and users are highly motivated, as they have to complete their task successfully to get what they want. As a consequence, reCAPTCHA achieves a transcription accuracy of well over 99% with only two- to threefold redundancy [Von Ahn 2008].

As argued before, corrections tasks in semantic markup generation processes for digitized legacy documents may be rather complex, e.g. page structuring. According to experience from previous projects discussed above, this would necessitate a rather high level of redundancy to ensure high data quality. However, the relatively low throughput of the Distributed Proofreaders project shows that too high a level of redundancy is unsuited for large-scale projects that aim at digitization of and extraction of information from large sets of documents. To really achieve Goal 9 and Goal 10 while still keeping throughput sufficiently high, data quality complying with Requirement O1 has to be achieved with a level of redundancy at most as high as in reCAPTCHA.

### 6.7.2 Countering Cheating

The danger of encountering cheating errors, i.e. errors that are not corrected by users because they do not bother to, is especially high in scenarios where Assumption U1 does not necessarily hold. This is the case when the contributing users' primary motivation is not interest in the data they help processing or a general attitude to help, but merely the reward on offer. The latter may be of the monetary sort, as in [Eckert 2010], or a sought service becoming accessible, as with reCAPTCHA, but also rising in rank in a social network, as in [Hütter 2008]<sup>14</sup>, etc.

These latter three works have taken various approaches to countering cheating errors. In [Eckert 2010], the tasks to perform consisted of twelve term pair, and contributing users were to rate each pair for relatedness and relative generality. In every task, the relatedness and relative generality was in advance known to the system for four of the term pairs; they were included to assess both attitude and expertise of

---

<sup>14</sup> This project did not do crowdsourcing in the general sense, as the people involved knew each other and formed a closed community. However, the mechanisms used to enforce data quality may be of interest.

the contributing users. The attitude tests were two term pairs whose relatedness and relative generality can be determined by anyone with common sense, so if users got them wrong, this was a good indicator for them not bothering to pay attention. The expertise tests, in turn, were term pairs whose relatedness and relative generality require profound knowledge of the application domain (philosophy in this case) to determine, with one term pair even selected to be misleading if judged by common sense alone. If users got these latter term pairs right, this was a good indicator for high quality answers. Eckert [2010] also considered the amount of time that users spend on a task as a predictor for data quality, based on the assumption that people who spend more time on a task perform it more thoroughly and therefore provide answers of higher quality on average. This assumption turned out wrong, however, as there was no significant correlation between the quality of the answers and the time users spent on producing them. ReCAPTCHA also uses known answers to assess whether a user does the transcription correctly; in particular, each pair of words a user is challenged with consists of one word whose transcription is not yet known, and one whose transcription is already known to the reCAPTCHA system. The transcription the user provides for the latter is the actual CAPTCHA, and at the same time indicates whether or not the system can expect the transcription for the former to be correct. [Hütter2008], in turn, had the members of their user community vote on each other's contributions to an ontology population effort and employed a complex mathematical mechanism to make sure users were acting truthfully.

Hiding parts with known answers within a task turns out to be a powerful measure to prevent cheating errors. However, this measure is hardly applicable in corrections for semantic markup generation processes: First, tasks are too complex to pair them up as reCAPTCHA does – think of structuring two pages instead of one in one task, for instance. Second, tasks are atomic, e.g. correcting the structure of one document page, so it is hard to insert sufficiently concealed parts with known answers as Eckert [2010] did. A voting scheme akin to the one used by Hütter [2008] is not applicable to semantic markup generation, either. In particular, such a scheme requires all contributing users to have access to the entirety of the data they process, so to rate the data inserted by their fellow contributors. In a multi-step semantic markup generation process, on the other hand, processing has to continue with the next step once the correction phase of a given step is complete, so contributing users only have access to the semantic markup they are tasked with to correct, and only for the duration of the correction phase of a particular step – afterward, corrections are of little use, as any undetected cheating errors already have propagated to subsequent steps the markup process.





## 7 Assisting Users in Semantic Markup Generation

To render interactive semantic markup generation, namely using NLP tools and correcting their output, as comfortable as possible for domain experts, several issues need addressing. A respective application has to integrate NLP tools in editing in a natural way, and it has to support users in manually editing the NLP results. This chapter first assesses how to integrate NLP tools and presents a general document displaying and editing concept intended to support corrections, thus to achieve Goal 1. Second, it presents visualization concepts for specific correction tasks as an approach to Goal 2. Finally, a controlled user experiment demonstrates the effectiveness of these concepts, including a questionnaire that assesses how the participants of the experiment perceived the concepts subjectively.

### 7.1 General Application Design

To generally support domain experts, i.e. to achieve Goal 1, an application for semantic markup generation has to provide several facilities: First, using NLP tools has to be a natural part of editing, just like any other function in an application. Second, the document has to be displayed in a way that optimally supports users in spotting and correcting the NLP errors in every step of a semantic markup generation process. As the markup generated in the individual steps is quite different in both granularity and purpose, the document display essentially has to be highly flexible. Third, editing XML has to be as little tedious as possible, especially with regard to handling the sophisticated and fragile XML syntax on character level.

#### 7.1.1 Encapsulation of NLP Tools

To hide the complexity of invoking NLP tools from the command line, that is to achieve this part of Goal 1, requires an application that seamlessly integrates them in the user interface just like any other editing function. This is also a prerequisite for achieving Goal 4, as only integration with the correction facilities allows for NLP tools to become aware of the users' corrections and possibly learn from them. Furthermore, it is desirable for an application for interactive semantic markup generation to not be bound to a specific suite of NLP tools or even to one of the common data models presented in Section 6.4.1. To the contrary, such an application needs to be very flexible and capable of integrating as many different NLP tools as possible in order to be useful for a multitude of different semantic markup generation tasks. These requirements have a series of implications, to be discussed in the following.

First, allowing for deploying as large a variety of NLP tools as possible requires for their individual implementation peculiarities to be encapsulated. This means that no individual NLP tool is to be a hard wired part of a semantic markup application. Instead, the NLP tools have to be integrated by means of a plug-in mechanism through a slim as possible interface to allow for easy integration of new ones. This interface has to manage the lifecycle of their wrapped NLP components, e.g. loading

and unloading, convert between the surrounding application's data model and the one of the wrapped NLP tool, and perform the invocation of the actual NLP functionality.

Second, the requirement to be compatible with the several data models used in existing NLP tools, as presented in Section 6.4.1, implies that the data model underlying a semantic markup generation application should be able to easily emulate them all; at least there has to be a lossless, yet simple bi-directional mapping to and from all of them, which implies a high lower bound in terms of expressiveness. Furthermore, to facilitate the removal of layout artifacts, e.g. to de-hyphenate words that run across line breaks, and to allow for the correction of OCR errors, such a data model also has to support editing of the document text.

Third, as it may turn out optimal to run a sequence of NLP tools as an atomic unit and correct only the final result (in exception to Assumption T2, as discussed there), a it is desirable for a semantic markup generation application to be capable of chaining several NLP tools together to be executable as an atomic unit, akin to the pipelining concept implemented in GATE.

### **7.1.2 Document Display**

A flexible and intuitive view of XML documents is essential for domain experts to be able to perform interactive semantic markup generation efficiently, specifically to render their effort in the correction phase of each step as low as possible. Thus in order to achieve the respective part of Goal 1, a semantic markup application has to provide a document display that optimally supports users in spotting and correcting errors both in structural and in detail level markup.

While semantic coloring has been known as a powerful and expressive means of visualizing semantic markup on the detail level, it is hardly suited to visualize markup that reflects structure. On the other hand, classical textual XML display is well suited for structural level markup, but due to its verbosity becomes very confusing in the presence of detail level markup, even with syntax highlighting and code folding. In addition, standard XML syntax highlighting does not provide the intuitive look of semantic coloring because the highlighting is on the character level and does not use colors that visualize the semantics of individual markup elements.

A promising visualization approach is a hybrid of semantic coloring and textual XML display, thus to integrate semantic coloring and its usual configurability, i.e. the option to show individual types of markup elements or not, with displaying XML in its textual syntax. In particular, the idea is to provide the option of showing or not showing tags for individual types of markup elements alongside the option of using highlights in the style of semantic coloring, and to color the tags semantically to make the visualization more intuitive. This approach has the advantage over code folding that the document text is always visible in its entirety and only the XML tags are shown or hidden.

In order to not mislead the domain experts' intuition in longer semantic markup generation projects, but to benefit it instead, the coloring scheme furthermore has to be persistent across restarts of a semantic markup application, as opposed to being randomly generated. This helps domain experts to understand the document display with the help of the colors, as they learn over time that, for instance, paragraphs are blue, footnotes are yellow, and bibliographic references are red. In addition, the coloring scheme has to be configurable so markup elements of similar granularity are

not assigned the same color, which increases clarity when displaying multiple elements of similar granularity alongside one another. The latter means, for instance, that for the result of semantic parsing to be visualized intuitively, the various parts of the parse (subject, predicate, object, adverbials, etc) have to be clearly distinct in the color they are highlighted with.

### **7.1.3 Editing Facilities for Document Text and Markup**

Because natural language processing generally treats words, or tokens in general, as atomic units, the elements in semantic XML markup always enclose whole words as well. Consequently, the corrections that users make refer to words as atomic units, and a user never selects parts of a word without selecting the entire word when correcting semantic markup elements. Therefore, a semantic markup application can relieve users from the tediousness of accurately selecting words on the character level by treating a word as selected in its entirety for a markup edit operation even if only parts of it are actually selected.

Furthermore, while editing the document text on the character level still has to be possible, e.g. for correcting OCR errors in digitized legacy documents, editing XML markup on the character level is not required. A semantic markup application can relieve users from this tedious and error prone task by generating the XML syntax automatically. Users then only have to select the word or sequence of words to mark, and to specify the type for the markup element to generate; the rest can be done automatically. This both prevents syntax level errors and simplifies the users' work.

For correcting existing markup that has been generated before, e.g. as markup of the document structure as recognized by OCR or semantic details marked by some NLP tool, a semantic markup application can further support users through higher-level assistance functions. Especially markup that reflects a document's structure often completely parquets a document, e.g. in that all text is included in paragraphs, which in turn are all nested in sections. Correcting such markup by removing erroneous elements and creating new ones in their place is more effort than necessary. Users can split or unite paragraphs, for instance, where markup generation missed actual paragraph boundaries or erroneously recognized them, respectively. This results in one user operation instead of three in either case, as splitting a paragraph or uniting two is equivalent to removing one and creating two or removing two and creating one, respectively. Another cumbersome task that automates easily through a respective assistance function is to convert text laid out for printing back into flowing text, i.e., to remove paragraph internal line breaks and to de-hyphenate word in the process.

Similar assistance is possible as well for correcting detail markup, e.g. the result of NER. Especially with NER components that make use gazetteer lists, it is quite likely that all occurrences of a given name have been missed throughout a document, or that there are several false positives marking the same string throughout a document. It is unnecessarily tedious for a user to find each and every single one of the errors in either case, especially in larger documents. Building upon the underlying assumption of all gazetteer-based NER, namely that within a single document, words and phrases are used unambiguously, assistance functions that help avoiding this effort are to mark all occurrences of a selected word or sequence of words in one operation, and to remove all markup elements of a given type that mark the occurrences of a given word or word sequence, respectively. With such assistance, users need to spot only

one instance of a repeated error and can correct all instances at once. In the converse case, i.e., that there are words or phrases used ambiguously, users still can resort to the more basic assistance functions that work on individual markup elements. However, the latter case is rather unlikely in scientific texts, as authors of scientific publications usually have a generic interest in using unambiguous formulations in their arguments. NER results reported in literature [e.g. Mikheev 1999] suggest that it is similarly rare in texts of other types, e.g. newswire.

## 7.2 Specialized Views for Specific Correction Tasks

In several semantic markup correction tasks, even a highly flexible view of an entire document may be harder than necessary to overview for users. Specifically, a semantic markup application can support users in such correction tasks by providing specialized views of a document.

When correcting NER results, for instance, users do not necessarily need to look through the entire document to find false positives, which is especially cumbersome in documents where named entities are sparse. Instead, a semantic markup application can provide a list view that displays all the markup elements marking a named entity in a concise fashion and thus make them easy to overview, and a user then can easily sort out false positives from such a list. Other applications for a list view are sorting out bibliographic references, captions, footnotes, etc. that were marked automatically beforehand, basically sorting out false positives from any NLP result that extracts and marks a small fraction of a document.

The correction of markup that reflects the structure of a document, in turn, benefits little from such a list view. A different kind of view is promising, however. To help a user focus on one page at a time, and thus also to reduce the risk for him to miss an error, it appears helpful to display a document in a page-by-page fashion, letting the user flip through the pages like through a book. Other applications of a one-by-one view of structural markup elements are correcting the subdivision of sections into sub sections, for instance, or correcting the result of shallow or full parsing in a sentence-by-sentence fashion.

## 7.3 The GAMTA Data Model

A data model that both is compatible with most of the existing NLP implementations and allows for changes to the document text without destroying existing markup is not readily available. Neither the widespread data models presented in Section 6.4.1 nor more advanced data models like LMNL [Tennison 2002] explicitly provide all of the required features. Namely, changes to the document text are a major problem, which is not surprising because most scientific NLP is intended for existing corpora of clean text, e.g. the Aquaint [Aquaint] corpus of newswire text.

To facilitate evaluating the concepts described in the previous sections in both laboratory experiments and field studies, we have designed and implemented the Generalized Annotation Model for Text Analysis (GAMTA, see Appendix A) as the basis for an application that support users in the interactive generation of semantic

markup for digitized legacy documents. The GAMTA data model essentially is a hybrid between the Tag-Array data model and the Spans data model that at the same time supports edit operations to the document text. GAMTA mainly treats the document text as a sequence of tokens, like the Tag-Array data model, as the spans that represent markup elements are anchored to token indices instead of character offsets to simplify and speed up adjustment to text level changes. But it can act as a character sequence as well, e.g. for actual character level editing operations, which are essential for document cleanup, or for regular expression pattern matching, which is widely used in NLP. The decision to integrate tokenization in the data model is based on the observation that it is the very first step to take in almost every NLP application anyways, and that almost all higher NLP algorithms work on tokenized text. Furthermore, similar to LMNL, the GAMTA data model supports interleaving markup elements, e.g. to temporarily represent both the layout and the logical structure of a document in parallel. Finally, to facilitate working with flexible views, documents can also be edited through the spans that represent the markup elements, as if the span was the entire document itself. This allows a semantic parser to work on individual sentences, for instance, or on the sentences in a single paragraph, no matter how many sentences or paragraphs are present in the backing document. Likewise, users can correct a semantic parse result one paragraph at a time, or markup that reflects the document structure one page at a time.

## 7.4 The GoldenGATE Editor

Based on the GAMTA data model, we have then implemented the NLP encapsulation and document visualization and editing concepts described above in a respective semantic markup application, dubbed the GoldenGATE Editor<sup>15</sup> (see Appendix B). This editor basically consists of two parts, namely the **Document Editor**, which provides the facilities for manual editing and correcting both document text and markup, and a **plug-in host** that allows for integrating NLP tools and various other extensions. The latter comprise visualization components that implement the concepts described in Section 7.2, adapters for various document storage formats, a component that facilitates pipelining of NLP tools and executing them together, and various other semantic markup generation and editing aids. The concepts implemented in the GoldenGATE Editor have been published in [Sautter 2007].

### 7.4.1 The Document Editor

A document editor (Figure 7.1) displays a single document and provides all the functionality required for manually editing both text and markup. It implements the various visualization aids and editing assistance mechanisms discussed in Section 7.1.

**Flexible Document Display.** If the number of tags in an XML document becomes too large, the document will not be concise any more, and readability suffers. Thus, the presentation of the document in the document editor is flexible to provide the

---

<sup>15</sup> The name is both a reference to GATE and a symbol for building a bridge between domain experts and NLP-assisted semantic markup. The current version of the GoldenGATE Editor is always available at <http://idaho.ipd.uka.de/GoldenGATE/>

appropriate level of detail for the current correction task. In the display control (see Figure 7.1, right of the document), a user may independently choose for every type of markup element that is present in the document to use semantic coloring for visualization, to display the tags, or not to show markup elements of a given type at all.

**Controlled XML Syntax Generation.** To handle the XML syntax manually on the character level is unnecessarily cumbersome for users and abets syntax errors. To counter this, the document editor only allows editing the tag content (the XML element name, and the names and values of attributes), but generates the syntax (the angle brackets, forward slashes of closing tags, escape syntax for attribute values) automatically and shields it from manual editing. Furthermore, it arranges the tags automatically to enforce wellformedness.

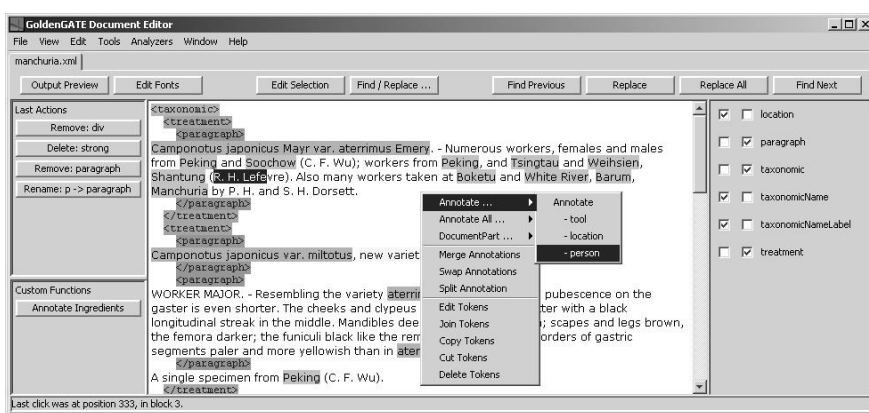


Figure 7.1: The annotation editor

**Markup Creation.** To mark a word or a sequence of words with an XML tag, the user can simply select the words in the document and use the *Annotate* function in the context menu. The document editor then prompts for the element name and does the rest automatically. To reduce the editing effort further, the context menu provides the most recently used element names for direct selection in a sub menu. Changing the name of an element works in a similar way, as the user does not need to modify start and end tag separately but simply enters the new element name in a respective prompt. Likewise, removing the markup elements around some text is an atomic operation, with the start and end tag removed automatically. In addition, marked document fragment can be atomically deleted together with their content, i.e., the element's tags, the enclosed text, and any nested markup elements as well.

**Global Markup Editing.** Creating, modifying, and removing XML tags often applies to all elements of a certain type. Thus, the document editor offers respective assistance functionality, e.g. for renaming all markup elements with a certain name, or for removing them with or without their textual content and possibly nested markup elements. This helps users with removing the `font` tags from HTML-formatted OCR output, for instance. When marking a word or sequence of words with an XML tag, however, the mechanism depends on textual content rather than names of markup elements, as the user can choose to mark all occurrences of the selected phrase throughout the document instead of just the one currently selected. This facilitates

marking all the mentions of a named entity in a document with just a single operation, for instance.

**OCR Cleanup:** After digitization and OCT, legacy documents often include parts that do not belong to the actual document text, but originate from print layout, like page numbers and page headers. A further problem are line breaks that do not mark the end of paragraphs, but also originate from the print layout, possibly with words hyphenated across them. Especially hyphenated words can severely compromise the accuracy of NLP results, but manual de-hyphenation is very cumbersome. Therefore, the document editor provides a function removing paragraph-internal line breaks and de-hyphenating words along the way. Automating the latter requires some attention, however, in order to not destroy enumerations that use word pre- or suffixes for abbreviations.

#### 7.4.2 The Extension Plug-In Host

To simplify deployment and evaluation of assistance mechanisms for interactive semantic markup generation, the GoldenGATE Editor provides a unified extension interface for integrating respective components. This interface manages the lifecycle of the plugged-in components and integrates them in the document editor by means of dedicated mounting points; the plug-in host further provides a centralized registry for the plug-in components to enable them to interact with each other. The components integrated through this interface can provide, among others, specialized document visualization functionality, adapters for various document storage formats and locations, and NLP tools. There are several extension components worth a special mention here; refer to Appendix B for a more comprehensive overview.

The **Analyzer Manager** integrates NLP tools through a subordinate extension interface. In order to be able to integrate a wide variety of existing NLP tools, this plug-in uses lightweight wrappers that manage the lifecycle of the actual NLP tools and convert between the GAMTA data model and the data model the NLP tools are implemented to work with.

To simplify the correction of NLP errors, the two dedicated plug-in components provide special visualizations of the document that help reviewing the NLP results. In particular, the **List Viewer** (Figure 7.2) can concisely display all markup elements that match a given XPath expression in a list, e.g. all markup elements with a specific name. With this view, a user can inspect all these markup elements without having to search them, and he can choose which ones to keep because they are correct, and which ones to remove because they are false positives, e.g. when correcting an NER result. Respective task-specific XPath expressions can be pre-configured, e.g. by an administrator, so to relieve domain experts from the burden of handling the complex XPath syntax themselves. The **Slide Viewer**, in turn, displays specific markup elements one by one, with these markup elements also being selected by an XPath expression. This helps a user with going through a document page by page and correcting structural markup, for instance. Again, the XPath expressions can be pre-configured, for the same reason as with the List View.

Some other extension plug-in components are worth mentioning: The **Pipeline Manager** provides sequenced atomic execution of selected NLP tools and other semantic markup generation aids, adopting the pipeline concept of GATE. Further plug-ins natively provide basic NLP functionality like gazetteer **Lists** and **Regular**

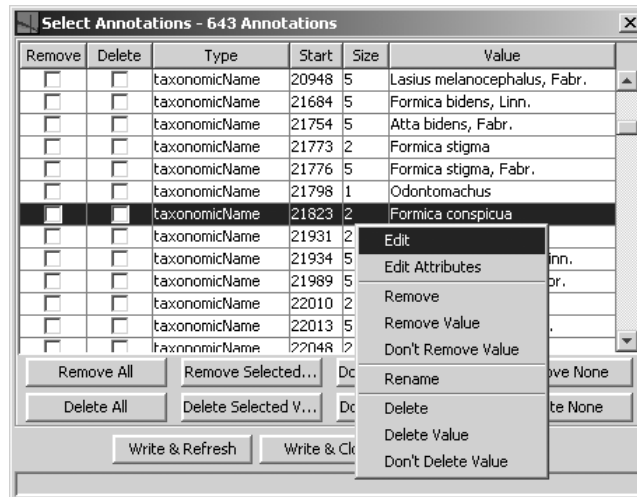


Figure 7.2: List view of XML elements for correcting detail markup

**Expression** patterns, which both can be applied for annotating a text document automatically. This is to overcome the need for integrating heavyweight external components for lightweight semantic markup generation tasks.

All facilities for automated semantic markup generation and document visualization can be configured to be one-click accessible in the document editor as **Custom Functions**; the buttons on the left of the document in Figure 7.1 are the means to access them. This saves users the effort of going through menus when accessing the functions most important for the current task.

## 7.5 Evaluation

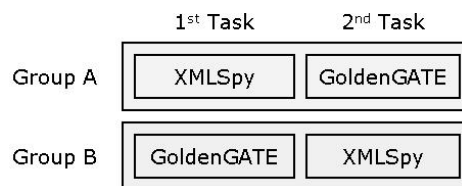
To empirically assess whether the assistance concepts presented in this chapter support users better in semantic markup generation for digitized legacy documents than off-the-shelf XML editors, we have conducted a controlled laboratory experiment. In this experiment, participants were tasked with generating semantic XML markup for two digitized legacy documents, with the markup defined by a dedicated XML Schema. The independent variable, also referred to as the experimental condition, was the application the participants worked with, either the GoldenGATE Editor, configured with a few task specific custom functions (see Section 7.5.5 for details), or XML Spy. The dependent variable we measured was the time it took the participants to complete their tasks. The experimental setup was designed to control all other variables that might have any influence on the participants' markup generation performance. The experiment and its results have been published in [Sautter 2007a]. To collect subjective first-hand user impressions, we additionally had the participants fill in a questionnaire after they had completed their tasks.



### 7.5.1 Experimental Design

In order to achieve sufficient statistical power (see Section 7.5.6), we needed about 10 data points for *each* markup application. Because we could not expect to attract more than 15 voluntary participants for our experiment, and because experience shows that not all volunteers actually show up, it was necessary to design the experiment such that every participant would contribute *two* data points, one for each editor.

Every participant in the experiment worked on two different, but equivalent tasks (see 7.5.5), using XML Spy for one task and the GoldenGATE Editor for the other. When exposing each participant to both experimental conditions (i.e., usage of both editors), there is a general risk of a sequencing effect, i.e., that the variation of the independent variable is not the only cause for an observed effect, but that the order in which the conditions were applied has an influence as well. There are two important sequencing effects that might affect our experiment: An increased familiarity with the markup task, the document structure, and the experimental environment after completing the first task can possibly have a positive impact on the performance in the second task (learning effect). Being asked to use the "old" XML Spy editor in the second task after using the "more comfortable" GoldenGATE Editor in the first one can possibly have a negative impact on motivation and performance (motivation effect).



**Figure 7.3: Counterbalanced experimental design**

To make sure that any conclusions about possible performance advantages of the GoldenGATE Editor are valid, it is mandatory to select a proper design which allows for controlling sequencing effects. We applied a *counterbalanced* design [Christensen 2007]: One half of the participants used XML Spy for the first task and the GoldenGATE Editor for the second one (Group A); the other half of the participants used the editors in the opposite order (Group B), as graphed in Figure 7.3. To level differences in individual abilities, we *randomized* the assignment of participants to groups.

### 7.5.2 Pilot Study

In a laboratory experiment, a pilot study helps validating the experimental setup, i.e., the environment and the material. A pilot study also helps estimating the size of the effect to be observed in the experiment, a number required for determining the number of data points needed for the experiment to have a meaningful result (see Section 7.4.6). In February 2007, we conducted a pilot study with about half a dozen student volunteers as participants and excerpts of documents from the biosystematics scenario (see Section 4.1) as the tasks. The day before the pilot study, we offered a half-day tutorial for the participants, covering the features of XML Spy and the GoldenGATE Editor, and the structure of biosystematics documents. The main focus

in the tutorial was hands-on work with both editors, so for the participants to become familiar with using them.

In the experimental tasks, some participants used XML Spy to create semantic markup for their document, others used the GoldenGATE Editor. The pilot study revealed several problems with setup and material: (1) Despite the training, the participants showed a lack of proficiency using the more advanced features of the GoldenGATE Editor. (2) The tutorial turned out to be too short for the participants to acquire sufficient domain knowledge regarding the structure and contents of the biosystematics articles, so it took them considerable time to recognize the relevant parts of the documents. (3) Finally, the experimental tasks proved too long, as participants became tired before the tasks were finished. Consequently, we adjusted the tutorial contents and the material for the main experiment as a consequence.

### **7.5.3 Tutorial**

We offered an extended tutorial one day in March 2007 and carried out the experiment on the next day. Due to the experiences from the pilot study, we extended the practical exercises covering the features of the GoldenGATE Editor, but we still covered XML Spy as well to make sure that the participants had the same degree of familiarity with both editors.

For both the tutorial and the experimental tasks, we used documents from the substitute pasta recipe scenario (see Section 4.2) this time instead of the biosystematics documents. The rationale behind this decision was that cooking is a generic domain many possible participants are familiar with, so they could immediately understand respective documents. In the end of the tutorial, we held a competition in which we asked the participants to generate semantic markup for a document as quickly as possible using the GoldenGATE Editor. The rationale was to see how individuals use this application when working under pressure, and our observations showed that the prospective participants were sufficiently familiar it.

### **7.5.4 Participants**

12 computer science graduate students volunteered to participate in the tutorial and the experiment; 2 of them were no-shows who attended the tutorial, but did not show up for the experiment, and 1 dropout who gave up after having worked on the first task of the experiment for more than 2.5 hours. Consequently, a total of 9 participants in the experiment contributed data points. The majority of these students were in their 7th semester; the others were more senior, up to their 13th semester. All of them had taken a graduate level database class this semester, a class that also covered XML.

We handed out a pre-test questionnaire at the beginning of the tutorial and therein asked for the students' knowledge of XML, their practical experience with editing XML documents using an XML editor, and their practical experience with correcting errors in digitized documents by hand. Except for two students who had used XML Spy on and off in the past, the pre-test questionnaire did not reveal any participant to have any special capabilities relevant for the experiment.

### 7.5.5 Tasks

For the experimental tasks, we used documents from the cooking scenario (see Section 4.2), which were mainly for pasta dishes. We made sure that the two documents were about equal in terms of difficulty, and that they had about the same length, namely 12 pages and 20 recipes. The participants easily understood the structure and contents of the recipes, which was important because we wanted to measure the speed advantages resulting from the features of the GoldenGATE Editor and not the time it took participants to understand the problem domain or document content.

The descriptions of the two experimental tasks were identical, except for the name of the document and name of the application to use for generating semantic markup for them. The participants were tasked with adding markup elements to structure the document into recipes, and the recipes into title, ingredient list, and a step-by-step description of the actual preparation. In the individual preparation steps, further markup elements were to identify individual ingredients and cooking tools. Furthermore, the participants had to remove artifacts typical for digitized legacy documents, including page headings and incorrect line and page breaks, and they had to correct misspelled words, which are typical for OCR results. This last requirement is particularly tedious when using XML Spy, and hence we relaxed it during the experiment for the XML Spy users in order to prevent them from dropping out in scores. Note that if this measure had any influence on the experimental results, it was in favor of XML Spy and in disfavor of the GoldenGATE Editor.

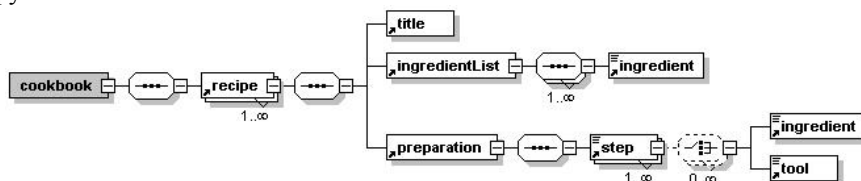


Figure 7.4: XML schema for the experiment

For the tasks in which they had to use XML Spy, users were given a schema (see Figure 7.4, a repeat of Figure 4.2 included for readability) to help them with the markup. The only NLP functionality that we made available in the GoldenGATE Editor used in the experiment was an automated tagger for ingredients and the function for normalizing paragraphs. Thus, if the GoldenGATE Editor would turn out superior already in this basic setup with very limited NLP support, this would allow us to validly expect this application to be better in ‘real’ settings with more NLP support as well, and even more so. Other features that we expected to be particularly useful for the experimental tasks are the list of most recently used annotations and the function that in one operation adds markup elements for all occurrences of a given word or sequence of words.

### 7.5.6 Sample Size

In the planning phase of the experiment, we performed a *power analysis* [Cohen 1988] to estimate the number of data points required to achieve statistically meaningful results. We first chose a significance level of 5% and a desired power [Cohen 1988] of 80%, and then estimated the effect size for a t-test by considering the expected overlap of the completion time distributions for XML Spy and the

GoldenGATE Editor. The data from the pilot study lead us to expect a large performance advantage of the GoldenGATE Editor, and thus we assumed that there would be only a small overlap of the time distributions, namely one of just 10%, which maps to an effect size [Cohen 1988] of 1.3 for the t-test. Given a significance level of 5% and an effect size of 1.3, a power of 80% maps to a requirement of 8.3 data points in each experimental condition (i.e., for each editor) for a one-sided t-test [Cohen1988]. Similarly, the desired power of 80% maps to a requirement of 9.6 data points for each of the two applications for a one-sided Wilcoxon test. Consequently, we needed to collect between 8 and 10 data points per application in the experiment.

### 7.5.7 Document Quality

For measuring the time it takes participants to complete their tasks as the dependent variable, it is utmost important to make sure that the output of the experimental tasks has a uniform (and minimum) quality; otherwise, short completion times could simply correlate with low or even unacceptable output quality. We therefore defined the following thresholds for the minimum correctness of the final document: 100% correctness for the structural markup, which comprises the individual recipes, their titles, the ingredient lists, the preparation, and the individual steps of the latter, and 85% correctness for the semantic markup, which comprises individual ingredients and cooking tools.

To assess the correctness of the semantic markup generated by the users, we set up a respective test server which compares the semantic markup in uploaded documents to semantic markup in respective gold standard documents. Because testing had very low overhead, we encouraged the participants to freely use the test server for acceptance testing during the experiment. Participants were finished with their task only after having passed the full acceptance test, which required meeting all correctness thresholds and thus implied that participants had worked on all parts of the task successfully.

### 7.5.8 Results

In all, we managed to acquire 9 valid data points for each editor. For all but one participant, the time it took to complete a task was *significantly smaller* when using the GoldenGATE Editor than when using XML Spy (see Figure 7.5). The mean of the task completion times with XML Spy is 107 minutes; whereas the mean with the

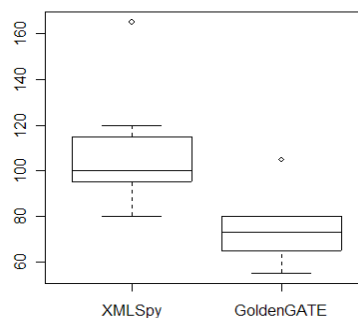


Figure 7.5: Boxplot of the completion time distributions

GoldenGATE Editor is only 77 minutes, which corresponds to an average relative speed-up of 25% when using the GoldenGATE Editor. The performance advantage of the GoldenGATE Editor over XML Spy is statistically significant at the 2% level, with a p-value  $< 0.013$  for the paired t-test and a p-value  $< 0.004$  for the paired Wilcoxon test. Thus, the experiment provides strong empirical evidence that the GoldenGATE Editor supports users better in semantic markup generation tasks for digitized legacy documents than a standard XML editor, such as XML Spy.

On average, the time it took users to complete the first task was 102 minutes, and thus was longer than for the second task, which was 82 minutes. Obviously, there was a *learning effect* between the two tasks, but this does *not* invalidate our findings because the learning effect applied uniformly to both editors: For XML Spy, the mean task completion time decreased from 117 minutes in Group A to 97 minutes in Group B between the two tasks; for the GoldenGATE Editor, it decreased from 84 minutes in Group B to 72 minutes in Group A. These differences were visible, but not statistically significant because p-values were larger than 0.11 and 0.19, respectively. Note that this analysis would not have been possible without a counterbalanced design.

When comparing XML Spy to the GoldenGATE Editor for the first task only, the performance difference is significant at the 6% level; similarly, when comparing the editors for the second task only, the difference is significant at the 2% level. This proves that the performance advantage of the GoldenGATE Editor over XML Spy is independent of the order in which the editors were used.

In one exceptional case, a participant was slightly faster when using XML Spy than when using the GoldenGATE Editor. What possibly explains this is that this participant had stated in the pre-test questionnaire that he had used XML Spy on and off prior to the tutorial. In addition, he used the GoldenGATE Editor in the first task so the learning effect between the two tasks is likely to have aggravated the observed effect.

From the means and variances of the completion time distributions, we furthermore computed [Cohen 1988] an observed effect size of 1.43. Given a significance level of 5%, the experiment has a post-hoc power of 89% for the t-test and of 77% for the Wilcoxon test. Thus the experiment had a satisfactory power even with fewer data points than originally planned.

### 7.5.9 Questionnaire

After the participants had completed both of their tasks, we had them fill in a post-test questionnaire (see Appendix C for details) to also collect their subjective impressions of the usefulness and usability of the NLP functionality and editing assistance features implemented in the GoldenGATE Editor. There were two general questions, one for an overall grade for the editor on a 1-6 scale and one for the difficulties using the editor on a 1-3 scale, with 1 being the best grade. Further, 19 detail questions regarding the access to and usage of individual visualization and editing assistance functionality asked for respective grades on a 4-1 scale, with 4 being the best grade.

The overall grade for the GoldenGATE Editor was 2.2, which on a 1-6 scale corresponds to an approval of about 76%; the overall difficulty of using it came out at 2.0, which on a 1-3 scale corresponds to 50% of approval. The answers to the detail questions were rather positive in general, especially for the markup generation functionality, see Appendix C for the individual questions and average answers. The

points the participants criticized were the support for editing the document text, especially for correcting OCR errors, and to some degree the immediate intuitiveness of the user interface for new inexperienced users.

A very interesting phenomenon we observed across all questions was that the grades from the Group A, who had worked with XML Spy first, were a lot more critical than the ones from Group B, who had worked with the GoldenGATE Editor first. A psychological reason for this tendency might be that the participants in Group B were more motivated and generally in a better mood when using the GoldenGATE Editor because they did not already have some two hours of work with XML Spy behind them, as opposed to Group A. However, observations regarding how the participants in Group B worked on their tasks and how they tackled specific problems reveal other reasons to be more likely. In particular, some of the participants who worked with XML Spy first found sophisticated ways of using that editor's native Find/Replace and Copy/Cut/Paste functionality to emulate markup editing assistance functions to some degree. When they later worked with the GoldenGATE Editor, these participants were somewhat alienated because these solutions did not work as used to any more. However, in general the answers of the participants from Group B were positives, if not as positive as those from the participants from Group A.

## **7.6 Discussion**

The laboratory experiment presented in the last section has clearly shown that in the generation of semantic markup for digitized legacy documents users greatly benefit from the advanced assistance concepts presented in Section 7.1 and Section 7.2, namely both from flexible visualization and from markup editing assistance mechanisms. The answers the participants gave in a post-test questionnaire indicate that they subjectively perceived these concepts as mostly helpful as well, only criticizing the editing functionality for the document text and, to a lesser degree, the immediate intuitiveness of the user interface. While these perceived weaknesses do not undermine the theoretical findings regarding the benefit of the user assistance concepts presented in this section as viable approaches to Goal 1 and Goal 2, they require addressing anyways because of Assumption U2 and Assumption U4. Namely, when remaining unaddressed, the weaknesses could take negative influence on the evaluation of further user assistance concepts, especially by scaring off or unnerving otherwise sympathetic users in longer-running field studies.

## 8 Optimizing Interactive Semantic Markup Generation

With the general concept of NLP in combination with assisted manual correction proven beneficial under laboratory conditions, the next step towards rendering semantic markup generation for digitized legacy documents as efficient as possible is to optimize semantic markup generation towards minimal overall correction effort. This comprises optimization of both the individual markup generation steps and the order of those steps within respective semantic markup generation processes.

The number of degrees of freedom in such an optimization is far beyond what can be covered in a laboratory experiment of reasonable scale, as such an experiment would require testing all possible orders of steps, which is the factorial of their number - a number of setup alternatives impossible to cover for processes with ten or more steps. Even if quite large a fraction of the alternatives could be ruled out from the start based on logical considerations, it would be practically impossible to cover the number of setups that would remain to compare experimentally. An aggravating factor is that the difference between some of the alternatives might well turn out rather small, so the overall number of data points required for a statistically significant mutual comparison of all alternatives would be prohibitively high.

A more promising approach for the optimization endeavor was to iteratively optimize both the individual semantic markup generation steps and their order in a longer running project that would allow for successively testing and comparing alternatives, and also for incorporating ideas of experienced users. We therefore conducted a field study accompanying a real-world semantic markup generation projects in the biosystematics scenario (see Section 4.1). A description of this field study, its results, and the gathered experiences has been published in [Sautter 2009].

This chapter first reports on the on the general setting of the semantic markup generation project, the Madagascar Corpus Project, which was the subject of the field study, on concrete optimizations developed in its course, and on the improvements achieved through these optimizations. This includes a description of the TaxonX Process as an example of a real-world semantic markup generation process; it was developed and optimized in the course of the project is optimized for Goal 5. The chapter closes with a thorough investigation into how to design and configure NLP algorithms for use in interactive semantic markup generation processes, generalizing and refining the step level optimizations into general approaches to Goal 3 and Goal 4.

### 8.1 The Madagascar Corpus Project

The Madagascar Project started in January 2007, and two biologists worked on it for more than a year. It has generated semantic markup for all publications on the ant fauna of Madagascar, thus the name of the project; the corpus of literature it worked on is dubbed the Madagascar Corpus for the same reason. The semantic markup generated for this corpus follows the TaxonX Schema [Catapano 2006], a dedicated XML Schema for the biosystematics domain.

### 8.1.1 The Madagascar Corpus

The Madagascar Corpus consists of all existing publications on the ant fauna of Madagascar, comprising 119 documents with a total of about 2,500 pages with roughly 1,000,000 words, their most important parts being the more than 4,000 taxonomic treatments (see Section 4.1 for an explanation of treatments). The 119 documents have been written and published over the course of the last 250 years in a multitude of different journals and books, and thus they show a broad variety in layout and style. Furthermore, the documents are written in five different languages, namely English (32 documents), French (54), Italian (5), German (16), and Latin (12).

This high degree of diversity poses a serious challenge for NLP algorithms and thus makes the effort to expect for manual correction relatively high. The Madagascar Corpus is thus an ideal proving ground for optimizations to semantic markup generation facilities; in particular, findings can be expected to have a very high level of generality because there are no favorable conditions around that might not be present in other corpora and therefore limit generality if exploited accidentally or deliberately.

Furthermore, the Madagascar Corpus has a number of characteristics that make semantic markup generation highly complex and a respective project a highly ambitious endeavor: (1) In the original printed documents, the logical reading order of the text is not always identical to the physical layout order, which necessitates restructuring the text, i.e. transforming the latter order into the former. In particular, individual treatments must not interleave with one another or with other parts of a document. (2) The text has to be freed from print layout artifacts, like page headers and page numbers, and interspersed additions like captions and footnotes have to be un-nested from logical main text paragraphs so they do not interrupt the logical text flow. (3) The individual treatments must be marked to become accessible individually. (4) Domain specific details like taxonomic names and locations in this current context need to be marked and disambiguated. (5) Abbreviations in named entities must be resolved to their full meaning because while their meaning is clear in the context of the surrounding document as a whole, it might not be in the context of a treatment in isolation.

These challenges, including the complexity of the TaxonX compliant semantic markup to generate, result in a relatively complex markup generation process, so there are no project specific favorable conditions in this regard either. To the contrary, semantic markup generation processes used in other projects might rather turn out less complex than the process in this project, e.g. when working on MEDLINE [MEDLINE] abstracts rather than digitized legacy documents of up to 100 or more pages.

### 8.1.2 The Applications

The initial OCR was done using Versions 8 and 9 of **ABBYY FineReader** [ABBYY], a commercial scanning and OCR application, with a specific setup tuned to cope with the challenges of the Madagascar Corpus: Because the documents are written in different languages, sometimes even with several languages occurring in one document, FineReader was set to use “multiple languages” mode, which makes it consider more than one language for the spell check. The setup also included two purpose-built domain specific dictionaries, one containing parts of known taxonomic names, and one with technical terms that frequently occur in morphological descriptions. Further, the setup included 15 so-called user patterns, i.e., trained confi-



gurations for a given layout style or font, which help processing old documents with uncommon layouts on which the built-in type recognition would fail.

The platform for the actual semantic markup generation, we again used the **GoldenGATE Editor** (see Section 7.4.2), incorporating some improvements in reaction to the criticism from laboratory experiment (see Section 7.5.9). In particular, the implemented changes were to simplify editing of the document text, e.g. for correcting OCR errors, and a cleaner and less cramped user interface intended to be easier to grasp.

## 8.2 The Markup Steps

This section describes the markup steps that had to be performed for each and every single document in the Madagascar Corpus, grouped into document cleanup and normalization, structural markup generation, and detail markup generation. This order is not the optimal one to perform the steps, as we will see, but it eases presentation; the optimization of the steps' order will be addressed later. Note that the steps presented here are logical ones that can well be implemented as a sequence of several NLP algorithms on the technical level. Unless mentioned otherwise, the automated semantic markup generation tools mentioned in this section are plug-ins to the GoldenGATE Editor.

### 8.2.1 Document Cleanup & Normalization

The steps for OCR error correction, layout artifact removal and document normalization are not specific to the Madagascar Corpus Project; to the contrary, they are inevitably part of any effort that generates semantic markup for digitized legacy documents.

**Layout Artifact Detection.** To prevent errors in later steps the structure of the text has to be intact. However, in printed text paragraphs can be broken for two reasons: (1) Paragraphs can run over page breaks, with page headers and page number interrupting the text flow, and possibly also with footnotes next to page breaks. (2) Paragraphs can have figures or tables embedded in them, with associated captions that also interrupt the text flow. To recognize insertions, a dedicated tool first ensures that pages are marked; other tools then search for page headers and page numbers at the top and at the bottom of the pages, and at the latter for footnotes as well. One of them makes the page numbers into attributes of the paragraphs of the page to facilitate page level citation.

**Structural Normalization.** Another step is cleaning up print layout artifacts. Deleting page headers is unproblematic because they do not contain any information that belongs to the actual document content, but footnotes and captions are another matter because they actually do represent content. Thus they stay in place if they do not interrupt the text flow of another paragraph; otherwise, a dedicated tool moves them to the position behind the main text paragraph they interrupt. While this somewhat changes the structure of the text, the alienation is not significant: Footnotes are usually located right above a page break, independent of the part of the main text actually referencing them, and captions stand above or below the figures, tables, etc. they belong to, with the exact position of the latter typically depending on layout

considerations as well. Structural normalization also removes the page markup, which is not needed any longer and might even interleave with logical paragraphs that ran across page breaks before this step.

**Paragraph-Border Correction.** Paragraph borders are sometimes erroneous in OCR output because of varying indentation and spacing schemes; bibliographies are especially challenging in this regard, for instance, because they tend to use hanging instead of normal indentation for paragraph starts. The result is that a paragraph may be split in the middle, or two or more paragraphs may be marked as one. Respective checks can be automated to some degree, e.g., based on capitalization or punctuation. But these checks can only point the user to likely errors because this evidence tends to be too unreliable for automated correction. Before removing recognized artifacts, like page titles and footnotes, a manual check into which paragraphs are artifacts and which are not is indispensable in order to avoid messing up the logical text order. A good document view for this correction task is the Slide View (see Section 7.4.2), which lets users flip through a document page by page, just like skimming a book.

**Paragraph Normalization.** When the paragraph boundaries are in place, the respective function (see Section 7.4.1) removes line breaks from within the paragraphs and de-hyphenates words in the process; the latter means that ‘fau-<line break>na’ becomes ‘fauna’, for instance.

**MODS Referencing.** To ease bibliographic referencing and provide a unique identifier for the XML documents, this step imports respective metadata [MODS], using the MODS data format<sup>16</sup>. A respective plug-in automatically retrieves the metadata from a web service<sup>17</sup> and inserts it at the start of the document; it also sets respective attributes.

## 8.2.2 Structural Markup

The semantic markup generated for the Madagascar Corpus comprises two levels of document structure, namely the treatments and other sections, and the inner structure of the treatments.

**Treatment Markup.** Once a document is structurally clean, a respective tool helps users with marking the sections, most prominently among them the treatments. As in other documents, the sections consist of one or more paragraphs each, and the tool exploits this by using rules that group the paragraphs into sections. To simplify correction of the section boundaries, in a dedicated document view users can visually check whether or not the paragraphs are correctly grouped and can correct errors.

**Structure of Treatments.** Treatments usually consist of several subsections (see Section 4.1 for a detailed explanation), which consist of one or more paragraphs each, just like the treatments themselves. Exploiting this, a rule-based tool groups the paragraphs of each treatment into subsections, and a dedicated document view helps the user with visually checking and correcting the grouping, just as in the step generating the markup for treatments and other sections.

---

<sup>16</sup> Metadata Object Description Schema

<sup>17</sup> [http://atbi.biosci.ohio-state.edu:210/hymenoptera/hym\\_utilities.format\\_ref?style=MODS&id=<documentIdNumber>](http://atbi.biosci.ohio-state.edu:210/hymenoptera/hym_utilities.format_ref?style=MODS&id=<documentIdNumber>)

### 8.2.3 Detail Markup

The semantic markup generated for the Madagascar Corpus comprises two types of semantic details, namely the taxonomic names and locations where referenced specimens have been collected. Both are normalized for disambiguation, the former by filling in abbreviations and adding LSIDs [Brazma 2006], the latter by geo-referencing. The taxonomic name markup is somewhat specific to the Madagascar Corpus, or to biosystematics in general, but marking and geo-referencing location names is not.

**Taxonomic Name Markup.** The taxonomic names are the most important details in the documents of the Madagascar Corpus; a dedicated algorithm [Sautter2006] detects and marks them automatically. Next is to normalize the taxonomic names, i.e., to add the ‘meaningful’ parts or taxonomic epithets of the names as attributes to the XML elements marking them. The taxonomic epithets are the parts of a name that carry the actual taxonomic information, as opposed to those that clarify naming authority or provide ranking information for an epithet. In the taxonomic name “*Drosophila melanogaster* Meigen”, for instance, “*Drosophila*” is the genus, “*melanogaster*” the species, and “Meigen” the author of the species name “*melanogaster*”; the author of the genus name “*Drosophila*” is “Fallén” and is not given here. For semantic markup, only the taxonomic epithets “*Drosophila*” and “*melanogaster*” are interesting, i.e., the parts that specify the position of a taxon in the tree of life. This also helps with matching taxonomic names from different documents because apart from the actual taxonomic epithets, the way taxonomic names are given in legacy documents varies wildly. A respective extraction tool parses out the mentioned parts and adds them as attributes named genus and species to the XML element marking the taxonomic name; it does not make an attribute out of “Meigen”. This process is referred to as taxonomic name atomization in domain biosystematics.

Because taxonomic epithets may be abbreviated or even omitted completely in the documents, the same tool also fills in resulting gaps and resolves abbreviations in the context of the surrounding document, a process known as taxonomic name reconciliation. “*Drosophila melanogaster*”, for instance, may be referred to as “*D. melanogaster*” later in the same document, with the meaning of the abbreviation “*D.*” being obvious to mean “*Drosophila*” in the context of the entire document, but not necessarily for a treatment in isolation.

After all taxonomic names are unambiguous on the syntax level (i.e., there are no more gaps or unresolved abbreviations), another tool links them to existing biosystematics nomenclature authority databases, so-called LSID providers. In particular, the tool retrieved the Life Science Identifiers (LSIDs, [Brazma 2006]) from the Hymenoptera Name Server [HNS] in the Madagascar Corpus Project. If a taxonomic name was not yet in the HNS database, i.e., if there existed no LSID for it, the tool uploaded it to the server and got a newly generated LSID in return.

**Specimen Location Markup.** The second most important detail information in the documents of the Madagascar Corpus is the locations where specimens have been collected. A somewhat tuned variant of standard NER techniques [Cucerzan 1999, Mikheev 1999] marks them automatically. To give the locations a unified unambiguous representation, a further tool adds their geographical longitude and latitude as attributes. It obtains this information from the GeoNames web service [GeoNames] and prompts the user for selecting the correct coordinate pair from a respective list whenever a location name is ambiguous.

## 8.3 Optimization of Markup Steps

The effort users spend in the correction phase of each step in a semantic markup generation process does not only depend on the accuracy of the NLP tool that generates semantic markup in the automated phase of the step; it also depends on the types of errors in the markup this tool generates, and on the assistance the user has available for the corrections. The field study that accompanied the Madagascar Corpus Project has revealed several optimizations the optimization both of NLP tools and of the editing facilities for correcting their output. This section first provides a classification scheme for the NLP tasks encountered in semantic markup generation processes, illustrated with both generic examples and examples from the Madagascar Corpus Project. Afterward, it describes the optimizations that helped with improving efficiency over the course of the latter project. The section closes with a generalization of the findings and observations.

### 8.3.1 Markup Task Classification

In general, there are four classes of NLP tasks that can be part of semantic markup generation processes; this classification explicitly excludes extremely high level NLP tasks<sup>18</sup> like Information Extraction as an atomic task, Automated Summarization [Endres 1998], and Question Answering [Hirschman 2001]:

**Classification Tasks.** NLP tasks that fall into this class mostly deal with choosing the appropriate category for all existing markup elements of a specific type from a given list. There are many examples of this class, for instance: (1) Almost every NER algorithm works in the two stage approach of first finding and then classifying proper names; the second stage is probably the most prominent instance of a classification task. (2) Generating markup elements that represent the structure of a document easily models as classifying individual paragraphs into ones that continue a section, and ones that start a new section of a specific type. (3) The inner structure of treatments generates best by classifying paragraphs into the possible types of subsections listed in Table 4.1. (4) POS tagging classifies the tokens of a text.

**Extraction Tasks.** Extraction tasks are NLP tasks that extract specific parts from a document and generate respective markup elements; examples of this class are manifold: (1) Every NER task includes an instance of this class, at least the extraction of proper names for later classification; in the Madagascar Corpus Project, the recognition of taxonomic names and the recognition of locations fall into this class. (2) Finding words that represent co-references also falls into this class because it extracts very specific parts from a document, namely pronouns, on purpose neglecting here the process of figuring out what these pronouns actually refer to because this is a disambiguation task, see below. (3) Among all the paragraphs in a document, identifying the ones that are footnotes, captions, or bibliographic references, respectively.

In principle, extraction tasks are low level classification tasks with a ‘not’ category: (1) When using HMMs or SVMs to extract proper names in the first stage of NER, the models usually classifying every token of the document text into ones that start a proper name, ones that continue one, and ones that do not belong to a proper name;

---

<sup>18</sup> Solutions to these tasks are usually composed from a combination of lower level NLP algorithms, e.g. NER.

several NER implementations also determine the category of NE in this stage, namely by distinguishing tokens that start a person name, ones that start a location name, etc. (2) Extracting paragraphs that are footnotes, captions, or page headers corresponds to classifying the paragraphs into the respective categories, plus a ‘not’ category for paragraphs that are neither.

Despite this technical analogy, it is helpful to uphold the distinction between extraction tasks and classification task because both the facilities that optimally support users in correcting their respective results and the optimizations that minimize correction effort are quite different, as will be shown below.

**Disambiguation Tasks.** Tasks in this class deal with figuring out the appropriate interpretations for pieces of a given document text that are ambiguous, with the possible interpretations depending on the actual piece of text; examples of this class of NLP task are, among others: (1) Figuring out which NE a given pronoun refers to, the second stage of co-reference resolution. (2) Determining the appropriate coordinates for a given location name in cases where there are multiple locations with the same name. (3) Figuring out the proper LSID for a taxonomic name that is a homonym, i.e., figuring out the actual meaning of a name that has been assigned to more than one taxon<sup>19</sup>. (4) The latter disambiguation may just as well be necessary for the names of persons, for instance to determine whether the name “Paul McCartney” refers to that famous musician in Example 2.3.2.

Disambiguation tasks differ from classification tasks in the interpretations available for the individual markup elements: Classification tasks use a fixed and usually small set of categories out of which one gets assigned to every markup element to classify; in disambiguation tasks, on the other hand, the available interpretations depend on and differ between the individual markup elements to disambiguate, e.g. the pairs of geo-coordinates available for a given location name, or the named entities a personal pronoun can refer to.

**Parsing Tasks.** The tasks in this class are ones that generate an overlay of non-overlapping detail markup elements for a piece of document text, usually for a rather small piece; among others, examples of this class are: (1) Semantic parsing, i.e., identifying subject, predicate, object, and any possible adverbials in a sentence. (2) Extracting the details from a bibliographic reference, e.g. the author names, the year of publication, and the publication title. (3) Finding and marking the individual taxonomic epithets in a taxonomic name.

Like extraction tasks, parsing tasks are in principle low level classification tasks, namely ones that classify the individual tokens of the text to parse into categories that correspond to the parts of the overlay to generate. Likewise, parsing tasks can be perceived as extraction tasks that identify the individual parts of the to-generate overlay in the text to parse.

However, the distinction between parsing tasks and classification task is worthwhile to uphold, as is the one between parsing tasks and extraction tasks, because

---

<sup>19</sup> In actuality, the disambiguation of a given taxonomic name is somewhat more complex than only resolving homonyms, as the actual taxon the name refers to can also depend on the date when a document was published. However, an in-depth explanation of this issue exceeds the scope of the current discussion, as resolving taxonomic homonymies only serves as one of many examples of disambiguation tasks encountered in semantic markup generation.

both the optimal correction facilities for their respective results and the optimizations that minimize correction effort are quite different in all three classes, as we will see.

### 8.3.2 Improvements during the Madagascar Corpus Project

Over the course of the Madagascar Corpus Project, we have developed and deployed many improvements of the NLP tools that generate semantic markup in the automated phases of the individual steps. This section describes these improvements; their generalization follows.

**Taxonomic Name Markup.** The List View (see Section 7.4.2) is very helpful in correcting the results of extraction tasks; namely, it helps users with sorting out false positives without searching through the whole document. However, the List View does not help with false negatives, which the user has to search the entire document text for. We have therefore optimized the taxonomic name recognition tool towards maximized recall to avoid any false negatives.

To correct the result of taxonomic name normalization, we created a dedicated dialog that allows for selecting the full form of abbreviated epithets from respective lists. Users can also change the taxonomic epithets, and an integrated parser then automatically adjusts the other epithets to comply with the correction. Further improvements include displaying the number of the document page where a given taxonomic name occurs in the document, as well as displaying the context of the name in the document, about 10 words to the left and right. Both of these measures simplify deciding which token of the taxonomic name represents which epithet.

**Location Markup.** Just as with taxonomic name markup, the List View also helps with sorting out false positives in location markup. For the same reasons as above, we have tuned the location recognized towards maximized recall to save the users the effort of searching the document text for false negatives. A further improvement was again to display both page number and context of location names to give users a reliable basis for deciding whether or not a listed location name is a false positive.

**Treatment Markup.** It has turned out very efficient to generate the markup for treatments and other sections as a grouping of paragraphs, exploiting that section boundaries usually coincide with paragraph boundaries. The respective markup generation tool uses rules to classify the individual paragraph regarding whether they continue a section from the previous paragraph or start a new treatment or a section of a different type. This can also be perceived as an extraction task, namely as extracting the paragraphs that start new treatments or other sections. A specialized document view helps with the corrections; the user can visually check if the paragraphs are correctly grouped into treatments and other sections and can correct errors. Over the course of the project, we tuned the rules that recognize section starts towards maximized recall as well because in the correction dialog section starts are highlighted, and paragraphs the classifier has mistaken for section starts are easier to spot for users than section starts the classifier has missed.

**Structure of Treatments.** The tool that generates markup for the inner structure of the treatments uses the same paragraph grouping approach as the treatment markup; and the same document view helps users with the corrections. It has turned out beneficial to visualize the paragraphs of the different categories (see Table 4.1) in different colors to make them easier to review.

### 8.3.3 Generalization

The optimizations described in the previous section that are in themselves somewhat specific to the Madagascar Corpus Project, but conceptually they represent instances of optimizations that are applicable the individual steps of semantic markup generation processes in general.

**Recall First.** In the correction phase of every step that performs an extraction task, false positives sort out relatively easily from a respective list view, which causes a lot less effort for the user than searching the document for false negatives, even if the overall number of errors is higher. Therefore, a good approach to Goal 3 is to optimize the NLP tools that run in the automated phase of such a step towards maximized recall, even at the cost of (some) precision, as opposed to optimizing f-score or accuracy, which are relevant criteria in NLP research.

**Structure By Grouping.** Markup elements that represent the structure of a document, like chapters, sections, or subsections, are typically groups of paragraphs. Given the paragraphs, a respective classifier can generate such markup elements easily. Respective document views are very helpful in correcting the groupings.

**Task-Specific Visualization.** Specialized document views are very helpful to reduce user effort in the correction phase of each semantic markup generation step, as they help making it as easy and intuitive as possible for users to spot errors. Which actual document view is most helpful depends on the nature of the NLP task performed in a given step. In particular, the following views have proven beneficial for the individual classes of NLP tasks:

- The results of **classification tasks** best display in a list view that provides a drop-down next to each listed markup element to allow users to change the category assigned to the individual elements. Coloring the list entries dependent on their category renders the list more intuitive to review. If the listed markup elements are extracted details (e.g. NERs) displaying some surrounding context helps users in assessing whether an element is correctly classified. Providing an additional ‘remove’ category facilitates sorting out false positives while correcting the classification of true positives, e.g. when correcting the result of the extraction and the classification stages NER at the same time.
- For correcting the results of **extraction tasks**, the best choice is a list view in which users can select elements for removal by means of a checkbox. Coloring the listed elements dependent on whether or not the checkbox is activated again make the list more intuitive. Likewise helpful is displaying some context around the list entries to give users a broader basis for their decisions. Coalescing markup elements with the same textual content (e.g. multiple occurrences of a name) into a single list entry reduces the size of a list and makes it easier to review. This is especially helpful when correcting the output of recall-optimized NLP tools, which can contain a considerable number of false positives.
- The results of a **disambiguation task** display well in a list view that provides a drop-down with the possible interpretations next to each listed markup element. Such views differ from the ones used for correcting the results of classification tasks in that the interpretations in the individual drop-downs may be different for each individual list entry. Coloring the list entries dependent on their selected interpretations is therefore not possible. However, providing some context is helpful in this type of list view as well.

- The best document view for correcting the results of **parsing tasks** is a display of the parsed piece of text (e.g. a bibliographic reference) that displays the markup elements of parse result by means of semantic coloring, exploiting that parse results usually consist of non-overlapping elements. Users can then edit the parse by selecting text and using a context menu.

## 8.4 Process Level Optimization

Finding an optimal order of execution for the individual steps in a semantic markup generation process is all but trivial; optimal in the sense of achieving Goal 5, i.e., minimizing the overall effort for manual correction. Optimizing the execution order of the individual steps of a semantic markup generation process requires analyzing the dependencies between the steps, more specifically between the NLP tools that run in their automated phase. This section first identifies classes of dependencies and illustrates them with examples, both generic ones and ones taken from the Madagascar Corpus Project. Afterwards, it describes an optimization technique based on the dependencies and describes the TaxonX Process, which arranges the semantic markup generation step from Section 8.2 in an optimized order. The section closes with a generalization of the findings from the field study.

### 8.4.1 Dependencies between Semantic Markup Generation Steps

There are four kinds of dependencies between individual steps to consider when optimizing their order towards minimum overall correction effort; two of them are opposites of one another, or opposing flavors of the same sort of dependency.

**Definition 8.1.** An **Input Dependency** between two given steps S1 and S2 exists if the output of S1 is required as the input for S2.

Input dependencies between the semantic markup generation steps in the Madagascar Corpus Project mostly exist within logical steps, but also between them, for instance: (1) Finding page headers, page numbers, and footnotes based on their position in pages requires the document to have the pages marked. (2) Marking treatments and other sections by grouping paragraphs requires the paragraphs to be marked; the same applies to the internal structure of the treatments (3) Generating markup that represents the inner structure of treatments requires the treatments themselves to be marked first. (4) Taxonomic names need to be marked before they can be normalized or disambiguated by means of their LSIDs. (5) Locations need to be marked before being geo-referenced.

There are also many generic examples of input dependencies that are not from this project: (6) An NE tagger that requires POS tags to be present in a document has an input dependency on a POS tagger such that the latter has to run first. (7) A shallow parser that works on sentences depends on a sentence tagger to first generate markup that identifies the individual sentences in a document. (8) A co-reference resolver depends on NEs to be tagged in order to know what to resolve co-references to; it thus has an input dependency on an NE tagger.



**Definition 8.2.** A **Direct Dependency** between two given steps S1 and S2 exists if S1 has to be executed before S2 in order to prevent errors.

There are many examples of direct dependencies between the semantic markup generation steps in the Madagascar Corpus Project: (1) Paragraphs need to have their boundaries corrected before respective algorithms can classify them into main text, page headers, captions, footnotes, etc. (2) For identifying page numbers, page headers, and footnotes, it is essential for the document to have the pages marked. (3) Page markup has to be removed before treatments are marked because otherwise they two types of markup elements might be interleaved and thus hamper the wellformedness of the document. (4) For generating markup for sections, including treatments, by grouping paragraphs, it is essential for the paragraph boundaries to be corrected beforehand because otherwise the boundaries of the sections are erroneous, which incurs additional correction effort for the users. (5) Any kind of NER, recognition of taxonomic name and locations in this project, requires flowing text because especially hyphenated words badly impact result quality and thus increase correction effort; consequently NER can run efficiently only after structural normalization.

**Definition 8.3.** A **Positive Evidential Dependency** between two given steps S1 and S2 exists if the output of S1 can serve as evidence in the automation of S2, possibly resulting in fewer errors in the latter.

There are several semantic markup generation steps in the Madagascar Corpus Project whose output provides valuable evidence for other steps: (1) Page numbers are very helpful in identifying page headers because the former are usually part of the latter, but are easier to identify. (2) Taxonomic names and respective labels are good hints for where treatments start, and thus they are helpful in generating the markup delimiting the latter. (3) Both taxonomic names and locations give clues in classifying the paragraphs inside a treatment to generate markup for its inner structure.

A generic example of a positive evidential dependency is an NE tagger that can use POS tags as evidence, but also works without them, even though to a worse result; such an NE tagger has a positive evidential dependency on the POS tagger, so the latter should run before the former.

**Definition 8.4.** A **Negative Evidential Dependency** between two given steps S1 and S2 exists if S1 destroys evidence that can be useful for the automation of S2, possibly resulting in more errors in the latter.

The negative evidential dependencies between the semantic markup generation steps in the Madagascar Corpus Project are mostly steps in document cleanup and normalization: (1) Because structural normalization removes the page markup, it has to run only after the detection tools for page numbers, page headers, and footnotes, which work on document pages. (2) The font in footnotes is often smaller than in the main text and therefore footnotes contain more characters per line; because paragraph normalization destroys this valuable evidence, it should run only after the footnote detection tool.

#### 8.4.2 Dependency Based Optimization of Step Order

Dependencies between semantic markup generation steps, or more specifically the NLP tools used in their automated phases, provide a reliable starting point for finding an optimal order for the steps. In particular, the steps and their dependencies together form the **Step Dependency Graph**, a weighted directed graph with the steps being the vortices, the dependencies being directed edges, and the weight of each edge defined by the class of the dependency it represents; an edge from step S1 to step S2 expresses that S1 should be executed before S2.

**Dependency Induced Step Order.** If the step dependency graph is free from cycles and thus provides an order of the steps that respects all dependencies, it is obvious how to arrange them; if the order is a partial order, the steps that are equal under that order can be arranged in the process in any order. It can also happen that the graph degrades into a forest if there are groups of steps that are mutually independent and do not depend on any other steps; in such a case, these groups can be optimized internally and then included in the process in any order.

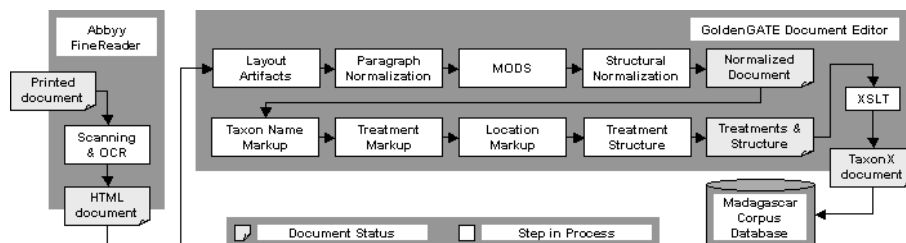
**Breaking Dependency Cycles.** The interesting case is when the step dependency graph does contain cycles and therefore there is no order for the steps that respects all their dependencies. The classes of dependencies then provide guidance for choosing the ones to ignore, also illustrating the fundamental differences between the individual classes: The two versions of the example with the NE tagger and the POS tagger imply that the distinction between input dependencies and positive evidential dependencies is gradual and can be implementation specific. However, these two types of dependencies are very distinct in their implications: An input dependency has to be respected in the order of the steps, whereas a positive evidential dependency is only a recommendation that can be ignored if doing so helps breaking a cycle in the step dependency graph. Likewise, the difference between direct dependencies and positive evidential dependencies lies in the severity of the implications when ignoring them: the additional correction effort that results from ignoring a direct dependency can be so high that ignoring the dependency does not make sense. Finally, when multiple positive evidential dependencies are candidates for ignoring in order to break a cycle, the task is to figure out the one whose exploitation yields the least reduction in correction effort and therefore is the least expensive to not respect.

**Ordering Independent Steps.** As mentioned above, there can be steps that are equal under the ordering relation induced by the step dependency graph. In such cases, a further aspect to consider in ordering these steps is whether the correction phases of multiple steps can be bundled, having the user inspect and correct the generated markup only after the last step of the bundle and thus reduce correction effort.

#### 8.4.3 The TaxonX Process

This section presents the TaxonX Process, the semantic markup generation process that we developed and optimized in the course of the Madagascar Corpus Project; its name is a reference to the TaxonX XML Schema [Catapano2006] in which its final output is marked. Note that the order of steps listed here represents the final result of the optimizations, not some intermediate state or even the starting point.

Figure 8.1 gives an overview of the Madagascar Corpus Process, consisting of the scanning and OCR part that was done in FineReader, the TaxonX Process responsible for generating the semantic markup, and the import of the completely marked docu-



**Figure 8.1. The Madagascar Corpus Process**

ments into the semantic corpus. The TaxonX Process is the part done in the GoldenGATE Editor, leading from the HTML document that comes out of OCR to the TaxonX document that finally goes into the semantic corpus. In detail, the TaxonX Process consists of the following steps:

1. Mark pages.
2. Identify page numbers, as an increasing sequence of numbers existing close to page boundaries.
3. Identify page headers at the top and bottom of pages, using page numbers as evidence, among others.
4. Identify footnotes at the bottom of pages, but above page headers.
5. Identify captions
6. Only now, have the user correct page internal paragraph boundaries and also the markup generated, using the Slide View (see Section 7.4.1) to go page by page. The rationale behind this aggregated correction is that both page headers and footnotes are paragraphs of their own, and that their boundaries are close to never erroneous because they usually have a distinctive distance to the main text in printed documents so OCR software can reliably recognize the boundaries of these paragraphs. The erroneous paragraph boundaries the user has to correct lie almost always within the main text.
7. Remove layout artifacts.
8. Normalize the document structure and remove page markup.
9. Normalize paragraph structure and de-hyphenate words.
10. Mark bibliographic references.
11. Import MODS document metadata from a respective web service.
12. Mark taxonomic names.
13. Normalize taxonomic names.
14. Get LSIDs for taxonomic names from a respective web service.
15. Mark treatments and other sections.
16. Mark locations within treatments. The rationale behind ignoring locations in the other sections is that the important information in this project are the associations of a taxon and a location, which are only given inside treatments. Therefore, all other locations can be ignored right away here, which saves the effort for correcting them; especially bibliographic references are very beneficial to ignore because they contain a many proper names and phrases in title case, which can severely irritate NER.
17. Geo-reference locations, getting the coordinates from a respective web service.

18. Mark internal structure subsections of treatments, i.e., their subsections. This works by classifying individual paragraphs into the categories from Table 4.1, based on rules that exploit the following observations, among others: (1) the ‘nomenclature’ subsection usually is the first one in the treatment and usually starts with a taxonomic name. (2) A series of taxonomic names identifies a ‘discussion’ or ‘diagnosis’ subsection, the former being much more frequent than the latter. (3) The presence of location names indicates a ‘distribution’ or ‘materials examined’ subsection, the latter being more frequent than the former. (4) The presence of the names of body parts (morphological features of specimens) indicates a ‘description’, ‘diagnosis’, or ‘discussion’ subsection.

#### 8.4.4 Generalization

The previous sections describe several optimizations that are in themselves specific to the Madagascar Corpus Project, but conceptually represent instances of optimizations that are applicable to process level optimization in semantic markup generation projects in general.

**Paragraphs First.** It is generally beneficial to restore the logical paragraphs as the very first thing in a semantic markup generation process, dealing with layout artifacts and normalizing paragraphs structure along the way. This is because only after normalization, NLP algorithms that generate semantic markup have the document text available in a condition that helps them achieve their best result quality.

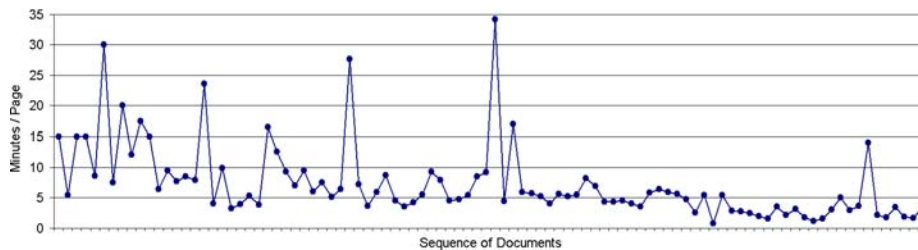
**Details Before Structure.** Whenever generating semantic markup that represents both the logical document structure and semantically important details, it is beneficial to generate the markup for the details first because it can serve as evidence for the steps that mark the structure.

**Structure Zoom-In.** If the logical document structure is deep, e.g. with chapters, sections, and subsections, and the lower level structure is somewhat uniform within each higher level structure element, it is beneficial to mark the structure in a top-down fashion because this allows exploiting the uniformity.

**Schema Last.** It is beneficial not to use schema-bound markup in the course of a semantic markup generation process, but to generate generic markup and to transform the latter into markup compliant with the output schema only as the very last step of the process, for several reasons: (1) Especially layout artifact detection greatly benefits from markup elements that are temporary and not part of the final output, namely the elements marking pages and page headers, etc. Using elements that do not exist in the final output schema would result in invalid intermediate states if the document were bound to a specific schema in that stage of a markup process. (2) Integrating existing NLP components for automated markup generation becomes harder when they have to produce markup that must conform to a schema right away. (3) To keep extracted information separate from the document text so it does not interfere with NLP algorithms that have yet to run, it is helpful to store this information in attributes of the respective markup elements. The output schema might require a different representation, however, e.g. in specific data elements, so it is beneficial to not be forced to use the final output representation immediately after the information is extracted.

## 8.5 Discussion

Over the course of the Madagascar Corpus Project, one of the participating biologists has generated semantic markup for about 100 documents; Figure 8.2 graphs the development of the average working time per page. At the start of the project, the markup generation tools were little sophisticated and not optimized for interactive use, and the order of the steps in the markup process was far from optimal. Semantic markup generation took over 12 minutes per document page at that time. As the project progressed, we implemented and deployed the improvements described in Section 8.3 and Section 8.4. As a result, the average working time per document page gradually decreased to about 3 minutes, a reduction by about 80%. The outliers are result from documents with exceptionally large pages or with very poor OCR quality. The aforementioned biologist accounts about 3 to 4 minutes of the 10 minute speedup per document page to training effects; the remaining 6 to 7 minutes result from the improvements. Unfortunately, it was not possible to validate these numbers more rigidly without actually disturbing the users in their work because the Madagascar Corpus Project proceeded under regular working conditions, with real output, not as a laboratory experiment. The user was familiar with the tools after about 10 to 15 documents, according to her own assessment, i.e., after a total of about 100 pages; the working time per document page was still around an average of about 9 to 10 minutes at that point. This furnishes additional evidence that the remaining speedup is largely due to the improvements implemented over time.



**Figure 8.2: Average working time per page**

To assess the benefit of specialized document views for correcting the results of automated semantic markup generation, we asked the users doing the markup to process a small document without the views for a test. Table 8.1 lists the amount of time users spent in the correction phases of the different semantic markup generation steps with and without the support of the views. The steps that benefit the most from automation and specialized document views are normalization of document, paragraphs, and taxonomic names, each with a speedup of over 90%. Three steps still cause considerable correction effort despite all improvements in user assistance: (1) Correcting the document structure takes considerable time because OCR errors do not follow any obvious regularity that could be exploited in a respective automated correction tool, and users consequently have to clean up every page. (2) Getting the LSIDs for the taxonomic names takes relatively long due to server side performance issues with the web-service lookups. (3) Correcting the inner structure of treatments takes rather long because the respective classification technique still yields many errors; despite this,

the effort with the respective document view (3:20) is still less than 20% of what it would be without (17:30).

Markup Step	Time (With Views)	Time (Without Views)	Speedup
<b>Scanning &amp; OCR</b>	<b>38:00</b>		-
Layout Artifact Detection	03:16	06:25	49%
Paragraph Normalization	00:10	02:39	94%
Get MODS Metadata	00:43		-
Taxon Name Markup	01:41	08:21	80%
Taxon Name Normalization	00:45	20:10	96%
LSID Referencing	03:53	14:40	74%
Treatment Markup	01:02	02:32	59%
Location Markup	01:40	02:11	24%
Treatment Structure Markup	03:20	17:30	81%
Structural Normalization	00:05	02:10	96%
<b>Markup Total</b>	<b>16:35</b>	<b>77:21</b>	<b>79%</b>
<b>Total</b>	<b>54:35</b>	<b>115:12</b>	<b>53%</b>

**Table 8.1: Time per step (in minutes) for 10 pages**

## 8.6 Optimizing NLP Components for Interactive Use

This section investigates in depth how NLP components can be designed specifically for use in interactive semantic markup generation, specifically how user interaction can help with improving NLP results and thus reduce the effort of manual corrections in the long haul. This comprises an analysis of the general potential, a discussion of NLP data models and machine learning techniques regarding how they can be used interactively, a design strategy for interactive NLP algorithms as a general approach to Goal 4, respective error metrics, and a description of the FAT taxonomic name recognition algorithm [Sautter2006] as an illustrating example.

### 8.6.1 Specific Requirements

The deployment of NLP tools in an explicitly interactive environment opens up whole new possibilities for learning and thus reducing errors and thereby the effort users have to spend on correcting NLP results. While the optimization for recall significantly reduces correction effort (see Section 8.3), it does not prevent the repetition of errors, which users then have to correct time and again; in the long haul, this is expectable to have a negative impact their motivation, according to Assumption U4.

Explicitly incorporating correction into the NLP procedure can do a lot more; namely, it facilitates feeding corrections straight back into the data structures the algorithm uses as the basis for its decisions. This means that the foundation of the decisions improves with every document processed, reducing the overall number of errors and preventing the repetition of ones that have been corrected before.

Furthermore, positives that users have confirmed as correct several times are unnecessary to display for correction time and again; omitting such confirmed positives in correction reduces the size of a list view that users have to go through when correcting the results of extraction tasks, for instance.

Conceptually, having NLP components learn from their own output after users have corrected it is an instance of online learning. However, online learning has been developed and used mostly for training perceptrons in time series prediction, e.g. to predict the development of stock market prices. Perceptrons have been deployed in NLP tasks only recently [Collins 2002], and they far less widely used than other models.

### 8.6.2 Suitability of Common NLP Data Models

Several different data models have been widely used in NLP, but few of them were designed for or tested with online learning; to the contrary, many of them are usually trained with machine learning algorithms that make a clear distinction between training phase and application phase. Thus, these data models are not necessarily suited to learn online from user corrections. Furthermore, interactive use requires an optimization for recall instead of accuracy in extraction tasks, as argued above, which has not been considered for any of the data models so far. This section therefore investigates data models regarding both online learning potential and the influence they allow on the types of errors they make.

Data models that consist of **Gazetteer Lists & Rules**, like the one presented in [Mikheev 1999], are well suited for online learning: (1) User corrections can be used to dynamically extend the gazetteer lists, overcoming the gazetteer compilation bottleneck identified by Cucchiarelli [1998]. While this is straightforward, unfortunately it is limited to learning concrete instances when classical gazetteer lists are in use, i.e., the model learns only the actual strings present in a document, with no generalization at all. Fuzzy list lookups can mitigate this problem to some degree, e.g. using a function that is agnostic of middle initials for lookups in a list of person names. (2) The learning methods presented in [Collins 1999] can be used to dynamically extend the rule system, achieving some generalization from the actual examples given in the document text. Even though these methods were originally designed for use in an unsupervised learning scenario, they are promising in this context as well. Finally, exclusion lists can filter out false positives that a user has corrected before.

Systems that use gazetteer lists and rules allow considerable influence on the types of errors they make: Lists used in a positive fashion, the usual case, are likely to yield high recall if populated eagerly, especially in combination with fuzzy lookups (see above). Extraction rules, in turn, can be designed to be very greedy, as well to boost recall, and rule learning algorithms can be optimized to generate such greedy rules.

As data structures, **Decision Trees** bear considerable similarity to the rule system presented in [Collins 1999], differing from the latter mostly in the way they are evaluated. This makes the rule learning methods from [Collins 1999] a rather promising

approach to improving decision trees based on NLP results after a user has corrected them. Because decision trees are almost exclusively used for classification tasks, an optimization regarding the types of errors they make is not necessary.

Both **Hidden Markov Models** [Rabiner 1986] and the more general **Conditional Random Fields** [Lafferty 2001] are statistical models, usually trained on labeled examples with supervised learning methods. NLP results that have been corrected by users are high quality labeled training data as well, so they can serve as supplementary training data. However, the statistical nature of both models makes it hard to ensure that the newly added training data has an immediate effect and that the corrected errors are sure to never occur again. In particular, adding new training examples causes the models to adjust the probabilities they consist of, but ensures in no way that the probability adjustments are sufficient to prevent errors from occurring again. If the new examples are given a very high weight to actually achieve the latter, on the other hand, the adjustments may incur new errors on other input.

Furthermore, neither model allows for adjustments regarding the types of errors in its output, rendering them ill suited for interactive use, at least if deployed standalone. A more promising approach is to deploy HMMs and CRFs as parts of larger NLP components that team them up with other data models in ways that exploit their strengths and contains their weaknesses.

**Support Vector Machines** [Cristianini 2000] are well suited for online learning because they are often trained through active learning, which differs from online learning mostly in that it does distinguish training and test data; thus, the incorporation of user corrections into an existing SVM is unproblematic. However, there are several issues to consider: (1) Research into active learning almost always assumes a training set of considerable size (1,000 to over 100,000 instances) to be available as a whole to choose the training examples from. This is not given in interactive NLP, not even in projects like the Madagascar Corpus Project; in particular, users will generate the markup for the documents one after another, quite likely starting out with rather small documents that provide only few examples. Pre-training the SVM before deployment can overcome this problem, but is only possible if a sufficiently large training corpus is available. (2) Active learning research usually works on corpora whose documents are homogenous and clean and follow a single style of speech. In contrast, corpora of legacy documents are rather heterogeneous in terms of layout, speech, punctuation styles, and possibly language, and they can contain a considerable number of character level OCR errors. (3) Even if an appropriate training corpus is available, an SVM will only yield acceptable performance on texts of the same genre; the same sort of documents (e.g. newswire) is insufficient. Even slight differences in genre result in a high rate of error [Chieu 2002], actually a rate that is unacceptable in a scenario with user correction. A difference in text sorts or even document language is likely to have an even stronger impact.

If a clear separation between all the instances of the two classes an SVM is trained to distinguish does not exist, there will always be some errors; the soft margin SVM [Cortes 1995] is an extension to the basic linear SVM that copes with such cases. The interesting part about soft margin SVMs is the penalty function used to control the optimization of the hyperplane: If chosen appropriately, namely to penalize false negatives way harder than false positives, a soft margin SVM can be optimized towards recall, just as required in interactive NLP.



SVMs have a further property that renders them well suited for use in interactive NLP; namely, the distance of an instance from the hyperplane directly corresponds to the certainty of its classification. This facilitates excluding highly certain positives from the correction list in extraction tasks, for instance, saving the correcting user the effort of looking through them.

**Neural Networks** [Hertz 1990] have great potential in online learning; actually, perceptrons, the usual subject of online learning, are a special case of them. However, neural networks require a considerable amount of training before achieving acceptable performance, incurring basically the same problems as discussed above SVMs.

Furthermore, reinforcement learning methods, which are often used with and well researched for neural networks, facilitate influencing the type of errors a neural network makes; namely, penalizing false negatives way harder than false positives tunes a neural network towards maximized recall as required.

### 8.6.3 Design Considerations

Whenever NLP components are specifically designed for deployment in interactive semantic markup generation, there are several things to consider, both with regard to decision making and to correction facilities.

**Decision Making.** (1) In components that perform extraction tasks, the decision process should be optimized for recall, as argued before; (2) The component does not need to present highly certainty instances to the user for correction, be it positives in an extraction task or assigned categories in a classification task; certainty can originate from prior user confirmations or corrections of a decision, or from a high certainty decision the NLP algorithm made itself. (3) Design can explicitly incorporate the option to delegate narrow decisions to the correcting users; namely, this means that the NLP algorithm at use need not reach a decision by all means, but can focus on identifying high certainty cases, leaving the uncertain ones to the user.

**Correction Facilities.** (1) To make correction facilities an integral part of an interactive NLP component facilitates providing users with the appropriate document views for the given correction task, as listed in Section 8.3.3. (2) Specifically designed correction facilities can considerably reduce the effort users spend on corrections because they can integrate logic that immediately draws conclusions from corrections and makes respective adjustments, which is not possible in generic document views like the List View. In parsing tasks, for instance, logic can adjust the parse to a correction just made so users do not have to make all the changes manually, an assistance feature that proved very beneficial in taxonomic name parsing. In classification or extraction tasks, in turn, logic can automatically make the changes implied by a correction, for instance by adjusting the category of NEs similar to the one the user just corrected. (3) Integrating correction facilities in NLP components offers the further advantage that the component can precisely determine when users are done with the corrections; at that point, the component can use its corrected output to update its data models, certain to not incorporate any errors.

### 8.6.4 Avoid Error, Reduce Uncertainty

The considerations on decision making in interactive NLP components discussed in the previous section boil down to the following underlying principle: **Avoid errors**

**from the start, reduce uncertainty over time.** Especially in extraction tasks, the following general strategy is promising:

**1: Candidate Generation.** Use means optimized for recall to extract any candidate match, i.e. any possible positive, from the document; in NER, for instance, this can be rules based on capitalization. Candidates may well be overlapping, as there are cases in which it is not immediately clear whether a given sequence of words is a single positive in its entirety, or whether it consists of multiple positives. For example, in the sentence ‘It is known that the Grand Duke of Finland is a person with a generous attitude’, ‘Grand Duke of Finland’ is a single NE, whereas in the rather similar sentence ‘It is obvious that Grand Duke of Miami is a person with a funny name’, ‘Grand Duke’ is the name of a person in which both first and last name are misleading, and ‘Miami’ is a separate NE, namely a location. To cope with such cases, candidate generation should mark all of ‘Grand Duke’, ‘Finland’ or ‘Miami’, and ‘Grand Duke of Finland’ or ‘Grand Duke of Miami’, respectively, as candidate NEs in these two sentences.

**2: Candidate Filtering.** Filter out candidates that have been confirmed to be negatives before. This stage has to be very conservative in order to make sure that no positive gets filtered out. It exists to make sure that candidates the user has flagged as negatives before do not show up in the correction facilities again.

**3: Sure Positives.** Apply very strict measures to identify sure positives among the candidates, similar to what Mikheev [1999] refers to as sure-fire rules, but even more restrictive, as errors in this stage would incur catastrophic consequences in the next.

**4: Iterative Inference.** Use the sure positives and the negative parts of the document, i.e., parts that are not candidates or positives, to dynamically generate rule based or statistical models. Then, use these models to decide on the remaining candidates, but do not make narrow decisions. Feed back the decisions into the model and apply the model to the remaining candidates again; repeat this process until there are no more new sufficiently certain decisions. Except for the reinforcement learning style extension of the model and the iteration, this is similar to the partial match stages of the technique presented in [Mikheev 1999].

**5: Correction.** Have the user decide on the remaining candidates; use the model from the previous stage to pre-populate the respective document view with the most likely decisions to likely reduce the number of changes the user has to make.

**6: Learning.** Extend the decision base for the sure positives based on the corrected result. It might also be possible to add new candidate filters to increase precision in that stage, but respective learning has to be very conservative in order to not hamper recall.

**Discussion.** By design, an interactive NLP algorithm implemented along the lines of this strategy will rarely make errors, if at all. If there is no pre-deployment training, users may have to make many corrections in the beginning, though, because the basis for the candidate filtering and the sure positive extraction is sparse then. Over time, however, learning is likely to decrease the number of candidates such an algorithm cannot decide on by itself with sufficient certainty, gradually reducing correction effort.

### 8.6.5 Metrics for Interactivity

To allow for a fair comparison between interactive NLP algorithms and non-interactive ones, the number of candidates left for the user to decide on has to be taken into account alongside precision and recall. The following definitions are very conservative, treating every candidate the user has to decide on as an error:

**Definition 8.1.** Positive Coverage, **P-Coverage** for short, is the fraction of positives decided on automatically:

$$\text{p - coverage} := \frac{|\{\text{true positives}\}| - |\{\text{positive user decisions}\}|}{|\{\text{true positives}\}|}$$

**Definition 8.2.** Negative Coverage, **N-Coverage** for short, is the fraction true positives among all possible positives, the latter comprising true positives and candidates the user had to decide on:

$$\text{n - coverage} := \frac{|\{\text{true positives}\}|}{|\{\text{true positives}\}| + |\{\text{negative user decisions}\}|}$$

**Definition 8.3.** Coverage is the product of p-coverage and n-coverage:  
 $\text{coverage} := \text{p - coverage} \cdot \text{n - coverage}$

**Definition 8.4.** Interactive Accuracy, **I-Accuracy** for short, is the product of accuracy and coverage:  
 $\text{i - accuracy} := \text{precision} \cdot \text{recall} \cdot \text{coverage}$

These metrics heavily penalize interactivity by treating every decision delegated to the user as if the NLP algorithm itself had decided the wrong way. An NLP algorithm may well achieve better results in these metrics by making decisions even if they are narrow and possibly erroneous, thereby preventing user interaction. However, learning from user corrected data has turned out to increase coverage rather quickly in the field study, while not incurring errors. Thus it appears rather realistic to expect interactive NLP algorithms to quickly close in on their non-interactive counterparts in terms of i-accuracy, achieving better values eventually. In addition, for non-interactive NLP algorithms, a decrease in i-accuracy indicates a decline in output quality, while in interactive NLP algorithms it merely indicates an increase in the effort a user has to spend on going through a list of markup elements for correction.

### 8.6.6 Excursion: the FAT Taxonomic Name Recognizer

FAT [Sautter 2006] is an extraction algorithm for taxonomic names that was designed explicitly for interactive use and was used successfully in the Madagascar Corpus Project. It may serve as a real-world proven example of the strategy presented in Section 8.6.4. The FAT algorithm is built upon the gazetteer list & rule data model. It implements the individual stages as follows:

(1) Candidate Generation: Regular expression patterns extract any sequences of words that might be taxonomic names. (2) Candidate Filtering: Based on gazetteers that contain words and sequences of words that are known to be negatives or that

users have decided to be negatives before, FAT filters the taxonomic name candidates. These lists comprise both author names, which can be nested in taxonomic names to clarify the authority of individual epithets, and words that are uncommon in general, but common in biosystematics literature. In addition, this stage uses regular expression patterns exploiting word ending that are common in natural language, but never occur in taxonomic epithets (3) Sure Positives: Exploiting both labels that are frequent in the nomenclature subsections of taxonomic treatments and labels that indicate the taxonomic rank of individual epithets, FAT finds sure positives among the candidates. In addition, this stage uses gazetteers containing taxonomic epithets that users have confirmed before. (4) Iterative Inference: Using the sure positives identified in the previous stage as well as the parts of the document text that are not part of any candidates, FAT finds further sure positives and rules out other candidates. (5) Correction: Any candidates that FAT could not decide on automatically are presented to the user so he can select the actual positives. (6) Learning: FAT integrates the epithets of the corrected result in its positive and negative gazetteer lists to prevent for the user to have to make any correction ever again. Exceptions to this are ambiguous word that can be taxonomic epithets, but also occur in common language; FAT always treats such words as candidates, but never as sure ones.

The FAT algorithm proved to generalize easily: Trained on ant literature, including respective lexicons, FAT achieved 99% of i-accuracy for literature on birds. This is remarkable because the gazetteer lists of known taxonomic epithets from the ant world are close to worthless in the bird world, so positive lists have to be learned from scratch. What remained valid were the negative lexicons and the structural rules. The ability to learn positive gazetteers from scratch may serve as proof for the general power of the interactive approach.

## 9 Controlling Complex Semantic Markup Processes

To efficiently perform semantic markup generation processes that are optimized based on the guidelines presented in Section 8.4, it is essential for users to strictly adhere to the optimized order of steps, and to correct any errors before proceeding to the next step. Otherwise, errors might propagate, and the NLP tools that run in the automated phase of some steps might lack their expected input, incurring preventable errors in their results. Providing users with a control mechanism that hints them to possible errors and enforces the order of the steps, i.e., achieving Goal 7, is therefore essential in order to render semantic markup generation as efficient as possible.

For controlling complex processes, Workflow Management Systems usually are the means of choice. The steps of a markup process correspond to activities of a workflow in such a system. In the context of semantic markup generation, however, it solely depends on the state of a document which step is next, i.e., on the markup created so far and on the errors in this markup, and not on the steps that have been executed before, at least not directly. This is because in contrast to workflows, there are no well-defined transitions between the steps in semantic markup generation processes, as free text editing can arbitrarily change the state of a document. For instance, users might simply undo the results of previous steps, which then have to be re-executed, a situation which would be impossible to model as a workflow in practice.

From the perspective of XML validation, a document has successfully passed through a semantic markup generation process if it passes a process-specific validation, e.g. if it is valid against a given XML Schema. Seen from this point of view, a step in a semantic markup generation process fixes a specific type of error with regard to the target schema, and thus, it is as promising to use XML schema languages to specify the desired outcome of the individual steps of such a process. XML Schemas are ill suited for a step by step validation because they validate a document strictly in a top-down fashion due to their grammar like nature, starting with the root element. This renders describing the intended outcomes of intermediate steps difficult to impossible. The rule base approach of SchemaTron is more promising because its point-check nature is well suited to spot specific errors; in fact we have found that SchemaTron rules are well suited to describe markup processes. However, a problem that XML Schema and SchemaTron have in common is their execution models: Validation tools report errors as they encounter them, not in the order a markup process intends to fix them, and thus existing tools are not well suited to control the step by step execution of a markup process.

As a solution to this problem, this chapter introduces ProcessTron, which in essence describes a semantic markup generation process by means of a slightly extended SchemaTron schema, but differs significantly from SchemaTron with regard to the execution model. In particular, the ProcessTron execution model applies the rules in sequential order, in line with the order of the markup steps, whereas SchemaTron applies them in the order they match markup elements in the document being processed. A thorough evaluation featuring both a laboratory experiment and a field

study confirms the effectiveness of the ProcessTron mechanism. The ProcessTron mechanism and its evaluation have been published in [Sautter 2010].

## 9.1 ProcessTron

This section explains how to control semantic markup generation processes by means of a rule based mechanism: It first shows how to represent such a process by means of slightly extended SchemaTron schemas, followed by the definition of the ProcessTron execution model, which controls semantic markup generation processes based on such schemas.

### 9.1.1 Describing Markup Processes with SchemaTron Schemas

A ProcessTron schema represents every step of a semantic markup generation process by means of a respective SchemaTron rule, with the following extensions: (1) Each rule bears an identifier for the automated markup generation tool to run in the automated phase of the step, and (2) immediate ID-based access to the possibly erroneous markup elements is indispensable. These requirements induce the following to elements as essential in a ProcessTron rule:

- **AMT-ID:** Each rule bears the identifier of the automated markup tool to run in the automated phase of the respective step to facilitate running the tool automatically.
- **Element-ID:** In each assertion and report, the textual message has to specify the ID of each markup element that fails the test or is recognized as erroneous. This facilitates highlighting suspected errors in the correction phase of the step represented by the rule. The optional **name** element of SchemaTron and its **path** attribute render this straightforward.

```
<rule context="paragraph" id="1">
  <automatedMarkupTool id="#paragraphBoundaryCorrector"/>
  <assert test="matches(text(), '.*[.|\!|\?|\']')">
    <name path="@id"/>: Paragraphs must end with a
      sentence-ending punctuation mark. </assert>
  <report test="matches(text(), '[a-z].+')">
    <name path="@id"/>: Paragraphs must not start
      with a lower case word.</report>
</rule>
```

#### Example 9.1: A ProcessTron rule

A SchemaTron schema that provides these extensions is referred to as a ProcessTron schema; Example 9.1 showcases a rule that represents the step which checks and corrects paragraphs boundaries, with parts specific to ProcessTron in bold for clarity. The `automatedMarkupTool` element specifies in its `id` attribute which markup tool to apply if a paragraph in a document does not comply with the rule, this being the case if the paragraph fails the test of an assertion or passes the one of a report. Both indicate that the paragraph's boundaries might be erroneous. The `path` attributes of the `name` elements in the assertions and reports are designed to include the IDs of the affected paragraphs in the error messages. The actual test is

exactly the same as in Example 2.5; the difference being that the ProcessTron specific parts (in bold) have been added.

The XPath tests require special attention to design; namely, to reliably point users to all potential errors in the correction phase of a step, the XPath tests have to be designed for 100% recall in order to reliably identify every possible error. A certain number of false positives are acceptable, just as in extraction tasks: Correcting users can quickly recognize that they are not actually erroneous and mark them as correct.

```

01 // functions for evaluating rules on a document
02 boolean fails(Test T, Document D) :=
03   true if D contains any markup elements that do not match
      (for assertions) or match (for reports) the XPath test of T,
04   false otherwise
05 boolean fails(Rule R, Document D)
06   for (Test T in R) // apply individual tests (assertions & reports) of R
07     if (fails(T, D) // D fails T, and thus  $\Phi$ 
08       return true
09   return false // D did not fail any test, thus does not fail  $\Phi$ 
10 // functions for performing individual steps in a markup process
11 void executeAutomatedPhase(Rule R, Document D) :=
12   apply the automated markup tool for the step represented by R
13 void executeCorrectionPhase(Rule R, Document D) :=
14   display D for manual correction, using R to highlight potential errors
15 void executeStep(Rule R, Document D) // execute the step represented by R
16   executeAutomatedPhase(R, D)
17   while (fails(R, D)) // stay in correction phase until D passes  $\Phi$ 
18     executeCorrectionPhase(R, D)
19 // main rule evaluation functions
20 Rule getCurrentStep(PT-Schema P, Document D)
21   for (Rule R in P) // treats schema as ordered sequence of rules
22     if (fails(R, D) // D fails R
23       return R
24   return nil // no failing rule found
25 // main function
26 void executeProcess(PT-Schema P, Document D)
27   while (true)
28     Rule R = getCurrentStep(P, D) // find current step
29     if (R == nil) // D did not fail any rule markup process complete for D
30       return
31     else executeStep(R, D) // execute step represented by  $\Phi$ 

```

**Figure 9.1: The ProcessTron execution model**

### 9.1.2 ProcessTron Execution Model

While SchemaTron schemas in themselves require only marginal extensions to describe semantic markup generation processes, the execution model is a different case: The XSLT-based execution model of SchemaTron is not well suited to control a markup process because its output does not reflect the order of the rules, but the document order of the markup elements the error messages refer to. The order of the rules reflects the order of the steps of the markup process, however, and thus it is essential

to enforce. The ProcessTron mechanism therefore requires an alternate execution model, which we dub the **ProcessTron Execution Model**, or PEM for short.

Figure 9.1 visualizes PEM as pseudo code, which applies the individual rules sequentially, one by one (Line 21); the rule order reflects the order of the steps in the semantic markup generation process the ProcessTron schema describes, and the loop goes through them in this order. As soon as a rule R fails (Line 22), i.e., it reports potentially erroneous markup elements, rule application stops. R identifies the first step of the markup process that is not yet complete, i.e., the next step to execute, referred to as S in the following. The PEM then executes S (Line 31): First, it applies the automated markup tool that belongs to S (Line 16), and then execution remains in the correction phase of S until the user has handled all potential errors reported by R (Lines 17 and 18). The user has two ways of dealing with errors: (1) He can correct the error. (2) He can approve the markup element in question, stating that it is not an actual error though it might look like one to the rule.

To reduce the effort users spend with corrections, ProcessTron can use the XPath tests of R to highlight all markup elements that might be erroneous. When S is complete, i.e., R reports no more errors, execution starts again by applying the first rule in the markup process definition (Line 28), which is necessary because a user might have introduced new errors in the correction phase. When no rule reports an error any more, the markup process is complete.

## 9.2 Evaluation

To assess the effectiveness of the ProcessTron mechanism, we have conducted both a laboratory experiment and a field study accompanying a real-world semantic markup generation project. The laboratory experiment showed with a statistical significance of over 90% that working with ProcessTron yields a speedup of over 50% in semantic markup generation for digitized legacy documents. The field study confirmed these findings under real-world conditions, deploying ProcessTron to control the TaxonX Process (see Section 8.4.3) in a project that generated semantic markup for all ant-related documents from the ZooTaxa<sup>20</sup> collection, in all 30 documents with a total of over 600 pages. In line with the results from the laboratory experiment, we observed the ProcessTron mechanism more than halves the time a user spends on generating semantic markup.

Further modeling the TaxonX Process in ProcessTron proved the latter well suited for its purpose, and it yielded valuable insights into markup process modeling in general. The focus of the evaluation was on the laboratory experiment and the ZooTaxa Project rather than on the process modeling itself, however. This is because in any semantic markup generation project there is only one markup process to model, which happens at a central instance. In sharp contrast to this, there can be many users who work with ProcessTron on many documents, and thus, reducing user effort is far more important and by far outweighs the effort for modeling the markup process.

---

<sup>20</sup> ZooTaxa (<http://www.mapress.com/zootaxa/>) is a biology journal.



### 9.2.1 Experimental Setup

This section describes the software and the metrics we used in both the laboratory experiment and the field study.

**Software.** As the platform for deploying ProcessTron, we have used the GoldenGATE Editor (see Section 7.4.2), integrating ProcessTron as a plug-in; besides the actual process-control mechanism, this plug-in also provides a specialized list view that displays potentially erroneous markup elements in the correction phase of each step to save them the effort of going through the whole document. The plug-in further has integrated editing facilities for ProcessTron schemas, namely an editor for individual rules, including test functions for the XPath expressions, a selector for the NLP tools to run in the automated phase of each step, and functionality for arranging the individual steps into a markup process.

**Metrics.** In both the laboratory experiment and the field study, we use the following metrics to quantify user effort: With  $d$  being the number of documents,  $t_i$  being the time it took to generate semantic markup for document  $i$  (subsequently referred to as the working time for the document), and  $p_i$  being the number of pages in document  $i$ , the metrics are the following ones:

**Definition 9.1.** The average working time per page (AWT) is based on the working times for the individual documents, regardless of document size:

$$\text{AWT} := \frac{1}{d} \sum_{i=1}^d \frac{t_i}{p_i}$$

The AWT treats all documents atomic units of equal weight, which is good for the comparison of documents with similar size. However, in the ZooTaxa Project the document sizes differ significantly, namely the smallest document has 6 pages, while the largest one has 119, and therefore this treatment gives overproportional weight to the smaller documents. To alleviate this, we use another metric that factors in the size of the documents:

**Definition 9.2.** The weighted average working time per page (WAWT) weights the times for each document relative to the document size, i.e., the overall average working time per document page:

$$\text{WAWT} := \frac{\sum_{i=1}^d t_i}{\sum_{i=1}^d p_i}$$

**Measurement.** In all experiments and studies, we have measured the time it took users to complete the markup of a document, starting with the OCR output. From these numbers, we have then computed AWT and WAWT. In the laboratory experiment, we consider the markup of a document to be complete if it matches that of a reference document. In all our markup efforts, the markup of a document is complete if the TaxonX Process is complete, i.e., the document is properly marked up and valid according to the TaxonX schema.

### 9.2.2 Laboratory Experiment

The purpose of this experiment was to assess the benefit of ProcessTron under controlled conditions. Except for deploying ProcessTron, we used exactly the same documents and the same setup as in the earlier experiment reported on in Section 7.5, to the benefit that we could use the results of that latter experiment as a baseline. From preliminary tests, we knew that we could expect a speedup of around 2, so according to [Cohen 1988], we would require 8 data points to achieve a statistical significance below 10% in a one-sided t-test with about 80% power. We gave the 8 participants in this experiment a brief training with the GoldenGATE Editor and ProcessTron, neither of which they had ever used before.

To enable using ProcessTron, we then defined and modeled a markup process for cooking recipes, logically consisting of the following steps:

1. **Layout-Artifact Detection**, same as in Section 8.2.1
2. **Paragraph Correction**, same as in Section 8.2.1
3. **Paragraph Normalization**, same as in Section 8.2.1
4. **Structural Normalization**, same as in Section 8.2.1
5. **Ingredient Markup**. Mark up the ingredients. This helps to identify recipe titles and ingredient listings in the subsequent steps.
6. **Recipe Markup**. Mark up individual recipes.
7. **Cooking Tool Markup**. Mark up cooking tools. This helps to distinguish between ‘ingredient list’ and ‘preparation’ in the next step.
8. **Structure of Recipes**. Mark up the subsections of the recipes, namely title, ingredient list and preparation, plus (if present) background information, advanced tips and recipe variations.

This markup process layout facilitated reusing the rules for almost all of the normalization steps from the TaxonX Process, namely Steps 1 through 4, which is about half of the process; the only part we had to adjust and partly model anew was the detail and structural markup following thereafter. For instance, the configuration of the NLP tool that was designed to generate markup for the inner structure of treatments required some adjustments to be able to generate markup for the inner structure of recipes, which involve other types of subsection, and consequently other categorization rules.

	<b>With ProcessTron</b>	<b>Baseline (from Section 7.5)</b>
<b>Average working time in minutes (minutes/page)</b>	29.50 (2.46)	79.1 (6.59)
<b>Standard Deviation</b>	14.22	18.30
<b>Speedup (over baseline)</b>	62.71%	N/A

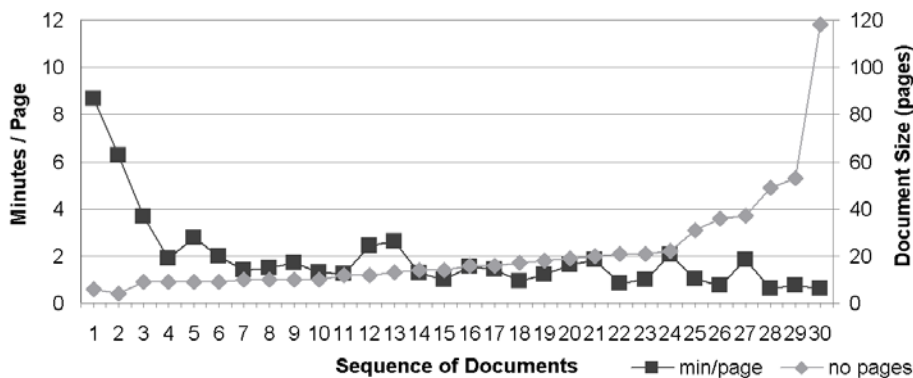
**Table 9.1: Results of laboratory experiment**

Table 9.1 shows the average time the participants spent on generating semantic markup for a document with and without the support of ProcessTron, with the baseline numbers taken from the earlier experiment reported on in Section 7.5. These

numbers clearly prove that ProcessTron yields a considerable speedup; in particular, the time it took the participants to generate the semantic markup for a document is less than half with ProcessTron than without. This result has a post-hoc statistical significance below 1% in the paired t-test, at over 90% power, far stronger than planned, emphasizing the usefulness of ProcessTron.

### 9.2.3 The ZooTaxa Corpus Project

In additions to the laboratory experiment, we have successfully deployed ProcessTron in the ZooTaxa Corpus Project, a real-world semantic markup generation project in the biosystematics scenario (see Section 4.1). Using the GoldenGATE Editor and ProcessTron, a biologist generated TaxonX markup for all ant-related documents from the ZooTaxa collection, 30 documents with a total of over 600. In a field study accompanying this project, we have measured the evolvement of the average working time per page, taking the figures from the Madagascar Corpus Project (see Section 8.5) as the baseline for the comparison. The sets of documents in the two projects are mutually disjoint. Apart from this, the only difference between the two projects is that ProcessTron was used in the ZooTaxa Corpus Project but not in the Madagascar Project. The biologist who participated in the ZooTaxa Corpus Project had participated in the Madagascar Project before, so she was proficient with the TaxonX Process and the GoldenGATE Editor before the start of the ZooTaxa Project, which rules out any learning effects in this respect and makes using or not using ProcessTron is the only variable.



**Figure 9.2: Document size in pages & working time per page in minutes**

Figure 9.2 shows how the working time per document page evolved throughout the ZooTaxa Corpus Project. The graph implies that it took the user only the first 3 documents to get used to working with ProcessTron, as the working time per page declined significantly with these initial documents; from the fourth document onward, the average working time per page leveled off around 2 minutes, keeping slightly decreasing towards around 1 minute per page over the remaining documents. The oscillations in the graph are result from the peculiarities of individual documents, namely from them requiring more or less manual corrections.

Table 9.2 lists the average and weighted average working time per page, with the number of documents  $d$  being 30. Due to the familiarization phase that lasted for the

initial 3 documents, Table 9.2 also lists both averages without these documents taken into account, labeled ‘**after familiarization**’, with *i* starting at 4 instead of 1 in the formulas defined in Section 9.2.1. The numbers clearly show the benefit of working with ProcessTron: The working time per page is slightly more than 1 minute, as opposed to the 3 minutes per page measured during the Madagascar Corpus Project; this represents a speedup of around 2.5, even higher than in the laboratory experiment.

<b>Measured</b>	<b>Working Time (minutes / page)</b>
Average working time per page over all documents	1:56
Average working time per page after familiarization	1:19
Weighted average working time per page over all documents	1:27
Weighted average working time per page after familiarization	1:11

**Table 9.2: Measurements from ZooTaxa Project**

### **9.3 Modeling the TaxonX Process**

A prerequisite for deploying ProcessTron in the ZooTaxa Corpus Project and elsewhere is to have the TaxonX Process modeled in a ProcessTron schema. Modeling the TaxonX Process did not pose any major difficulties, which furnishes proof that ProcessTron is sufficiently expressive; but there are several interesting issues to consider. The individual rules, and in particular the tests (report or assertion) they consist of, are not very complex: Most of the rules require only one test. Only few ones turned out easier to model with two or three tests, mostly the ones with tests that involve regular expression patterns; otherwise the latter would have become highly complex.

#### **9.3.1 Modeling Possibly Traceless Steps**

There are two kinds of steps that may appear somewhat hard to model at first, steps that (a) generate markup for semantic details that may or may not be present in a document, and steps that (b) work with temporary markup elements or generate markup for artifacts that will be deleted later on. Both kinds of steps pose the following challenge: Suppose a given Step *S* creates markup elements of a specific type, referred to as *M* in the following. An apparently straightforward approach to model *S* is an XPath test that checks whether or not markup elements of type *M* are present in a document: If markup of type *M* is present, *S* has been executed, otherwise not. This simple approach does not work in either of Case (a) and Case (b), as the next paragraphs will explain. Keeping a list of the steps that have been executed so far is not an option either, since users can simply undo entire steps by hand, as explained earlier, and thus ProcessTron has to be completely data driven.

(a) Checking the results of steps that generate the markup for important details is not trivial: If no respective markup elements (e.g., for locations) are present in the document, this can either mean that the step generating these markup elements has not been executed yet, or that the NLP tool that runs in the automated phase of the step did not create any markup elements, which can well be because no respective details are present in the document at all. In this case, we have two options: First, if the NLP tool performing a given step leaves specific traces besides the markup elements it creates, we can rely on these traces to check if the step has been executed, as Example 9.2 illustrates.

Page numbers are first marked, but later deleted from the document because they are layout artifacts. The NLP tool that extracts the page numbers, however, does not only generate respective markup elements, but also adds page number attributes to the page and paragraph elements. Thus, the page number attributes of the paragraphs are reliable evidence whether or not the page number extraction step has been executed.

#### **Example 9.2: Using side effects of steps**

Second, regular expression patterns are a reliable means to check whether or not a step that generates markup for distinctively structured parts of the text has been executed; namely, the respective XPath test can check if there are pieces in the document text that match a specific pattern, but are not marked accordingly. Example 9.3 illustrates this for geo-coordinates; and the same approach proved reliable for dates.

Geographic coordinates are highly valuable details, but they are not always given in older documents, and thus it is not sufficient to check for the presence of respective markup elements to find out whether or not the step responsible for marking them has been executed. The respective XPath test uses a regular expression pattern to check if the document text contains parts that might be geo-coordinates, but are not marked accordingly.

#### **Example 9.3: Using regular expression patterns**

(b) Temporary markup comprises elements that are created in a specific step S1 of a semantic markup generation process, but then are removed in a later step S2. The purpose of such markup elements solely is to help in the steps between their creation and their removal. To decide whether temporary markup is yet to create or has already been removed is a challenging task. As logging is not an option, respective rules have to rely either on markup elements created in steps related to S2, or on markup elements generated in steps in between S1 and S2, or on evidence from the document text; Example 9.4 illustrates this.

An early step of the TaxonX Process temporarily marks pages to help detecting footnotes and other print layout artifacts. After artifact detection is complete, a later step removes the page markup. Thus, checking for the presence of page markup elements alone is insufficient to tell if pages markup is yet to generate; the respective step additionally relies on the page boundaries, which mark the border between two pages in the OCR output: The first step marks the page borders and the actual pages between these borders, and both page borders and

pages are removed after structural normalization. Exploiting this dependency, the ProcessTron rules that represent the respective steps check if the document contains both page borders and pages: If the former are present, but the latter are not, page markup is yet to generate; if neither is present, the page markup has been removed.

#### **Example 9.4: Using multiple related types of markup elements**

### **9.3.2 Further Process Modeling Guidelines**

In the course of modeling the TaxonX Process for the ZooTaxa Corpus Projects, it turned out very helpful to study the NLP tools that run in the automated phases of the individual steps. In particular, studying the way they work, the evidence they rely on (see Example 9.5), and the output and the errors they might generate is extremely helpful when designing the XPath tests for the rules. Furthermore, it is just as helpful to test the rules for the individual steps in isolation on specific example documents, as this shows possible errors early on, similar to unit tests [Kolawa2007] in software engineering. Only after these individual tests it is sensible to compose the actual ProcessTron schema, i.e., to arrange the individual rules according to the order of the markup generation steps they represent.

The NLP tools that run in the automated phases of the individual steps of a semantic markup generation process may rely on different kinds of evidence. NER components, for instance, might use word structure (by means of regular expression patterns) or gazetteer lists. NLP tools that create structural markup may rely on statistical models, or on rules referring to detail markup. As these different techniques are susceptible to different errors in the document, it is sensible to use rules whose design reflects these differences: If the tool for the automated phase of a given step uses regular expressions, for instance, it is often sensible to as well use regular expressions in the ProcessTron rule that represents this step. The following two instances illustrate this:

1. Suppose an NLP tool that generates markup for dates, and that this tool uses regular expression patterns to recognize dates based on their distinctive syntactical structure. Then an XPath expression with a regular expression that tests if all text snippets with this particular structure are marked as dates is a suitable means to test whether or not the step that generates markup for dates has been executed.
2. Suppose an NLP tool that marks figure captions, and that this tool relies on caption paragraphs to start with ‘Figure X.’, where X is the figure number, just like in this work. Then an XPath expression that tests if all paragraphs starting with the word ‘Figure’ followed by a number and a dot are marked as captions is a suitable means to test whether or not this tool has run.

#### **Example 9.5: Formulating rules dependent on NLP tools**

Modeling a markup process is not as straightforward as it might seem at first glance: (1) Handling temporary markup requires to carefully study the interdependence of individual markup steps. (2) For modeling steps that generate markup for

semantic details of documents, it has turned out helpful to pay a high degree of attention to the way of the NLP tools work internally.

#### **9.4 Discussion**

In both a laboratory experiment and a field study that accompanied a real-world semantic markup generation project, ProcessTron has proven itself a good approach to Goal 7, i.e., to provide users with a mechanism that guides them through semantic markup generation processes and assists them in correcting errors. Namely, both deployments show that using ProcessTron more than halves the time it takes users to interactively generate semantic markup for a digitized legacy document. A further advantage is that ProcessTron is very lightweight, so it also complies with Goal 8, i.e., to be possible to deploy on a desktop computer.





## 10 Relieving Experts from Non-Expert Work

Generating semantic markup, i.e., performing a complete markup process like the TaxonX Process (see Section 8.4.3), often requires domain knowledge when working with scientific documents. Especially correcting the markup of domain specific details can be impossible for lays, like the taxonomic names in the Madagascar Corpus Project (see Section 8.1) and the ZooTaxa Corpus Project (see Section 9.2.3). However, there can also be steps in which correction does not require domain knowledge, e.g. the whole document normalization part that makes up the first half of the TaxonX Process. To make this sort of corrections is a nuisance for expert users because it takes considerable time, but can be repetitive and dreary; in addition, experts often have little time on their hands. So achieving Goal 9 is likely to yields a high benefit. Namely, the share of working time accounted for by document normalization was roughly half of the overall working time in both the Madagascar Corpus Project and the ZooTaxa Corpus Project, so relieving expert users from the respective corrections is likely to cut half their average working time per document page.

Hiring or contracting lay users to perform the corrections is one option. However, in recent years another approach has become more and more popular: To let a large crowd of volunteers work over the Internet, an approach that has become known as crowdsourcing. It has proven successful for tasks like image labeling [Lintott 2008, Von Ahn 2006], double-keying individual words for OCR correction [Von Ahn 2008], grading the relatedness of word pairs for ontology construction [Eckert 2010, Siorpaes 2007], or correcting below sentence level NLP results, e.g. word sense disambiguation [Snow 2008].

While theoretically providing a virtually unlimited workforce, however, crowdsourcing also poses a series of unique challenges. Namely, there is no guarantee that users make high quality corrections whose results comply with Requirement O1: users may make errors by accident, out of sloppiness, or due to inability. Further, if users contribute for some external reward rather than genuine interest, like in [Eckert 2010, Von Ahn 2008], raking in the reward might well be their primary motivation, and they might not even have any interest in actually correcting any errors; to get as high a reward as possible for as little as possible effort, they might not bother to spot and correct errors, but simply ignore them. Whatever the reason for a user to not correct errors properly, for crowdsourcing to have any expectable benefit, it is essential to establish dedicated data quality enforcement mechanisms that prevent erroneous corrections, i.e., to achieve Goal 10.

Distributed Proofreaders, so far the only project to crowdsource a task comparable to page structuring in terms of complexity, uses a very high level of redundancy to enforce data quality. However, redundancy has a severe impact on throughput: over 4.000 volunteers managed to process about 18.000 documents in 10 years, corresponding to an average of merely one third of a document per volunteer and year.

This level of redundancy is not suitable for correcting the structure of pages in the enormous document collections produced by recent digitization projects. Think of the vast number of documents that Biodiversity Heritage Library (BHL) has digitized and

OCR-processed so far, over 32.000.000 pages in total, as of January 2011. This amount of data is impossible to process in reasonable time with the high redundancy approach used in Distributed Proofreaders. It requires alternative approaches that achieve the required data quality with a lower level of redundancy and higher throughput. ReCAPTCHA has already achieved this for the correction of character level OCR errors, with a transcription accuracy of well over 99% with only threefold redundancy [Von Ahn 2008]. However, the size of the transcription task is very small (two words), and users are highly motivated, as they have to complete their task successfully to get what they want. Thus, it is unlikely that the data quality enforcement mechanisms used in reCAPTCHA are readily applicable to more complex tasks, like correcting the structure of pages.

To facilitate crowdsourcing large numbers of complex tasks, like correcting the page structure in large collections of digitized legacy document, this chapter develops generic data quality enforcement mechanisms that can handle complex tasks and mitigate the tradeoff between data quality and throughput. After introducing some basic notations, this chapter first formalizes the errors that can occur to facilitate mathematical analyses. Second, it revisits related work, adding formal analyses. Third is the presentation of a no-redundancy baseline case and a running example. Fourth, this chapter develops three mechanisms for enforcing data quality and increasing throughput, including mathematical proofs of their effectiveness; note that the explanations in this chapter assume that a crowdsourcing system knows its users and can distinguish them. Thorough mathematical analyses of expected result quality and throughput, and the impact on user motivation accompany the description of the three data quality enforcement mechanisms. A simulation-based evaluation of all three mechanisms and their combinations and an experience report from a real-world deployment attempt conclude the chapter. Two of the mechanisms and the analyses and evaluation of their impact on throughput and result quality have been published in [Sautter 2011].

## 10.1 Decisions, Tasks & Functions

**Definition 10.1:** A **Decision D** is an atomic parameter set by a user.

For instance, a decision is to specify if a given paragraph belongs to a document's main text or is a page header, a footnote, a caption, or an artifact originating from OCR.

**Options(D)** := {**O**<sub>1</sub>, ..., **O**<sub>o</sub>} denotes the **set of options available for D**. In addition, **N** ∉ **Options(D)** denotes the **null option**, which models the case that D is undecided so far.

For instance, the options for a decision can be the available classes for named entities or the paragraphs types. The null option then indicates that a paragraph or named entity has not yet been assigned a type or class, respectively. Options(D) can be large, e.g. when users have to type the transcriptions of word images into a text field, like in [10, 11]. Options(D) then contains all strings of a certain length, and practically all valid words of the language; N is the empty string in this case.

At every point of its time of residence in the crowdsourcing system, a decision  $D$  has an option  $S(D) \in \text{Options}(D) \cup \{N\}$  assigned to it.  $S(D)$  is the so-called **state** of  $D$ ; there are several dedicated states to distinguish:

$S_O(\mathbf{D}) \in \text{Options}(D) \cup \{N\}$  is the **original state of  $\mathbf{D}$** , i.e., the state assigned to  $D$  when it enters the crowdsourcing system.

$S_I(\mathbf{D}, U) \in \text{Options}(D)$  denotes the **state a user  $U$  has assigned to  $\mathbf{D}$  in his input**, i.e., the option this user has selected; an input state cannot be  $N$ .

$S_R(\mathbf{D}) \in \text{Options}(D) \cup \{N\}$  is the **result of  $\mathbf{D}$** , i.e., the state of  $D$  when leaving the crowdsourcing system. A null result, i.e.,  $S_R(D) = N$ , indicates that the system could not determine a meaningful result for  $D$ .

$S_C(\mathbf{D}) \in \text{Options}(D)$  is the **correct state of  $\mathbf{D}$** , i.e. the outcome that respective experts would agree on, serving as the gold standard in mathematical analyses.

$\text{Input}(\mathbf{D}) = (S_I(D, U_1), \dots, S_I(D, U_u))$  is the **input list of  $\mathbf{D}$** , comprising the inputs that users  $U_1, \dots, U_u$  have contributed to  $D$ .

For instance, the initial state can be the class an NLP tool has assigned to a named entity.

$\text{Contribute}(\mathbf{D}) = (U_1, \dots, U_u)$  is the **contributor list** of  $D$ , i.e., the list of all users who have contributed an input to  $D$ , in the order they made their contributions.

$\text{Contribute}(\mathbf{D}, O) = (U \in \text{Contribute}(D) \mid S_I(D, U) = O)$ , with  $O \in \text{Options}(D)$ , denotes the list of the users whose input for  $D$  is equal to a given option  $O$ .

**Definition 10.2:** A **Task  $\mathbf{T} = (D_1, \dots, D_d)$**  is the unit of work assigned to users, consisting of one or more decisions  $D_1, \dots, D_d$ .  $\square$

The individual decisions that make up a task can be either **connected** or **independent**. In the former case, tasks are fix and atomic, and a crowdsourcing system cannot modify them by adding or removing decisions. In the latter case, the system can freely assemble decisions into tasks.

At any point of its time of residence in a crowdsourcing system, a task  $T$  has a **state**  $S(T)$ . The state of a task is the composition of the states of the individual decisions it consists of, namely  $S(T) = (S(D_1), \dots, S(D_d))$ . Analogously to individual decisions, there are several dedicated states to distinguish:

$S_O(\mathbf{T}) = (S_O(D_1), \dots, S_O(D_d))$  is the **original state of  $\mathbf{T}$** .

$S_I(\mathbf{T}) = (S_I(D_1, U), \dots, S_I(D_d, U))$  the **input user  $U$  has contributed to  $\mathbf{T}$** .

$S_R(\mathbf{T}) = (S_R(D_1), \dots, S_R(D_d))$  is the **result of  $\mathbf{T}$** , i.e., its state after all user interactions.

$S_C(\mathbf{T}) = (S_C(D_1), \dots, S_C(D_d))$  is the **correct state** of  $T$ , again defined as the outcome that respective experts would agree on, serving as the gold standard in mathematical analyses.

$\text{Input}(\mathbf{T}) = (S_I(T, U_1), \dots, S_I(T, U_u))$  is the **input list of  $\mathbf{T}$** , comprising the inputs that users  $U_1, \dots, U_u$  have contributed to  $T$ .

**Definition 10.3:** The distance **Distance**( $S_1(\mathbf{T}), S_2(\mathbf{T})$ ) of two states  $S_1(\mathbf{T}), S_2(\mathbf{T})$  is the number of decisions for which they differ, formally  $\text{Distance}(S_1(\mathbf{T}), S_2(\mathbf{T})) := |\{D \in T \mid S_1(D) \neq S_2(D)\}|$ .

The probability  $\mathbf{P}(S_X(\mathbf{T}) = S_Y(\mathbf{T}))$  of two arbitrary, but distinct states  $S_X(\mathbf{T})$  and  $S_Y(\mathbf{T})$  of a task  $T$  to be equal is the product of the probabilities of the respective states of the individual decisions  $D \in T$  to be equal, formally:

$$\mathbf{P}(S_X(\mathbf{T}) = S_Y(\mathbf{T})) = \prod_{D \in T} \mathbf{P}(S_X(D) = S_Y(D))$$

Assuming that  $\mathbf{P}(S_X(D) = S_Y(D))$  has the same value for all  $D \in T$  leads to a simpler formula that eases both presentation and computations:

$$\mathbf{P}(S_X(\mathbf{T}) = S_Y(\mathbf{T})) = \mathbf{P}(S_X(D) = S_Y(D))^{|T|}.$$

Note that this assumption does not incur any loss of generality because it does not make any further assumptions regarding the nature of the individual decisions a task  $T$  consists of.

**Definition 10.4:** An **abstract input aggregation function** **Result**( $\mathbf{D}$ ) is a function of type  $\text{Input}(\mathbf{D}) \rightarrow \{N, S_R(\mathbf{D})\}$  that computes the result of a decision  $D$  from  $\text{Input}(\mathbf{D})$ . Analogously, **Result**( $\mathbf{T}$ ) is the respective function of type  $\text{Input}(\mathbf{T}) \rightarrow \{\emptyset, S_R(\mathbf{T})\}$  for a task  $T = (D_1, \dots, D_d)$ . Unless specified otherwise,  $\text{Result}(\mathbf{T})$  is as follows:

$$\text{Result}(\mathbf{T}) := \begin{cases} \emptyset & \text{if } \exists D \in T : \text{Result}(\mathbf{D}) = N \\ (\text{Result}(D_1), \dots, \text{Result}(D_d)) & \text{otherwise} \end{cases}$$

A crowdsourcing system successively obtains inputs from users and adds them to  $\text{Input}(\mathbf{T})$ . It evaluates  $\text{Result}(\mathbf{T})$  after the addition of each input; once  $\text{Result}(\mathbf{T})$  does not return  $\emptyset$ ,  $T$  is complete, and no further input is required. Example 10.1 gives an impression of possible definitions of concrete input aggregation functions; other concrete input aggregation functions are the ones used for  $r$ -Redundancy (cf. Section 10.4.XYZ),  $v$ -Voting (cf. Section 10.5.3), and Vote Boosting (cf. Section 10.5.4).

A very simple example of a concrete input aggregation function is the following:  $\text{Result}_{E1}(\mathbf{D}) := N$  if  $|\text{Input}(\mathbf{D})| = 0$ ,  $S_1(D, U)$  otherwise; this function defines the result of  $D$  as the first input some user  $U$  contributes for  $D$ .

A more complex example is  $\text{Result}_{E2}(\mathbf{D}) := S_1(D, U)$  if  $|\text{Input}(\mathbf{D})| = 2$  and  $S_1(D, U_1) = S_1(D, U_2)$ ,  $N$  otherwise; this function defines the result of a decision as empty/undefined unless there are exactly two agreeing inputs from two users  $U_1$  and  $U_2$ .

#### Example 10.1: Instances of concrete input aggregation functions

**Definition 10.5:** The **abstract reward function** **Payoff**( $\mathbf{U}, \mathbf{T}$ ) is a function of type  $U \times T \rightarrow R$  that computes the payoff user  $U$  gets for contributing input to task  $T$ .

Payoff(U,T) facilitates modeling scenarios that involve a reward system; situations without any reward correspond to a reward function that always returns 0.

**Payoff(D)** and **Payoff(T)** denote the overall payoff a reward system offers for a decision D and task T, respectively.

Both Payoff(D) and Payoff(T) can be fixed values, or they can depend on D or T, respectively; they are measures for the cost of having a task T processed by a crowdsourcing system. Further, the overall payoff for a task T is the sum of the overall payoffs for the decisions T consists of, formally:  $\text{Payoff}(T) := \sum_{D \in T} \text{Payoff}(D)$ .

**PayoffExp(U,T)** denotes the expected payoff user U receives for contributing input to a task T.

**WorkExp(T)** denotes the expected value of |Input(T)| at the moment the input aggregation function returns a non-empty result.

In other words, WorkExp(T) is the expected number of inputs to obtain until  $S_R(T)$  emerges; it is a measure for throughput: the lower WorkExp(T), the higher the latter, and vice versa.

Further definitions and notations will be introduced throughout the technical sections of this chapter to formalize and analyze the presented data quality enforcement mechanisms

## 10.2 Types of Errors

This section revisits and formalizes the errors that can occur in the initial states of decisions and tasks, in the inputs that users contribute through the crowdsourcing system, and in the task results. Note that the goal of the following explanations and this chapter in general is not to enable crowdsourcing systems to distinguish between the reasons of errors. Especially with regard to user inputs, this is generally not possible because the observation of an error typically does not reveal anything about the motivation of the user who incurred it. However, errors occurring for different reasons differ in their statistical nature, i.e., follow different patterns of occurrence, and thus require specific countermeasures.

In general, there is an error in a decision D if  $S(D) \neq S_C(D)$ . The prevention of errors in the result of D, namely that  $S_R(D) \neq S_C(D)$ , is the interest of the data quality enforcement mechanisms presented in this chapter. Orthogonal to the reasons of errors discussed below, there are two types of errors:

**Miss Errors** are errors that remain uncorrected, i.e., errors that prevail from the original state of a decision to a respective input from a user U or the decision result; formally, a miss error exists if  $S_O(D) \neq S_C(D)$ , and  $S_I(D,U) \neq S_C(D)$  or  $S_R(D) \neq S_C(D)$ , respectively.

**Added Errors** are errors introduced by users, i.e., correct original states that are falsified in a respective input from a user U or the decision result; formally, such an error exists if  $S_O(D) = S_C(D)$ , and  $S_I(D,U) \neq S_C(D)$  or  $S_R(D) \neq S_C(D)$ , respectively.

### 10.2.1 Accidental Errors

Accidental errors are errors in the inputs of otherwise benevolent users incurred by mistake, be it out of sloppiness, lack of focus, or erroneous judgment. Presumably, accidental errors occur randomly; further, errors resulting from sloppiness likely tend to be miss errors, while the ones resulting from misjudgments can be of both types.

**P('accidental miss')** is the average probability across all users that some user accidentally misses an error in a decision  $D$  of a task  $T$ .

**P('accidental add')** is the average probability that some user accidentally adds an error in a decision  $D$  of a task  $T$ .

### 10.2.2 Cheating Errors

Cheating errors occur because users do not bother to contribute thoughtful input. If the original state of a task  $T$  is a valid input, i.e.,  $S_o(D) \in \text{Options}(D)$  for all  $D \in T$ , it is safe to assume that cheating users simply submit  $S_o(T)$  as their input because this is the least possible effort. If the original state of a task consists of null values, i.e.,  $S_o(D) = N$  for all  $D \in T$ , like the initially empty text fields in [10, 11], cheating is modeled as users randomly selecting an option from  $\text{Options}(D)$  as their input. In the former case, adding an error requires making a change to the original state of a task; thus, submitting the original state of a task as an input without changing anything cannot add any error, so cheating errors are generally miss errors in this case.

**P('cheat')** is the average probability that some user cheats on a task  $T$  and thereby contributes an input with miss errors for all errors in  $S_o(T)$ .

### 10.2.3 Destructive Errors

Destructive errors are ones that malicious users make on purpose, be it out of vandalism or with the goal to outright sabotage a crowdsourcing project. Such errors can be assumed to occur randomly because users do not have any influence on the tasks they get to work on and thus cannot coordinate their falsifications. Destructive errors can be both miss errors and add errors; because they exhibit the same properties as accidental errors, destructive errors do not require special consideration or any specific countermeasures – from the viewpoint of the crowdsourcing system, they have the same effect, even though they originate from a different attitude.

### 10.2.4 Combined Error Probability

To simplify subsequent computations, this section aggregates the individual error probabilities into universal ones.

**P('miss')** is the average probability of a miss error to occur in a single input, namely:

$$P(\text{'miss'}) = (1 - P(\text{'cheat'})) \cdot P(\text{'accidental miss'}) + P(\text{'cheat'})$$

**P('add')** is the average probability of an add error to occur in a single input, namely:

$$P(\text{'add'}) = (1 - P(\text{'cheat'})) \cdot P(\text{'accidental add'})$$

### 10.3 Parameters & Figures

This section lists the exogenous and endogenous parameters of crowdsourcing systems and describes the optimization goals.

The **exogenous parameters** are: (1) The nature of the tasks, i.e., the number of decisions they consist of, the number of options in the decisions, and whether the decisions are connected or not. (2) The accuracy of the original states of the tasks, or, in other words, the number of errors to correct in each task, which corresponds to the accuracy of the artificial intelligence generating the original states. (3) The probabilities of users to make accidental errors and to cheat on tasks.

The sole **endogenous parameter** is the input aggregation function in use and its parameterization. The payoff function does not have any direct impact on the accuracy of task results, and is therefore not an endogenous parameter; it is rather a means of influencing user motivation and therefore the exogenous parameter modeling the probabilities of errors.

The **numbers to optimize** are: (1) the expected accuracy of task results, which corresponds to the probability that the result of a task is correct, and (2) the expected number of inputs required to achieve this accuracy, i.e., the expected value of  $\text{WorkExp}(T)$ . The latter is particularly important when using third-party crowdsourcing platforms that require a fixed monetary reward per input, like the Amazon Mechanical Turk [AMT]; in such a setting, the value of  $\text{WorkExp}(T)$  is proportional to the expected cost.

### 10.4 Related Work, Revisited

This section revisits crowdsourcing projects already introduced in Sections 6.6 and 6.7, analyzing the mechanisms they have deployed to enforce data quality using the formal notions from Sections 10.1 and 10.2. Further, the mechanisms are grouped by technical criteria here rather than by the nature of the errors they aim at preventing.

#### 10.4.1 r-Redundancy

Many crowdsourcing projects [Eckert 2010, Lintott 2008, Snow 2008] use a rather simple redundancy-based input aggregation function, referred to as **r-Redundancy** in the following, with  $r$  being the parameter specifying the number of inputs required per task, often an odd number. In particular,  $r$ -Redundancy means that, once  $r$  users have contributed an input to a task  $T$ , the most frequently given input becomes the result of  $D$ , for each Decision  $D$  in  $T$ . In general,  $r$ -Redundancy is suboptimal with regard to throughput because  $\text{WorkExp}_r(T)$  is always equal to  $r$ , i.e., a task always takes  $r$  inputs to complete, even if the first  $(r+1)/2$  inputs agree in all decisions, and the last  $(r-1)/2$  inputs do not have any influence on the result.

$\text{Result}_r(D)$  is the input aggregation function for  $r$ -Redundancy, formally:

$$\text{Result}_r(D) := \begin{cases} O_0 \in \text{Options}(D) \text{ such that} & \text{if } |\text{Input}(D)| = r \\ \quad | \text{Contribute}(D, O_0) | \text{ is maximal} & \\ N & \text{otherwise} \end{cases}$$

Observe that even if  $r$  is odd, results can be ambiguous with  $r$ -Redundancy: if there are more than two options for a decision  $D$ , i.e.,  $|\text{Options}(D)| > 2$ , it can happen that more than one option occurs in a simple (non-absolute) majority of inputs, as Example 10.2 illustrates

Suppose that  $\text{Options}(D) = \{O_1, O_2, O_3\}$  and  $\text{Input}(D) = (O_1, O_2, O_2, O_3, O_1)$  in a 5-Redundancy scenario. Both  $O_1$  and  $O_2$  occur in 2 inputs. In principle, the input aggregation function can resolve such cases in three ways: (1) pick the option that occurred first ( $O_1$  in this case), (2) pick the option that has achieved the majority first ( $O_2$  in this case)<sup>21</sup>, or (3) pick one of the two at random. Note that in neither case the result has an absolute majority.

### Example 10.2: Ambiguous decisions in $r$ -Redundancy

**Deployments.** Eckert et al. [2010] use a 5-redundant approach to arrange terms into a concept hierarchy. Each task consists of 12 independent decisions. Each decision is to compare a pair of terms with regard to relatedness and relative generality, i.e., which term is more specific or more general than the other one. To detect cheating, each task includes two dedicated decisions  $P$  and  $Q$ . Namely,  $P$  and  $Q$  are term pairs for which users can easily determine relatedness and relative generality solely based on common sense. If users get them wrong, this serves as an indicator for them not working thoughtfully. With this mechanism, Eckert [2010] has achieved a degree of data quality comparable to that of a concept hierarchy domain experts have constructed from the same terms. However, embedding decisions with known results like  $P$  and  $Q$  in every task only works with independent decisions that a crowdsourcing system can freely bundle into tasks; it is not possible if tasks consist of connected decisions, like correcting the structure of document pages.

Snow [2008] has successfully used 10-Redundancy based crowdsourcing for detail level NLP tasks like word-sense disambiguation, all tasks consisting of 30 independent decisions bundled randomly. The system does not include any mechanisms to detect or prevent cheating. The reported result quality is similar to the figures of Eckert [2010], which suggests that cheating at least has not been pervasive. However, it is questionable if this still holds if users work on many tasks over an extended period of time; the more so as Eckert [2010] has detected cheating attempts in considerable numbers. Thus, it is very unlikely to constantly obtain high-quality results in large crowdsourcing efforts without a mechanism that discourages cheating.

Distributed Proofreaders [Newby 2003] is an ongoing crowdsourcing project related to the digitization of legacy documents, with the purpose to correct OCR errors by means of repeated review. Tasks consist of one very large decision, namely the transcript of an entire document page. In contrast to other projects, users work incrementally, i.e., each user gets to see the changes of the users who worked on the task before him. Data throughput has been relatively low so far. Tens of thousands of volunteers have proofread around 18,000 works in roughly eight years, as of July 2010. A more sophisticated process separating the pages into smaller chunks might be more promising, e.g., a process using reCAPTCHA (see below) on the word level.

**Summary / Analysis.**  $r$ -Redundancy has turned out a good means of ensuring data quality, but this comes at a severe restriction of throughput – the more so as  $r$ -Redun-

---

<sup>21</sup> This is the mode used in the simulations later on.



dancy tends to waste user time by obtaining inputs for tasks whose result is already clear. Furthermore, the possibility of ambiguous decisions, as explained in Example 10.2, can impact result accuracy because the crowdsourcing system ceases to obtain inputs for a task before its result is backed by a secure majority of users.

Because  $\text{WorkExp}_R(T)$  is fixed to  $r$  for every given task  $T$ , it is hard to define a payoff function  $\text{Payoff}_R(U, T)$  for  $r$ -Redundancy that makes  $\text{PayoffExp}_R(U, T)$  dependent on the accuracy of  $S_I(T, U)$ , which would motivate a given user  $U$  to strive error-free inputs to increase his expected payoff. Namely, if  $\text{Payoff}_R(U, T)$  is defined to distribute a fixed reward  $\text{Payoff}(T)$  among all users who contributed an input to a given task  $T$ , the accuracy of  $S_I(T, U)$  has no effect on  $\text{WorkExp}_R(T)$ , so users cannot easily increase their payoff by avoiding errors. Achieving the latter requires a rather complex definition of  $\text{Payoff}_R(U, T)$ , namely one that counts the errors in each user's input and factors this figure in when distributing  $\text{Payoff}(T)$ . Furthermore, if  $S_O(T)$  is rather accurate, users may be tempted to cheat and submit it as their input right away without further checking to rake in the reward for an input with a few errors at as little effort as possible. Complicated additional measures are required to discourage this latter behavior, like the mechanism used by Eckert [2010]. Unfortunately, Eckert's mechanism is not applicable in the general case, particularly if tasks consist of connected decisions, which often happens in document digitization.

#### 10.4.2 Agreement Games

Agreement Games *synchronously* obtain inputs from two random users, referred to as  $U$  and  $V$ . Each task  $T$  usually consists of a single decision  $D$ , and usually  $S_O(D) = N$ . If the two inputs agree, they count as correct, and both users get a reward.

**Result<sub>AG</sub>(D)** is the input aggregation function for Agreement Games, namely:

$$\text{Result}_{AG}(D) := \begin{cases} S_I(D, U) & \text{if } |\text{Input}(D)| = 2 \text{ and } S_I(D, U) = S_I(D, V) \\ N & \text{if } |\text{Input}(D)| = 2 \text{ and } S_I(D, U) \neq S_I(D, V) \\ \emptyset & \text{otherwise} \end{cases}$$

**Payoff<sub>AG</sub>(U, T)** is the payoff function for Agreement Games, with a fixed value  $p$ , namely:

$$\text{Payoff}_{AG}(U, T) := \begin{cases} p & \text{if } S_R(D) = N \forall D \in T \\ 0 & \text{otherwise} \end{cases}$$

The rationale is that random pairing prevents users from colluding, e.g., from agreeing on some fixed input a priori. The odds of two random inputs to agree is only 1 in  $|\text{Options}(D)|$ . Thus, contributing thoughtful input is the only way of increasing the chance of reaching agreement and thus increasing  $\text{PayoffExp}_{AG}(U, T)$  beyond  $1/|\text{Options}(D)|$ .

**Deployments.** Von Ahn [2006] has successfully used this approach for image labeling. OntoGame [Siorpaes 2007] has shown that it also works well for ontology construction and alignment, and for named entity disambiguation.

**Analysis.** Theoretically, the agreement approach works for tasks that consist of multiple decisions; however, a single mistake of either user invalidates both inputs.

Defining agreement in a per-decision fashion can alleviate this if the decisions are independent. The system can then collect incomplete decisions and bundle them into new tasks until some pair of users agrees on an input. However, this does not work with tasks that consist of connected decisions; it is unclear if and how the approach could work in this case.

#### 10.4.3 Centrality-based Approaches

Centrality-based approaches do not work in a task-by-task fashion, but on a task list as a whole; namely, they gather a specific number of inputs for each task and then use centrality measures to compute the overall ability of each user from his frequency of agreement with other users. The ability then serves as a weighting factor in combining the individual inputs into task results.

**Deployments.** The GalaxyZoo [Lintott 2008] project has had over a million galaxy images classified into six basic categories by over 10,000 volunteers in less than 200 days. Their system presents randomly selected images to its users. Hütter [2008] uses a comparable approach for community-driven ontology construction. In his system, users can actively rate the inputs of other users, and a centrality measure computes the score of each user.

**Analysis.** Centrality-based approaches require the whole set of tasks to be available from the start, a condition not met by efforts like the digitization of legacy literature, where new tasks are generated continuously as work proceeds. Furthermore, centrality-based approaches can compute task results only in the very end, when they can weight the inputs of individual users. This renders it impossible to stream data objects through processes that produce crowdsourcing tasks subsequently for each object, like the individual steps of the TaxonX Process.

#### 10.4.4 ReCAPTCHA

ReCAPTCHA [Von Ahn 2008] is a crowdsourcing project that double-keys images of document pages in a word-by-word fashion, building on the CAPTCHA mechanism [Von Ahn 2003]. The CAPTCHA users have to solve consist of two random word images. One of them is the crowdsourcing task  $T$ , consisting of a single decision  $D$  on the correct transcription of the given word image. The other one is the actual CAPTCHA, referred to as  $C$  in the following. A CAPTCHA is a word image the system already knows the correct transcription  $S_C(C)$  for. ReCAPTCHA takes an input for  $D$  into account only if the CAPTCHA is solved, i.e.,  $S_T(C) = S_C(C)$ . A task is complete as soon as there are 3 agreeing inputs. The system has earlier obtained  $S_C(C)$  through the same mechanism now used for  $D$ . This means that as soon as there is a result for  $D$ , it can serve in the same role as  $C$  later on. When tasks are presented to users, the original state for both  $C$  and  $D$  is empty. Requiring 3 agreeing inputs for  $D$  renders mistakes highly unlikely: In practice, reCAPTCHA achieves a word-error rate well below 1%. The presence of the CAPTCHA  $C$  that is indistinguishable from the actual task  $T$  ( $= D$ ) counters cheating well.

**Analysis.** Tasks in reCAPTCHA are very small, and users normally need only a few seconds to make their input. Tasks that take more time are impractical as CAPTCHAs because they would probably annoy many users. Thus, few web pages would integrate such a mechanism, so throughput would be too low. Furthermore,

insisting on fully agreeing inputs is impractical if tasks consist of multiple decisions, as explained below.

However, the idea of requiring a fixed number of *agreeing* inputs for decision D is promising for preventing a crowdsourcing system from gathering obsolete inputs, thus increasing throughput; it will be generalized and reused below.

## 10.5 High-Throughput Crowdsourcing

To facilitate crowdsourcing of large numbers of complex tasks like proofreading digitized documents or checking and correcting the structure of their pages, this section now introduces data quality enforcement mechanisms that only mildly restrain throughput and work with arbitrary tasks. The baseline for the mathematical analyses of expected result accuracy and throughput is a crowdsourcing system that obtains a single input per task, introduced right after a running example scenario; after that follow the actual data quality enforcement mechanisms and their analytical evaluation.

**Restrictions of Analytical Evaluation.** To keep the computations in analytical evaluations simple and easy to follow, all computations assume the worst case for errors; that means that if several inputs contain add errors on a decision D of a task T, these errors are assumed identical, leading to agreement. This actually is the case only for binary decisions (i.e.,  $|\text{Options}(D)| = 2$ ), in non-binary decisions like the classification task from the running example introduced below in Section 10.5.1, it is an assumption that increases the error probability. The reason to make this assumption is to keep  $|\text{Input}(T)|$  low and thus reduce the number of cases to consider. The simulations do not make this assumption; their results show that the average value of  $|\text{Input}(T)|$  barely increases for  $|\text{Options}(D)| > 2$ , in the range of a few percent, over a wide range of values for the other exogenous parameters. Thus, the throughput computed with the simplifying assumption does not differ from the actual figure by much.

### 10.5.1 Running Example

The following running example sets the scenario for the illustrations: Be there a task  $T = \{D_1, D_2, D_3, D_4\}$ , with  $D_d$  being to determine the type of the d-th paragraph in a page. Further be

$\text{Options}(D_i) = \{\text{'page header'}, \text{'main text'}, \text{'caption'}, \text{'footnote'}\}$ ,  
 $S_O(T) = (\text{'main text'}, \text{'main text'}, \text{'caption'}, \text{'footnote'})$ , and  
 $S_C(T) = (\text{'page header'}, \text{'main text'}, \text{'main text'}, \text{'main text'})$ .

This corresponds to only 25% accuracy in automated classification, a figure far below the actual accuracy of close to all modern machine classification algorithms. This extremely low accuracy value is chosen for presentation purposes, namely so  $\text{Dist}(S_I(T), S_C(T)) = 3$  and  $|T|$  still remains easy to overview.

The values assumed for the exogenous parameters in the mathematical analysis are conservative but realistic; for an individual decision D in a task T, be on average

$P(S_O(D)=S_C(D)) = 80\%$   
 $P(\text{'miss'}) = 10\%$ , and  
 $P(\text{'add'}) = 5\%$ .

This means that the original state of a given decision D is assumed to have an 80% chance of being correct, and that users cause miss errors and add errors with 10% and 5% probability when contributing an input for D, respectively. The proportion of the error probabilities is presumably realistic, as it is more likely for a user to overlook an error and thus cause a miss error than to falsify a correct original state.

### 10.5.2 Base Case

The baseline for the mathematical assessment of the effectiveness of individual data quality enforcement mechanisms is the base setting that exactly one user works on each task T and contributes a respective input, which immediately becomes the result of T. Then, the probabilities  $P_{BC}(\text{'miss'})$  of a miss error and  $P_{BC}(\text{'add'})$  of an add error occurring in a decision D of a task T are

$$\begin{aligned} P_{BC}(\text{'miss'}) &= P(\text{'miss'}) \\ P_{BC}(\text{'add'}) &= P(\text{'add'}) \end{aligned}$$

This results in the following probability of a correct result:

$$P_{BC}(\text{'S}_R(D)=\text{'S}_C(D)) = 1 - P(\text{'S}_O(D)=\text{'S}_C(D)) \cdot P_{BC}(\text{'add'}) - P(\text{'S}_O(D)\neq\text{'S}_C(D)) \cdot P_{BC}(\text{'miss'})$$

This means that there are two ways an incorrect result for a decision D can emerge: either the original state of D is correct and the contributing user falsifies it (incurs an add error), or the original state of D is erroneous and the contributing user fails to correct it (incurs a miss error). If neither of this happens, the result of D is correct.

Note that always  $\text{WorkExp}_{BC}(T) = 1$ , representing optimal throughput. In the scenario the running example, the expected result accuracy is

$$\begin{aligned} P_{BC}(\text{'S}_R(D)=\text{'S}_C(D)) &= 0.94 \text{ and} \\ P_{BC}(\text{'S}_R(T)=\text{'S}_C(T)) &\approx 0.7807. \end{aligned}$$

### 10.5.3 v-Voting

v-Voting is a mechanism countering accidental errors. Like r-Redundancy, it does so by obtaining and aggregating several inputs for each task. As opposed to r-Redundancy, however, it uses an agreement-based input aggregation function, controlled by the so-called *vote-majority parameter*  $v$ . That is, there is a fixed level of agreement to reach, but no fixed number of inputs to obtain. Von Ahn [2008] uses this technique for individual words, with a fixed  $v = 3$ . We generalize it here to a parametric level of agreement, referred to as  $v$ , and for any multi-decision task.

**Result<sub>v</sub>(D)** is the input aggregation function for v-Voting. Formally, this is:

$$\text{Result}_v(D) := \begin{cases} O_0 & \text{if } \exists O_0 \in \text{Opts}(D) \text{ such that } |\text{Contribute}(D, O_0)| \geq v \\ N & \text{otherwise} \end{cases}$$

$\text{Result}_v(T)$  inherently avoids the ambiguous cases that can occur with r-Redundancy, as illustrated in Example 10.2. Another advantage of  $\text{Result}_v(T)$  is that it requires fewer inputs than r-Redundancy for the same expected result quality; namely, tasks of low difficulty require fewer inputs because user inputs easily agree. In addition,  $\text{Result}_v(T)$  computes the result decision-wise and does not require entire inputs to agree, in contrast to [Von Ahn 2008].

Suppose that in a 2-Voting scenario, three users  $U_1$ ,  $U_2$ , and  $U_3$  contribute inputs to the task  $T$  from the running example, and that their inputs are as follows:

$S_1(T, U_1) = (\text{'page header'}, \text{'main text'}, \text{'main text'}, \text{'footnote'})$

$S_1(T, U_2) = (\text{'main text'}, \text{'main text'}, \text{'main text'}, \text{'main text'})$

$S_1(T, U_3) = (\text{'page header'}, \text{'main text'}, \text{'caption'}, \text{'main text'})$

Even though no two inputs are equal as a whole, and all deviate from  $S_C(T)$  in one decision, at least two inputs agree on each decision. Namely, the agreed-upon overall result  $S_R(T)$  is  $(\text{'page header'}, \text{'main text'}, \text{'main text'}, \text{'main text'})$ , which is equal to  $S_C(T)$ , even though none of the users has actually provided this input. Had users  $U_1$  and  $U_2$  given the same overall input, the system would not have obtained an input for  $T$  from  $U_3$  at all.

### Example 10.3: The benefit of decision-wise voting

Example 10.3 illustrates how decision-wise voting can decrease the number of inputs required for a consensus result; the larger the number of decisions a given task consists of, the higher the advantage. An analytic comparison to requiring inputs to agree completely follows below.

**Restrictions of Analytical Evaluation.** To render the analytical evaluation of  $v$ -Voting easy to follow and comprehensible, computations use the assumption set up in the introduction of this section, i.e.,  $|\text{Options}(D)| = 2$  for any given decision  $D$ . Further, considerations are restricted to  $v = 2$ . This ensures that  $|\text{Input}(D)| \leq 3$  for any given decision  $D$ , and with decision-wise voting also that  $|\text{Input}(T)| \leq 3$  for any given task  $T$ , independent of  $|T|$ . The simulations presented below in Section 10.6 cover a substantially wider range of values for both  $v$  and  $|\text{Options}(D)|$ .

**Expected Result Accuracy.** What is the overall probability of a correct result for a task  $T = (D_1, \dots, D_d)$ , i.e.,  $P_V('S_R(T) = S_C(T)')$ ? With  $v = 2$  and  $|\text{Options}(D)| = 2$  for all  $D \in T$ , this computes as follows:

$P_V('miss')$  and  $P_V('add')$  denote the probabilities of a miss error and an add error occurring in the result of a decision  $D \in T$ , respectively.

Informally, if  $v=2$ , an error in the result of a decision  $D$  occurs if the first two inputs are erroneous, and also if one of the two first two and the third input are erroneous. Formally,  $P_V('miss')$  and  $P_V('add')$  compute as:

$$P_V('miss') = 3 \cdot P('miss')^2 - 2 \cdot P('miss')^3$$

$$P_V('add') = 3 \cdot P('add')^2 - 2 \cdot P('add')^3$$

The overall probability for a decision  $D \in T$  to be correct in the result then is:

$$P_V('S_R(D) = S_C(D)') = 1 - P('S_O(D) = S_C(D)') \cdot P_V('add') - P('S_O(D) \neq S_C(D)') \cdot P_V('miss')$$

The overall probability for the result of a task  $T$  with  $d$  decisions  $D_1 \dots D_d$  to be correct then is:

$$P_V('S_R(T) = S_C(T)') = P_V('S_R(D) = S_C(D)', D \in T)^d$$

Example 10.4 illustrates these computations with the values from the running example introduced in Section 10.5.1:

With the exogenous parameters given there, the probability of a correct result for the task from the running example computes as

$$P_V('S_R(D) = S_C(D)') = 0.9886$$

$$P_V('S_R(T) = S_C(T)', D \in T) \approx 0.9552$$

In the base case, things are different.

$$P_{BC}('S_R(D)=S_C(D)', D \in T) = 0.94$$

$$P_{BC}('S_R(T)=S_C(T)') \approx 0.7807.$$

With no correction at all, i.e., with fully automated NLP and no user interaction, it would be, just for comparison:

$$0.8^4 = 0.4096$$

#### Example 10.4: Result accuracy comparison for 2-Voting in the example scenario

In Example 10.4, 2-Voting increases the probability of a correct result for the example task T to about 95.5% from about 78% in the base case. This corresponds to a reduction of error by a factor of about 4, for the at most threefold effort. Note that accuracy, for instance that of classification algorithms, is usually measured for individual objects, which corresponds to the individual decisions of a task. In this example, 2-Voting increases the probability of a correct final result for a decision D of a task T from about 94% to about 99%. This corresponds to a reduction of error by a factor of almost 6 in comparison to the base case, again, for at most three times the effort.

**Throughput.** The actual increase in effort in comparison to the base case depends on the expected number of inputs to obtain until there is an agreed-upon result, denoted as  $WorkExp_V(T)$ . For  $v = 2$ , the latter figure depends on the probability  $P('S_1(T,U_1)=S_1(T,U_2)')$  of the first two inputs to agree on all decisions in T, in other words the probability that two inputs already yield agreement so a third one is not required; this formalizes as follows:

$P('S_1(D,U_1)=S_1(D,U_2)')$  is the probability that the first two inputs  $S_1(D,U_1)$  and  $S_1(D,U_2)$  agree for an individual decision D.

Informally, this is the probability that either none or both  $S_1(D,U_1)$  and  $S_1(D,U_2)$  are erroneous in some way. It is as follows:

$$P('S_1(D,U_1)=S_1(D,U_2)') = P('S_O(D)=S_C(D)') \cdot (P('add')^2 + (1-P('add'))^2) \\ + P('S_O(D) \neq S_C(D)') \cdot (P('miss')^2 + (1-P('miss'))^2)$$

$P('S_1(T,U_1) = S_1(T,U_2)')$  denotes the probability that the first two inputs  $S_1(T,U_1)$  and  $S_1(T,U_2)$  agree for an entire task T.  $\square$

Formally, this is:

$$P('S_1(T,U_1) = S_1(T,U_2)') = P('S_1(D,U_1) = S_1(D,U_2)', D \in T)^{|T|}$$

With this, the expected number of inputs required per task  $WorkExp_{V_2}(T)$  is:

$$WorkExp_{V_2}(T) = 2 \cdot P('S_1(T,U_1) = S_1(T,U_2)') + 3 \cdot P('S_1(T,U_1) \neq S_1(T,U_2)')$$

Further,  $WorkExp_{V_2}(T) / WorkExp_{BC}(T)$  is the overhead 2-Voting incurs in comparison to the base case, and  $1 - WorkExp_{V_2}(T) / WorkExp_R(T)$  is the reduction in effort

2-Voting yields in comparison to 3-Redundancy. Example 10.5 illustrates this for the values from the running example from Section 10.5.1:

With the values from the running example,  $\text{WorkExp}_{v_2}(T)$  computes as:

$$P('S_{i,U_1}(T) = S_{i,U_2}(T)') = 0.6162$$

and thus

$$\text{WorkExp}_{v_2}(T) = 2.3838$$

Compared to the reduction in error, which is by a factor of almost 5 (see Example 10.4), the overhead over the base case is relatively low at a factor of less than 2.5. The reduction in effort as compared to 3-Redundancy is 21%, corresponding to a 26% increase in throughput, at no increase of the probability of errors at all.

Note that these values are for decision-wise 2-Voting; an input aggregation function that requires inputs to agree in all decisions to be considered equal requires an average of about 2.5<sup>22</sup> inputs for the values from the running example. This emphasizes the benefit of decision-wise voting, which reduces the expected number of inputs to obtain without affecting expected result accuracy at all.

#### Example 10.5: Throughput of 2-Voting in the example scenario

Note that in reality both  $P('miss')$  and  $P('add')$  will be far lower than the rather pessimistic values from the example computations. Further, the probability of a correct original state  $P('S_o(D)=S_c(D)')$  is often higher, resulting in a higher probability of the first two inputs to agree, i.e., a higher  $P('S_i(D,U_1)=S_i(D,U_2)')$ . On the other hand, tasks can consist of far more decisions, so the exponent in the computation of  $P('S_i(T,U_1) = S_i(T,U_2)')$  increases, resulting in lower values. Example 10.6 illustrates that depending on the actual numbers, the effect can go either way:

A value of 99% for  $P('S_i(D,U_1)=S_i(D,U_2)')$  in a task with 20 decisions results in 82% for  $P('S_i(T,U_1)=S_i(T,U_2)')$ ; in a task with 50 decisions, the latter is 61%.

#### Example 10.6: Throughput of 2-Voting with other exogenous parameter values

**User Motivation.** With an appropriately designed payoff function,  $v$ -Voting can also foster high-quality inputs; suppose that the total payoff  $\text{Payoff}(T)$  for each task  $T$  is shared between all users who have contributed inputs to  $T$ .

**Payoff<sub>v</sub>(U, T)** is the payoff function for  $v$ -Voting, namely:

$$\text{Payoff}_v(U, T) := \text{Payoff}(T) / |\text{Input}(T)|$$

Then the expected payoff a user  $U$  receives for contributing an input to  $T$  is:

$$\text{PayoffExp}_v(U, T) = \text{Payoff}(T) / \text{WorkExp}_v(T)$$

<sup>22</sup> The computation is highly complex, as the maximum depth of the resulting decision tree is 17 even with  $v = 2$ ,  $|T| = 4$ , and  $|\text{Options}(D)| = 2$  for  $D \in T$ ; the result given here was obtained with the help of a computer program.

This means that  $\text{PayoffExp}_v(U,T)$  increases for each user  $U$  contributing an input to  $T$  if the expected number of required inputs  $\text{WorkExp}_{v2}(T)$  decreases because  $U$  has to share  $\text{Payoff}(T)$  with fewer fellow contributors. Thus, such a payoff function incentivizes users to seek agreeing inputs, so to increase  $P('S_i(T,U_1) = S_i(T,U_2)')$ . For the extreme case that there is no payoff at all if the first two inputs do not agree, Agreement Games have been shown to incentivize inputs of high quality (see Sections 6.6.2 and 10.4.2).

Note that  $v$ -Voting can be combined with other payoff functions as well, notably ones that observe the number of errors in the individual inputs and thus set up even higher incentives for users to avoid errors in their inputs.

#### 10.5.4 Vote Boosting

Vote Boosting is a mechanism that increases the weight of inputs from users who are known to make few mistakes, so to increase throughput without loss of result accuracy. It exploits that likely not all users make mistakes with the same probability, and that  $v$ -Voting allows for individually observing the frequency of any given user  $U$  making mistakes. If  $U$  has made very few mistakes recently, Vote Boosting can give a higher weight to an input from  $U$  in the aggregation function, referred to as *vote boost*. This way, it reduces the number of inputs required for computing a result and thus increases throughput. Vote Boosting is formally described in the following:

**CoinFlip(c)** is a random function that returns 1 with a probability of  $c$  and 0 with a probability of  $(1-c)$ .

**BoostProb(U,T)** is the function that computes the probability that the input  $S_i(T,U)$  of a user  $U$  for a task  $T$  receives a vote boost (referred to as the **boost probability** in the following).

A possible formula for  $\text{BoostProb}(U,T)$  is derived below, starting from a minimum value for the expected result accuracy to maintain. Note that alternative and more sophisticated definitions of  $\text{BoostProb}(U,T)$  than the one used here are conceivable, with more configuration parameters, or more complex computations, or both. The following considerations stick to a relatively simple definition, however, both for clarity and for ease of presentation.

**Result<sub>VB</sub>(T)** is the input aggregation function for Vote Boosting, as follows:

$$\text{Result}_{VB}(T) := \begin{cases} S_i(T,U) & \text{if } |\text{Input}(T)| = 1 \text{ and } \text{CoinFlip}(\text{BoostProb}(U,T)) = 1 \\ \text{Result}_v(T) & \text{otherwise} \end{cases}$$

With this definition of  $\text{Result}_{VB}(\text{Input}(T))$ , the weight of input  $S_i(T,U)$  becomes  $v$  with a probability of  $\text{BoostProb}(U,T)$ , so  $S_i(T,U)$  becomes the result of  $T$  immediately. This circumvents the  $v$ -Voting mechanism and thus reduces  $\text{WorkExp}_{VB}(T)$  to 1, the baseline level, completely eliminating the overhead. However, it also abandons the error-preventing effect of  $v$ -Voting. Thus,  $\text{BoostProb}(U,T)$  should return a value considerably greater than 0 only for users who are very unlikely to make mistakes, or, conversely, for users who are very likely to provide an error-free input for  $T$ . The required probability for the latter to happen is the starting point for the formalization of  $\text{BoostProb}(U,T)$ :



**minCorrProb** denotes the minimum probability required for the result of a task T to be correct.

minCorrProb is an endogenous parameter; its value is to be determined by the operators of a crowdsourcing system that implements Vote Boosting, based on the result accuracy they aim at. A respective configuration strategy is devised in the evaluation section in the course of the discussion of simulation results.

$P('S_I(D,U) = S_C(D)')$  is the probability that a user U provides a correct input for a decision D. The respective probability for a task T is  $P('S_I(T,U) = S_C(T)') = P('S_I(D,U)=S_C(D)', D \in T)^{|T|}$ .

The actual value of  $P('S_I(T,U)=S_C(T)')$  is unknown, but can be estimated from the number of decisions and tasks a given user U has provided correct inputs for since last making a mistake. In particular, this estimation is based on testing the hypothesis " $P('S_I(T,U)=S_C(T)') \geq \text{minCorrProb}$ ", referred to as the *boostability hypothesis*<sup>23</sup>, based on the number of observed error-free inputs from user U. The lower the significance level we can accept this hypothesis with, the higher the boost probability for U. Three further notions help formalizing this:

**Correct(U)** denotes the observed number of correct inputs from user U since his last erroneous input.

**CorrSig(U)** is the post-hoc significance level for accepting the boostability hypothesis based on Correct(U) observed correct inputs.

**maxFalseBoostProb** denotes the maximum acceptable significance level for accepting the boostability hypothesis for a given user, a second endogenous parameter besides minCorrProb.

The computation of BoostProb(U,T) starts with computing CorrSig(U), i.e., a test of significance for accepting the boostability hypothesis for a user U based on Correct(U) observed correct inputs. The quotient of maxFalseBoostProb and CorrSig(U) then becomes an upper bound for the boost probability for U; formally:

$$\begin{aligned} \text{BoostProb}(U, T) &\leq \frac{\text{maxFalseBoostProb}}{\text{CorrSig}(U)} \\ \Leftrightarrow \text{BoostProb}(U, T) &\leq \frac{\text{maxFalseBoostProb}}{P("P('S_{I,U}(T) = S_C(T)' < \text{minCorrProb}")')} \\ \Leftrightarrow \text{BoostProb}(U, T) &\leq \frac{\text{maxFalseBoostProb}}{\frac{\text{Correct}(U)}{\text{minCorrProb}}^{|T|}} \\ \Leftrightarrow \frac{\text{maxFalseBoostProb}}{\frac{\text{Correct}(U)}{\text{minCorrProb}}^{|T|}} &\geq \text{BoostProb}(U, T) \end{aligned}$$

<sup>23</sup> Note that 'hypothesis' does not refer to a research hypothesis in this current context; it refers to the hypothesis that a user has a sufficiently low error probability to be eligible for a vote boost.

Note that this upper bound increases exponentially with  $\text{Correct}(U)/|T|$ . To prevent  $\text{BoostProb}(U,T)$  to grow to or beyond 1, which would factually deactivate voting for user  $U$  and thus might foster cheating,  $(1 - \text{maxFalseBoostProb})$  serves as an additional upper bound for the boost probability. The rationale behind this bound is that the lower  $\text{maxFalseBoostProb}$  is, the slower is the growth of  $\text{BoostProb}(U,T)$ ; a slower growth renders a higher maximum boost probability less critical, as it takes a very large number of correct inputs to achieve. Further,  $\text{BoostProb}(U,T)$  should be 0 for  $\text{Correct}(U) = 0$ , which is easy to achieve by subtracting  $\text{maxFalseBoostProb}$  from the formula derived so far. This finally facilitates a definition of  $\text{BoostProb}(U,T)$ :

$$\text{BoostProb}(U,T) := \min \left( \begin{array}{l} (1 - \text{maxFalseBoostProb}), \\ \text{maxFalseBoostProb} \cdot \left( \frac{\text{minCorrProb}^{\frac{-\text{Correct}(U)}{|T|}}}{-1} \right) \end{array} \right)$$

Example 10.7 illustrates how the boost probability is computed for a given user  $U$  and a given task  $T$ , and how it develops over time as  $U$  contributes more error-free inputs:

Suppose that a given task  $T$  consists of 3 decisions. Further, suppose user  $U$  has contributed a correct inputs to the previous  $\text{Correct}(U) = 100$  decisions. Finally, let  $\text{maxFalseBoostProb} = 1\%$ , and  $\text{minCorrProb} = 99\%$ . Then the probability of boosting the vote of  $U$  is:

$$\text{BoostProb}(U,T) = 0.01 \cdot (0.99^{\frac{-100}{3}} - 1) = 0.4\%$$

For the boost probability to exceed 50% for the given task  $T$  and values of  $\text{maxFalseBoostProb}$  and  $\text{minCorrProb}$ ,  $\text{Correct}(U)$  has to exceed 1173. This means that, for tasks consisting of 3 decisions,  $U$  has to contribute inputs to 391 tasks without making a mistake. When  $\text{Correct}(U)$  becomes 1374, i.e., after 458 tasks of the size of  $T$ , the boost probability finally reaches its upper limit of  $(1 - \text{maxFalseBoostProb}) = 99\%$ .

For values that are less strict, e.g.,  $\text{maxFalseBoostProb} = 5\%$  and  $\text{minCorrProb} = 95\%$ , the boost probability is much higher:

$$\text{BoostProb}(U,T) = 0.05 \cdot (0.95^{\frac{-100}{3}} - 1) = 22.6\%$$

When  $\text{Correct}(U)$  exceeds 141, i.e., after 47 tasks the size of  $T$ ,  $\text{BoostProb}(U,T)$  exceeds 50% with this second set of values. The upper limit of 95% is reached when  $\text{Correct}(U)$  exceeds 176, i.e., after 59 tasks. ■

### Example 10.7: Development of boost probability

**User Motivation.** In addition to increasing throughput, vote boosting can stimulate high-quality inputs if combined with an appropriately designed payoff function: For instance, let there be a fixed total payoff  $\text{Payoff}(T)$  for each task  $T$ , and let this payoff be shared between all users who have contributed an input for  $T$ , as in the previous section. If the input of a user  $U$  receives a vote boost, his reward  $\text{Payoff}(U,T)$  is equal to  $\text{Payoff}(T)$ ; otherwise, he only receives  $\text{Payoff}(T) / \text{WorkExp}_v(T)$ , his usual share of

the reward. With Vote Boosting, the expected payoff for a user  $U$  on a task  $T$  then formally becomes:

$$\text{PayoffExp}_{\text{VB}}(U,T) = \text{Payoff}(T) \cdot \text{BoostProb}(U,T) + (1-\text{BoostProb}(U,T)) \cdot \text{Payoff}(T) / \text{WorkExp}_v(T)$$

Observe that  $\text{WorkExp}_v(T) > 2$ . This means that receiving a vote boost at least doubles the payoff a user receives for his input, rendering it highly desirable for users to first achieve and then maintain a high boost probability. As the only way to do so is to contribute correct a input to each task, Vote Boosting fosters high quality inputs.

### 10.5.5 Sampled Probing

Sampled Probing is a generic measure to discourage cheating. In particular, it tests users for their honesty and penalizes them if they fail such a test. A respective penalization function reduces the expected payoff when cheating to such a degree that making thoughtful inputs becomes the dominant strategy. This holds even if simply submitting the original state of a task as an input takes considerably less time than contributing thoughtfully. Namely, the penalty factor (see below) allows for adjusting the penalization such that cheating is disadvantageous, irrespective of how much it reduces the working time per task.

Sampled Probing generalizes the approaches used by Eckert [2010] and Von Ahn [2008]. Like the latter mechanism, it uses tasks that are already complete, and whose results are therefore known, to probe users for their reliability. It does not do so in every task, however, as this is impractical if tasks are larger than in [Von Ahn 2008] and consist of connected decisions. Instead, Sampled Probing occasionally confronts users with whole tasks whose result is already known.

**User Motivation.** Is a cheating penalization mechanism even necessary, considering that  $v$ -Voting to some degree and Vote Boosting rather strongly foster error-free and thus, presumably, thoughtful input? In order to render contributing thoughtful input advantageous in comparison to cheating in the long haul, the expected payoff per time has to lower with cheating than it is without.

$\text{Time}_H(U,T)$  and  $\text{Time}_C(U,T)$  denote the time it takes a user  $U$  to honestly provide input for a task  $T$  and the time it takes him to submit an input when cheating, respectively. The following considerations generally assume that  $\text{Time}_H(U,T) > \text{Time}_C(U,T)$ , as otherwise cheating would not yield any advantage for any user at all, with or without penalization.

$\text{PayoffExp}_H(U,T)$  and  $\text{PayoffExp}_C(U,T)$  denote the expected payoff user  $U$  receives for contributing thoughtful input to  $T$  and when cheating on  $T$ , respectively.

For cheating to be inefficient, the following must hold:

$$\frac{\text{PayoffExp}_H(U,T)}{\text{Time}_H(U,T)} > \frac{\text{PayoffExp}_C(U,T)}{\text{Time}_C(U,T)}$$

The relation of  $\text{PayoffExp}_H(U,T)$  and  $\text{PayoffExp}_C(U,T)$  strongly depends on the input aggregation function in use:

- With **r-Redundancy**, the two values are equal, so cheating is always advantageous.
- With **v-Voting**,  $\text{PayoffExp}_H(U,T)$  is  $\text{Payoff}(T) / \text{WorkExp}_v(T)$ , thus at most  $\text{Payoff}(T) / v$  (if the first two inputs for T agree), so cheating is advantageous if  $v \cdot \text{Time}_H(U,T) > (v+1) \cdot \text{Time}_C(U,T)$ , i.e., if the increase in tasks worked on (cheated on) compensates the reduction in  $\text{Payoff}(U,T)$  for the individual tasks that results from the additional input for T necessitated by the cheat.
- Finally, with **Vote Boosting**, the maximum value for  $\text{PayoffExp}_H(U,T)$  is close to  $\text{Payoff}(T)$ . Because users who cheat regularly tend to frequently have errors in their inputs, they will hardly receive a vote boost. Thus,  $\text{PayoffExp}_C(U,T)$  is the same with or without Vote Boosting. Consequently, with Vote Boosting, cheating is only advantageous if  $\text{Time}_H(U,T) > v \cdot \text{Time}_C(U,T)$ .

$\mathbf{P}(\mathbf{S}_O(\mathbf{D}) = \mathbf{S}_R(\mathbf{D}))$  denotes the probability that the original state of a decision D equals its result;  $\mathbf{P}(\mathbf{S}_O(\mathbf{T}) = \mathbf{S}_R(\mathbf{T})) = \mathbf{P}(\mathbf{S}_O(\mathbf{D}) = \mathbf{S}_R(\mathbf{D}), \mathbf{D} \in \mathbf{T})^{|\mathbf{T}|}$  denotes the same probability for an entire task T.

Using standard combinatorics yields the following probability of a correct result for a decision D, i.e.,  $\mathbf{P}(\mathbf{S}_O(\mathbf{D}) = \mathbf{S}_R(\mathbf{D}))$ :

$$\mathbf{P}(\mathbf{S}_O(\mathbf{D}) = \mathbf{S}_R(\mathbf{D})) = \mathbf{P}(\mathbf{S}_O(\mathbf{D}) = \mathbf{S}_C(\mathbf{D})) \cdot (1 - \mathbf{P}(\text{'add'})) + \mathbf{P}(\mathbf{S}_O(\mathbf{D}) \neq \mathbf{S}_C(\mathbf{D})) \cdot \mathbf{P}(\text{'miss'})$$

Note that even though this formula looks very similar to the one for the first two inputs for a decision D to agree, it is different because it refers to the equality of the original state of a decision D to its result.

For 2-Voting, the expected payoff  $\text{PayoffExp}_C(U,T)$  then is as follows:

$$\text{PayoffExp}_C(U,T) = ((1 - \mathbf{P}(\text{'cheat'})) \cdot \mathbf{P}(\mathbf{S}_O(\mathbf{D}) = \mathbf{S}_R(\mathbf{D}), \mathbf{D} \in \mathbf{T})^{|\mathbf{T}|} + \mathbf{P}(\text{'cheat'})) \cdot \text{Payoff}(T) / 2 + (1 - (1 - \mathbf{P}(\text{'cheat'}))) \cdot \mathbf{P}(\mathbf{S}_O(\mathbf{D}) = \mathbf{S}_R(\mathbf{D}), \mathbf{D} \in \mathbf{T})^{|\mathbf{T}|} - \mathbf{P}(\text{'cheat'}) \cdot \text{Payoff}(T) / 3$$

The rationale behind this is the following: Assume user U cheats on a task T, i.e., he submits  $\mathbf{S}_O(\mathbf{T})$  as his input; then one more input is required for 2-Voting to yield the result  $\mathbf{S}_R(\mathbf{T})$ . Further, let a second user V provide this input. Now the payoff of user U is as follows: If V submits  $\mathbf{S}_O(\mathbf{T})$  as his input as well, be it due to cheating as well or due to checking honestly and not making any changes,  $\mathbf{S}_R(\mathbf{T})$  emerges after two inputs, so the payoff for both U and V is  $\text{Payoff}(T)/2$ . If V submits something different, i.e.,  $\mathbf{S}_I(\mathbf{T},U) \neq \mathbf{S}_I(\mathbf{T},V)$ , a third input is required for  $\mathbf{S}_R(\mathbf{T})$  to emerge. Consequently, the payoff for U, V and the third contributor is  $\text{Payoff}(T)/3$ .

Example 10.8 illustrates that a user can in fact increase his overall payoff by cheating in a scenario with 2-Voting and Vote Boosting:

Let the global probability of cheating be as high as 20%; with the values of our running example, the expected payoff turns out to be:

$$\mathbf{P}(\mathbf{S}_O(\mathbf{D}) \neq \mathbf{S}_R(\mathbf{D})) = 0.2128 \text{ and } \text{PayoffExp}_C(U,T) = 0.4179 \cdot \text{Payoff}(T)$$

Due to Vote Boosting, the expected payoff for contributing a thoughtful input,  $\text{PayoffExp}_H(U,T)$ , is at most slightly more than twice this value. This means that if cheating on a given task T by submitting its original state as an input takes less than half as long as contributing thoughtfully, cheating is advantageous.

### Example 10.8: Increased payoff due to cheating

**The Sampled Probing Mechanism.** Example 10.9 shows that cheating remains to be penalized if contributing thoughtful input to a task is considerably more effort than cheating, so a dedicated cheating prevention mechanism is in fact necessary. Sampled Probing achieves this by probing users for their honesty: with a certain probability (referred to as the **probe rate**), the mechanism inserts tasks with already-known results in the stream of tasks a user  $U$  gets to work on, compares their input to this already-known result, and penalizes them if they fail the test.

A **Probe  $P$**  consists of a probe task  $T_P$ , its original state  $S_O(T_P)$ , and its result  $S_R(T_P)$ , formally  $P := (T_P, S_O(T_P), S_R(T_P))$ .

**pr** is the **probe rate**, i.e., the probability of a crowdsourcing system probing a user  $U$  with a probe task  $T_P$  belonging to a probe  $P$  instead of presenting him a task the result is yet to be determined for.

**Pass( $P, S_I(T_P, U)$ )** is the function that decides whether or not a user  $U$  who has submitted  $S_I(T_P, U)$  as his input to  $T_P$  passes the probe  $P$ , formally:

$$\text{Pass}(P, S_I(T_P, U)) := \begin{cases} \text{true} & \text{if } \text{Dist}(S_I(T_P, U), S_R(T_P)) \leq \text{Dist}(S_I(T_P, U), S_O(T_P)) \\ \text{false} & \text{otherwise} \end{cases}$$

Probing users now works as follows: whenever a user  $U$  retrieves a task to work on it, the crowdsourcing system instead returns a probe task  $T_P$  belonging to a probe  $P$  with probability  $pr$ . Because  $T_P$  has been sampled, it is similar to other tasks; thus,  $U$  cannot recognize that he is being probed and submits his input  $S_{I,U}(T_P)$  as normal. He passes the probe if  $\text{Pass}(P, S_I(T_P, U))$  returns true; otherwise, the crowdsourcing system considers his input a cheating attempt and punishes  $U$  as described below.

To obtain appropriate probe tasks, Sampled Probing inspects each task  $T$  after its result  $S_R(T)$  is available. If the result differs substantially from the original state of the task, i.e.,  $\text{Dist}(S_O(T), S_R(T))$  is large,  $T$  is a good probe, for two reasons: (1) If  $S_O(T)$  was free from errors, i.e.,  $\text{Dist}(S_O(T), S_R(T)) = 0$ , users would not have to correct anything to generate a correct input, foiling any attempt to catch a user cheating on  $T$ . (2) If a user makes a mistake in a probe task, this has to be distinguishable from cheating, and thus a probe task has to have several errors to correct in its original state, so users can still pass if they make few mistakes.

Example 10.9 illustrates how a probe works, and how a user  $U$  passes or fails; the paragraph-classification task from the running example presented in Section 10.5.1 is a good probe task  $P$  because  $\text{Dist}(S_O(P), S_R(P)) = 3$ .

Suppose user  $U_1$  from Example 10.3 is probed with  $P$  and submits the input  $S_I(P, U_1) = (\text{'page header'}, \text{'main text'}, \text{'main text'}, \text{'footnote'})$ . Even though this input contains a miss error in decision  $D_4$ , it still passes the probe because  $\text{Dist}(S_I(P, U_1), S_R(P)) = 1$  is smaller than  $\text{Dist}(S_I(P, U_1), S_O(P)) = 2$ .

#### Example 10.9: Probing a user

To render cheating a disadvantageous strategy, Sampled Probing introduces a *penalty factor* into the payoff function to reduce the payoff for a user who has recently been caught cheating, i.e., has recently failed a probe. Formally, this means the following:

**LastFail(U)** is the number of tasks user U has contributed inputs to since last failing a probe.

**PenaltyFactor(U)** is the function that provides the **penalty factor** for user U, with **penSev** being a configuration parameter that controls the severity of the penalization:

$$\text{PenaltyFactor}(U) := \begin{cases} \text{pr} \cdot \text{penSev} & \text{if LastFail}(U) < (1/\text{pr}) \\ 1 & \text{otherwise} \end{cases} \quad \square$$

To influence, based on the penalty factor, the payoff a user U receives for contributing an input to a task T, Sampled Probing replaces any given payoff function  $\text{Payoff}_0(U, T)$  with one that enforces penalties:

**Payoff<sub>SP</sub>(U, T)** is the payoff function for Sampled Probing:

$$\text{Payoff}_{\text{SP}}(U, T) := \text{Payoff}_0(U, T) \cdot \text{PenaltyFactor}(U)$$

**User Motivation with Sampled Probing.** Above definition of  $\text{Payoff}_{\text{SP}}(U, T)$  means that after failing a probe, a user U has to contribute inputs to the next  $(1/\text{pr} - 1)$  tasks, which statistically are not probes, to make up for the failed probe. For this effort, he gets an overall reward that is at most **penSev** times the reward for submitting thoughtful input for the probe in the first place. The probability to get caught cheating is equal to the probe rate **pr**. The expected reward for cheating on task T then is:

$$\text{PayoffExp}_{\text{SP,C}}(U, T) = (1 - \text{pr}) \cdot (\text{pr} \cdot \text{p}) \cdot \text{PayoffExp}_C(U, T)$$

The lower **penSev**, the harder the penalization. After a user U has failed a probe, different values for **penSev** mean the following:

- **penSev** = 0 denies user U any payoff for the next  $(1/\text{pr} - 1)$  tasks.
- **penSev** = 1 grants user U exactly the same overall payoff for the next  $(1/\text{pr} - 1)$  tasks that he would have received for providing thoughtful input on the probe he failed.
- Finally, **penSev** =  $(1/\text{pr})$  pins **PenaltyFactor(U)** to 1 and thus alleviates the penalization completely.

Thus, values between 0 and  $(1/\text{pr})$  for **penSev** represent different severities of penalization. Values above  $(1/\text{pr})$  would turn penalization into a reward, as such values increase the penalization factor beyond 1 after a user has failed a probe.

For a probe rate **pr** = 10% and a penalty factor **p** = 1, for instance, Sampled Probing reduces the expected reward for continuous cheating to 9% of what it would be without Sampled Probing. Thus, cheating does not pay off any longer. This is the case even if cheating on a task is considerably less effort / less time-consuming than contributing honestly.

#### **Example 10.10: Expected payoff for cheating with Sampled Probing**

Example 10.10 illustrates that Sampled Probing is a promising means to discourage cheating, if at the cost of some throughput, as explained below. Due to its parameters **pr** and **penSev**, it is sufficiently flexible; namely, the value of the configuration parameter **penSev** can be chosen to render cheating inefficient for any time advantage it may have over thoughtful contribution. The lower the probe rate is, the lower is the

expected reward for cheating, and the lower is the impact of the probes on throughput. However, users also have to perceive a realistic threat of being probed for the measure to be effective, so  $pr$  should not be too low.

**Throughput.** Users cannot contribute input to real tasks while they are working on probe tasks, and thus probing does decrease throughput to a degree equal to the probe rate  $pr$ . Consequently, the probe rate should be relatively low, but not too low, either, as explained before.

## 10.6 Simulations

All three mechanisms presented in this chapter were evaluated in extensive simulations, ranging many parameter combinations and many variations of input aggregation and payoff functions.

### 10.6.1 Experimental Setup

The **sets of tasks** have two parameters: the number of options per decision, and the accuracy of the initial states. 9 sets of 1,000,000 tasks each were generated, with 2, 3, or 4 options per decision and 80%, 90%, and 95% as the accuracy for the original states. Each task consists of 5 to 10 decisions, normally distributed over that interval.

The **user populations** tested have two parameters: their mean probabilities of cheating and of mistaking. Mean values of 1%, 4%, and 15% for both probabilities were used for generating populations of 1000 users for each of the resulting 9 combinations. For the individual users, the probabilities of cheating and of making errors by mistake were exponentially distributed over  $[0,1]$  around the respective mean values.

The simulated users behave as follows: When by mistake making an add error on a decision with more than two options, a user selects one of the erroneous options at random with equal likelihood. Users take a fixed time  $t$  per decision when contributing thoughtfully. Changing the state of a decision increases this time to  $2 \cdot t$ , whereas cheating decreases this time to  $t/2$ . At runtime, each user is a separate thread, so users are independent of each other and work concurrently.

In all, the simulations cover 181 **input aggregation functions**: One is the base case, i.e., each task receives one input. The other 180 are as follows:

- $r$ -Redundancy with  $r = 3,5,7$
- $v$ -Voting with  $v = 2,3,4$
- the latter combined with 14 different parameter combinations for Vote Boosting, one being to deactivate it, the other 13 are different combinations of values for `minCorrProb` and `maxFalseBoostProb`
- all above setups without Sampled Probing, and with probe rates of 1%, 4%, and 15%.

There is a fixed payoff  $\text{Payoff}(D)$  per decision; the **payoff function** distributes  $\text{Payoff}(D)$  equally among all users who have contributed an input. Users also receive a payoff for providing input for probe tasks if they pass it. The rationale behind this decision is that a passed probe is a thoughtful input and deserves a payoff. In addition, crowdsourcing services like the Amazon Mechanical Turk [AMT] require a per-task payoff, so this decision reflects reality.

### 10.6.2 Results

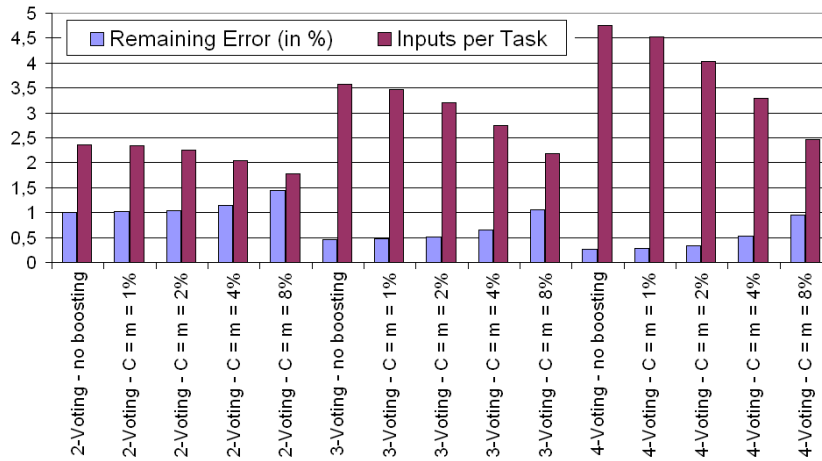
From a total of 14,661 simulated scenarios, the following four analyses are by far the most interesting:

**v-Voting vs. r-Redundancy.** Table 10.1 shows the average result quality and the average number of inputs per task for v-Voting and r-Redundancy against each other. For fairness, the numbers for v-Voting exclusively come from input aggregation functions that do not use Vote Boosting. All numbers are aggregated over all user populations, task sets, and settings of Sampled Probing.

	Base Case	3-Red.	2-Voting	5-Red.	3-Voting	7-Red.	4-Voting
Remaining Error (in %)	4,25	1,11	1,01	0,48	0,46	0,27	0,27
Inputs per Task	1	3	2,36	5	3,57	7	4,75

**Table 10.1: Inputs per task and remaining error**

Clearly, v-Voting is superior to r-Redundancy in terms of throughput, requiring significantly fewer inputs for the same result quality. This substantiates the results of the analytical assessment. Interestingly, result quality also improves slightly with 2-Voting and 3-Voting in comparison to 3-Redundancy and 5-Redundancy, respectively. This is likely because v-Voting avoids the ambiguous decisions that can occur in r-Redundancy. From a different angle, v-Voting increases the data quality achievable with a given maximum number of inputs: 4-Voting requires even less inputs than 5-Redundancy, yet halves the number of remaining errors.



**Figure 10.1: Effects of Vote Boosting**

**Vote Boosting.** Figure 10.1 visualizes the impact of Vote Boosting, namely the increase in throughput and in errors. The effect of changes to the other 13 are different combinations of values for minCorrProb and maxFalseBoostProb (abbreviated as C and m in Figure 10.1 for readability) is similar for all three values tested for



v: The more liberal the parameter settings, the higher the increase in throughput, but the number of errors is higher as well; the dependency seems almost linear for both. For a given result quality required, this predictable behavior allows system designers to tune the parameters to achieve the highest throughput possible while meeting their given accuracy goals.

**Sampled Probing.** Figure 10.2 graphs the cheating tendency of users against their expected payoff per task and overall, without probing and for different probe rates; setups with Vote Boosting are excluded from this aggregation in order to isolate the effects of probing. Sampled Probing turns out to serve its purpose well, as the overall payoff for frequently cheating users is less than for users who contribute thoughtfully, even though the former submit a considerably higher number of inputs. The fact that the per-task payoff increases for higher probe rates is due to the payoff users receive for working on probe tasks.

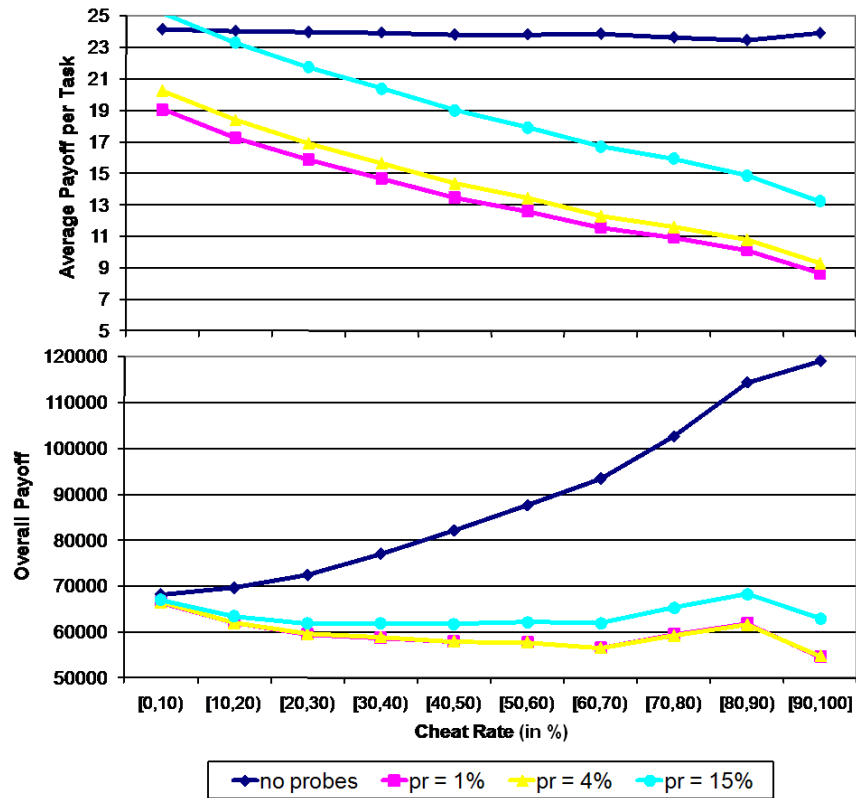


Figure 10.2: Per-task and overall payoff depending on cheating

**Cost of High-Quality Results.** Table 10.2 shows the average number of inputs required for each task to achieve at least 99.5% accuracy in the result, broken up across the 9 different user populations. The accuracy actually achieved is given in brackets, with the parameters of the input aggregation function listed beneath. The

input aggregation function always uses v-Voting (parameter v), mostly with Vote Boosting (parameters minCorrProb and maxFalseBoostProb, again abbreviated as C and m for readability), and some with Sampled Probing (parameter pr). A value of 0 for pr or m indicates that Sampled Probing or Vote Boosting were not used in the respective input aggregation function, respectively. These results point out the correlation between the capability and honesty of contributing users and crowdsourcing throughput; the latter translates directly into the per-task cost in scenarios with a per-input payoff, e.g., the Amazon Mechanical Turk [AMT]. With low probabilities for both mistakes and dishonesty, 1.14 inputs per task are sufficient to achieve the desired accuracy. This figure increases sharply if either of the two probabilities increases. With pessimistic values for both, even 5.38 inputs per task are not enough to reach the goal, highlighting the importance both of fostering high-quality inputs and of deterring users from cheating.

Mean probability of Cheating	Accidental Errors		
	1%	4%	15%
1%	<b>1.14 (99.51%)</b> v=2 pr=0 m=8% C=92%	<b>1.78 (99.63%)</b> v=2 pr=0 m=4% C=96%	<b>3.78 (99.55%)</b> v=3 pr=0 m= 2% C=98%
4%	<b>1.42 (99.57%)</b> v=2 pr=0 m=4% C=96%	<b>1.93 (99.51%)</b> v=2 pr=0 m=4% C=96%	<b>4.48 (99.51%)</b> v=4 pr=1% m=4% C=96%
15%	<b>3.94 (99.65%)</b> v=4 pr=0 m=2% C=98%	<b>4.6 (99.61%)</b> v=4 pr=0 m=2% C=98%	not achieved 5.38 (98.62%) v=4 pr=0 m=0

**Table 10.2: Inputs required for achieving 99.5% result accuracy**

**Crowdsourcing Strategy.** As the simulations have shown, the best suited strategy to achieve a desired result quality at as much throughput as possible strongly depends on the exogenous parameters. These parameters are hardly predictable at the start of a crowdsourcing project. Thus, it seems promising to start out on pessimistic assumptions, i.e., initially choosing a setup that favors result quality over throughput; later, experts can assess the result quality achieved so far (e.g. from a sample of task results) and deduce the actual values of the exogenous parameters. Afterwards, the endogenous parameters can be adjusted to optimize throughput. Operators of crowdsourcing systems should repeat this cycle of assessment and adjustment periodically to become aware of and react to changes in their user base, for instance.

## 10.7 Deployment Experience

To assess the data quality enforcement mechanisms and their impact on user behavior in a real-world scenario, they have been implemented and deployed in a Facebook Application [Facebook] that crowdsources page structure correction in digitized legacy documents; Figure 10.3 shows an exemplary task.

**What to do in this dialog?**  
 Please make sure the boxes in this page reflect the logical paragraphs, and also specify their type:

- main text paragraph, the continuation of one, or a heading, e.g. of a section or chapter
- footnote, the continuation of one from an earlier page or column, or a bibliographic reference
- caption, e.g. of a figure or table, an actual table in itself, a page header or footer
- OCR artifact that is not actually text at all, but rather a stain on the page, a hand written mark, etc

Use boxes of type **Column** to group paragraphs into page columns.  
 Manipulate a box through the buttons in its corners: R for resizing, X for removing, T for changing the type.  
 Draw a new box by pressing, dragging, and then releasing the left mouse button.

**Westafrikanische Ameisen** **137**

sich Querriefung. Die Vorderfläche des Pronotums und die Umgebung der Okzipitaldornen tragen ähnliche stachelartige, spitze Höcker wie bei *C. erinaceus*.

Die langen Epinotaldornen stehen an ihrer Basis etwas weniger auseinander als sie lang sind und divergieren unter einem großen, spitzen Winkel; mit dem Abfall des Epinotums bilden sie, von der Seite gesehen, einen ungefähr rechten.

Der 1. Stielchenknoten ist etwas gedrungener als bei *C. erinaceus*, von oben betrachtet fast sechseckig, am vorderen Gelenk breiter als am hinteren. Unten trägt er den bekannten, pflugscharähnlichen Fortsatz mit nach hinten und unten gerichteter Spitze, der 2. Stielchenknoten einen kräftigen, nach vorn gerichteten und etwas nach unten gebogenen Dorn. Beide Knoten sind wie der Thorax grob längsgerieft und höckerig, der vordere so, daß sich die wenigen, gewundenen Leisten auf der Hinterfläche im Bogen vereinigen. Der zweite zeigt hinten über seiner Einlenkung schwache Querfurchen.

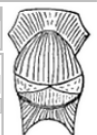

Das länglich ovale Abdomen ist sehr fein längsgestreift, vorn und an den Seiten etwas stärker.

Die Oberschenkel tragen regelmäßige Längsriefen, denen des Thorax gleich.

Schwarz, glänzend, das Abdomen etwas matter; Unterschenkel, Fühlerschäfte und Augen rotbraun. Körper mit starren, abgestumpften Borsten von gelber Farbe sparsam bekleidet, auf dem Ende des Abdomens und dessen Unterseite etwas länger und zahlreicher.

♂. — Länge 8 mm; — Skulptur der Körperoberfläche wie beim ♂, aber noch größer. Epinotaldornen kurz und platt. Die Form des Thorax und den Verlauf seiner Leisten gibt die nebenstehende Figur wieder.

♀. — Länge 6 mm; — Kopf mit nur wenigen, zarten, aber scharf ausgeprägten Längsleisten, die weit voneinander entfernt sind, und deren spärliche Anastomosen einige weite Maschen bilden. Leisten am Ursprung der Anastomosen mit spitzen Höckern. Okzipitaldornen stark entwickelt. Epinotaldornen wie bei ♂. 1. Stielchenknoten hinten etwas in die Länge gezogen. Die Form des Thorax stellt die nebenstehende Figur dar.

T	Main Text	♂, ♂ aus Jaundestation (Zenker).
	Continue Main Text	♂, ♀, fernandensis n. v. <input checked="" type="checkbox"/>
	Heading (of Chapter, Section, etc)	em Kopf 6 mm.
	Page Header / Footer	
	Caption (of Figure, Table, etc)	st wie beim Typus, aber feiner und bedeutend
flache	Table	n. Sie verlaufen gewundener, so daß die Längs-
skulp	Footnote	st so ausgeprägt ist als bei ersterem. Die Skulptur
	Continue Footnote	es Kopfes; doch sind auch hier die Leisten viel
des k	Citation / Bibliographic Reference	von den Dornen der scharfkantigen Leiste jeder-
flache	OCR Artifact	seits
	Text Column (for grouping only)	ore entwickelt.

♂ von Fernando Po (Zenker).

♂♂. *Catantopus sulcatus* n. sp. v. *alenensis* n. v.

♂. — Länge mit ausgestrecktem Kopf 5,5 mm.

Kopfleisten weniger regelmäßig in der Längsrichtung verlaufend, vielfach gewunden, mit zahlreichen Anastomosen, so daß die Skulptur eine mehr netzartige

OK OK & New Skip Cancel

Figure 10.3: A page structuring task in the Facebook application

### 10.7.1 Research Hypotheses

In total, there are three hypotheses regarding the user motivation capabilities of the three mechanisms to test in the deployment:

1. 2-Voting makes users seek agreement to increase their payoff, and thus fosters thoughtful inputs.
2. Vote Boosting motivates users to make highly accurate inputs to increase their payoff.
3. The Sampled Probing mechanism effectively discourages users from cheating.

Besides testing these three hypotheses, a further goal is to assess which probabilities of mistaking and of cheating are to be expected in a real-world setting. Respective figures for a somewhat different setup with a completely different payoff function can be found in Eckert [2010]: The payoff per input was fixed to a value known to the users because the Amazon Mechanical Turk [AMT] platform requires this, and no attempt was made to measure the effect of users knowing the tasks included probe decisions with results known to the system. The hypothesis Eckert [2010] tested was coarser as well, namely whether or not a group of lay users participating in a crowdsourcing system could produce data processing results of the same quality as a respective expert group.

### 10.7.2 Experimental Setup

The **documents** the application processes are from the AntBase Collection [Agosti 2005]; imported into the applications from Internet Archive [KahleIA] as raw OCR results in the DjVu XML format [DjVu].

The **users** are regular Facebook users, starting with the friends of the author and spreading out from there. For their contribution, users gain **scores** that are visible in a public ranking. In addition, users can earn **medals** for high amounts of contributions (contribution awards) and streaks of highly accurate contributions (accuracy awards), and for inviting highly productive users to participate in the application (community building awards), each in bronze, silver, and gold. These medals are on public display as well, and at the recipient's choice go to their wall for everyone to see.

The application distinguishes **six user groups**, each with a different explanation of the input aggregation function. This approach has been chosen to measure differences in user behavior that originate from the opportunities the application offers him to optimize his payoff. In particular, three of the six groups are informed about Sampled Probing, while the remaining three are not, so to test the psychological cheating prevention effect of that mechanism (Hypothesis 3). Out of each threesome, the first group has no information about voting at all, representing the base case of one input per task, the second group is informed that there is a 2-Voting mechanism, and the third group gets to know that there is a 2-Voting mechanism combined with Vote Boosting. This trisection facilitates assessing whether 2-Voting makes users seek agreement by contributing thoughtful inputs (Hypothesis 1), and whether Vote Boosting in addition motivates users to avoid errors in order to maximize their boost probability and thus their payoff (Hypothesis 2).

The application assigns each user to one of six groups at random when he visits it for the first time. To obtain significant results, the initial hope was to attract at least 600 users to contribute on a somewhat regular basis, about 100 per group. This would have yielded highly significant results.

In reality, the **input aggregation function** the application uses is 2-Voting and Sampled Probing; Vote Boosting is present only virtually, so to test its effect on user behavior, as actual use of Vote Boosting would render the assessment of result quality, identification of actual cheating attempts, etc. too complicated.

### 10.7.3 Results

The whole endeavor of having the Facebook user community correct the page structure in digitized legacy documents has to be considered a failure; over the course of a year, the application failed to attract more than a few regular users. The data collected is by orders of magnitude too little to allow for any conclusions regarding the research hypotheses set up in Section 10.7.1.

### 10.7.4 Discussion

The hope that a crowdsourcing application for correcting the page structure in digitized legacy documents would spread and flourish in a mostly fun-oriented online user community like Facebook has turned out elusive. Not even 20 of the 500,000,000 Facebook users found it sufficiently interesting to participate.

Arguably, it is more than doubtful for the vast majority of the Facebook users that they even got aware of the application, which indicates the need for a sophisticated user acquisition strategy, maybe involving Facebook internal advertisement campaigns and similar measures. However, the lack of success also indicates that correcting the structure of digitized document pages just does not have the appeal of collaboratively searching for signs of aliens or classifying galaxies, as in SETI@home or GalaxyZoo, respectively. In particular, the amount of data processed in the Facebook application discussed here is similar to the amount of data processed by Distributed Proofreaders. The reason may well be that the tasks have just the same appeal and ‘coolness factor’, or lack of the latter.

A promising means to foster contribution is to offer material rewards to contributing users; the data quality enforcement mechanisms presented in Section 10.5 have been proven to be up to the challenges that arise from such a scenario. A minor problem with Amazon Mechanical Turk in this context is that it requires specifying a fixed reward users get for contributing an input, which is incompatible with the flexible adjustment of the payoff used in all of the mechanisms presented here.

## 10.8 Conclusions

Mathematically and in simulations, the data quality enforcement mechanisms for crowdsourcing scenarios presented in this chapter are sure to work, clearly achieving Goal 9. The attempted real-world deployment failed to attract sufficient users to verify these findings, however, so whether or not the mechanisms achieve Goal 10 remains open.

In particular, **v-Voting** increases throughput over the static redundancy based approaches used in previous work by means of more sophisticated aggregation of the individual inputs. **Vote Boosting** builds upon v-Voting, further increasing throughput by capitalizing on especially capable users, and especially rewarding these users in

addition. **Sampled Probing** finally tests users for their honesty and punishes them in case of a failed test, so to discourage cheating and thus prevent do away with its negative effect on data quality.

Extensive simulations over a wide range of exogenous parameters have confirmed the suitability of all three mechanisms, substantiating the findings from theoretical analyses. In particular, simulation results show (1) that v-Voting yields higher result quality than r-Redundancy with fewer inputs per task, (2) that Vote Boosting allows trading result quality for throughput in a predictable fashion, and (3) that Sampled Probing turns cheating into a losing strategy even in the long haul.

What remains is to create a deployment scenario that renders tasks like correcting the page structure in digitized legacy documents sufficiently appealing to achieve Goal 10 in practice, i.e., to attract a community of users large and powerful enough to handle the vast amounts of raw document page images produced by projects like Biodiversity Heritage Library in reasonable time. Only this will enable experts to focus on the work they are qualified for, namely to extract the enormous amount of human knowledge contained in these pages with the help of the techniques presented in earlier chapters of this work.

As a byproduct, this chapter has developed a mathematical framework for assessing the effectiveness of data quality enforcement mechanisms in crowdsourcing scenarios that is likely applicable to a wide variety of further mechanisms as well.

## 11 Conclusions & Future Work

The generation of highly accurate comprehensive semantic markup is a prerequisite for information extraction from digitized scientific legacy documents. Namely, only such markup makes the huge amount of scientific data available to applications like mesh-ups, sophisticated visualization, and machine reasoning. Generating semantic markup of high quality is a challenging process, however, and involves considerable effort for expert users from the domains the documents belong to.

This work has developed approaches to assist the expert users in several different ways and to thereby mitigate their effort. Except for the crowdsourcing, all of those approaches have proven highly effective both in laboratory experiments and in real-world deployments; the latter were two projects that created semantic markup for over 3,000 pages of digitized legacy document from the biosystematics domain. In all, the deployed visualization techniques, editing assistance functions, optimizations of NLP tools, process optimizations, and process control mechanisms have reduced the effort by over one and a half orders of magnitude. Once appropriate incentives are in place for the crowdsourcing application, the latter will once again half the effort of the expert users.

Thus, the work presented in this thesis significantly lowers the effort and thus the cost for creating high quality semantic markup, a big step towards the latter eventually becoming realistic for large collections of documents, like the one digitized by Biodiversity Heritage Library. Only then, real world semantic web data, both general domain and scientific, can become available at large scale, and only then the semantic web can evolve into a widely used reality.

Besides further detail level optimizations to NLP algorithms and the way they learn from the data they process and the corrections users make to their results, a major direction of future work results from the starting point of the semantic markup generation process investigated in this work, which is HTML or XML documents that come out of OCR with a good share of the page structure already properly marked.

The effort users have to invest to create high-quality OCR output whose markup reflects the page structure with good accuracy has yet to be considered. It seems highly promising to optimize OCR software based on the same principles this work has developed for semantic markup generation, be it in interactive desktop OCR applications or fully automated mass OCR servers. A further possible improvement is to extend OCR software in a way that prepares their output better for semantic markup generation, namely to integrate paragraph classification with their page structuring features to better prepare documents for structural cleanup and normalization. The rationale is that, for instance, the type of a paragraph is likely by far easier to determine for both classification algorithms and correcting users based on a page image than based on the HTML documents the TaxonX Process starts from in this work.

In other words, the goal is to extend the TaxonX Process to start out right from the page images and to integrate and optimize it and all its individual steps from there to the comprehensive semantic markup required to make the knowledge contained in the written document available to the semantic web.





## Acknowledgements

There are several people I want to thank for their support in writing this thesis, an in performing all the research that culminated in it. Without their support, assurance, guidance, advice, and teaching, this thesis simply would not exist.

First of all, I want to thank my parents Ruth and Axel Sautter and my brother Dominik for always assuring me and believing in me, and for all the different ways they endeavored to make me know. Likewise, I want to thank my friends (the three I mean know that it's them) for their continued support. The same goes to my fraternity brothers, who regularly got me out of self talk when I was pondering problems late at night, listened to my explanations and, even though they often could not understand them in full, allowed me to phrase my thoughts and this way often helped me find a solution.

On a somewhat more subject-specific level, I want to thank all the fantastic people I got to work with on projects, especially the Plazi people. Their expertise in the application domain of biosystematics and their teaching of the same facilitated the practical parts of this thesis, and both their patience and their insistence as users allowed the implementations of the presented concepts to grow into the actually useful applications they have become. In addition, their continued promotion of the latter paved the way for new user groups and new application domains initially unthought-of. Those seminal people are (in alphabetical order) Donat Agosti, Terry Catapano, Chris Freeland, Gregor Hagedorn Brian Heidorn, Hubert Höfer, Norman F. Johnson, Christiana Klingenberg, Richard Pyle, Thomas Stierhof, and Manfred Verhaagh.

Also, I want to thank my colleagues at IPD for all the helpful and inspiring discussions, and for their invaluable input on oral presentations. Special thanks go to Stephan Schosser and Björn-Oliver Hartman for their dedicated support regarding empirical methodology, game theory, and experimental designs, and also to Frank Eichinger for taking responsible care of supplying good coffee, which I got to refer to simply as fuel. Another thanks I want to pay to the network administrators of both the computer science department and my home institute for helping me with my out-of-line requests, for trusting me with the same, and for handling security policies correspondingly.

Finally, I want to thank my advisors Klemens Böhm and Robert A. Morris; they spoke the two seminal sentences that got the whole endeavor underway: "Get us some visibility" and "That's a nice tool, we want to have this". I want to thank my secondary advisor Robert A. Morris for teaching me all of his understanding of taxonomic names and respective literature, as well as its modeling, and for always keeping my engineering thoughts on generic tracks. Last and most important, I want to thank my primary advisor Klemens Böhm for his guidance that kept me on the scientific track, as well as for granting me the freedom and autonomy I had, and for his patience with what I made out of them.



## References

- ABBYY: Abbyy FineReader, [http://www.abbyy.com/finereader\\_ocr/](http://www.abbyy.com/finereader_ocr/)
- Agosti 2003: D. Agosti, Encyclopedia of life: should species description equal gene sequence?, Trends Ecol. Evol. 18: 273, 2003
- Agosti 2005: Agosti, D., Johnson, N. F. Antbase, <http://antbase.org/>. 2005.
- AMT: The Amazon Mechanical Turk, <http://www.mturk.com>
- Anderson 2002. Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D. SETI@home: an experiment in public-resource computing. Communications of the ACM 45:56-61, 2002. doi: <http://doi.acm.org/10.1145/581571.581573>
- Apache Lucene: Apache Lucene, [lucene.apache.org/java/docs](http://lucene.apache.org/java/docs)
- Aquaint: D. Graff. The acquaint corpus of English news text. Linguistic Data Consortium, Philadelphia, 2002.
- Bahl1 1989: Bahl, L. R., Brown, P. F., deSouza P. V., Mercer, R. L. A tree-based statistical language model for natural language speech recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing 36 (7), 1989.
- Bancilhon 1981: Bancilhon, F., Spyratos, N. Update semantics of relational views. ACM Transactions on Database Systems, 6 (4), 1981.
- BHL: The Biodiversity Heritage Library, <http://www.biodiversitylibrary.org>
- Bledsoe 1973: Bledsoe, W. W., Bruell, P. A man-machine theorem proving system. In Proceedings of International Joint Conference on Artificial Intelligence, Stanford, CA, USA, 1973.
- Blum 1998: Blum, A., Mitchell, T. Combining labeled and unlabeled data with co-training. COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann, 1998, p. 92-100.
- BPEL: Business Process Execution Language (BPEL) [http://www.bpelsource.com/bpel\\_info/spec.html](http://www.bpelsource.com/bpel_info/spec.html)
- Brants 2000: Brants, T. TnT: a statistical part-of-speech tagger. In Proceedings ANLC 2000, Seattle, WA, USA, 2000. doi: <http://dx.doi.org/10.3115/974147.974178>
- Brazma 2006: Brazma, A., Krestyaninova, M., Sarkans, U. Standards for systems biology. Nature Reviews Genetics 7, pp. 593–605, 2006.
- Butler 2006: D. Butler, Mashups mix data into global service. Nature 439: 6-7, 2006
- Catapano 2006: Catapano, T. et al. TaxonX: A Lightweight and Flexible XML Schema for Mark-up of Taxonomic Treatments. In Proceedings of TDWG 2006, St. Louis, MO, USA, 2006.
- Chieu 2002: Chieu, H. L., Ng, H. T. Named entity recognition: a maximum entropy approach using global information. In Proceedings of COLING 2002. Taipei, Taiwan, 2002.
- Chinchor 1997: Chinchor, N. MUC-7 Named Entity Task definition. In Proceedings of Message Understanding Conference, Washington, DC, USA, 1997.
- Christensen 2007: Christensen, L., Experimental Methodology, 10th ed., Pearson, Boston, MA, USA 2007, ISBN 0-205-48473-5
- Cohen 1988: J. Cohen, Statistical Power Analysis for the Behavioral Sciences, 2nd ed., Erlbaum, Hillsdale, NJ, USA, 1988, ISBN 0-8058-0283-5
- Cohn 1994: Cohn, D., Atlas, L., Ladner, R. Improving generalization with active learning. Machine Learning 15 (2), 1994. doi: 10.1007/BF00993277
- Collins 1999: Collins, M., Singer, Y. Unsupervised Models for Named Entity Classification. In Proceedings of EMNLP 1999. New Brunswick, NJ, USA, 1999.

- Collins 2002: Collins, M. Discriminative training methods for hidden Markov models: Theory and experiments with the perceptron algorithm. In Proceedings of EMNLP 2002. Philadelphia, PA, USA, 2002.
- Collobert 2008: Collobert, R., Weston, J. A unified architecture for natural language processing: deep neural networks with multitask learning. In Proceedings of ICML 2008. Helsinki, Finland, 2008.
- Cooper 2010: Cooper, S., Khatib, F., Treuille, A., Barbero, J., Lee, J., Beenen, M., Leaver-Fay, A., Baker, D., Popovic, Z. Predicting protein structures with a multiplayer online game. *Nature* 466, 2010.
- Cortes 1995: Cortes, C., Vapnik, V. Support-Vector Networks. In *Machine Learning*, 20, 1995.
- Cristianini 2000: Cristianini, N., John Shawe-Taylor, J. An Introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press, 2000.
- Cucchiarelli 1998: Cucchiarelli, A., Luzi, D., Velardi, P. Automatic semantic tagging of unknown proper names. In Proceedings of COLING 1998, Montreal, Canada, 1998.
- Cucerzan 1999: Cucerzan, S., Yarowsky, D. Language Independent Named Entity Recognition Combining Morphological and Contextual Evidence. In Proceedings of the 1999 Joint SIGDAT Conference on EMNLP and VLC, 1999
- DarwinCore: DarwinCore2, <http://darwincore.calacademy.org/>
- DjVu: <http://www.djvu.org/>
- Donisthorpe 1946: Donisthorpe, H. S. J. K. *Ireneopone gibber* (Hym., Formicidae), a new genus and species of myrmicine ant from Mauritius. *Entomologists Monthly Magazine* (82), pp. 242-243. 1946
- DublinCore: DublinCore ISO Standard 15836: [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_ics/catalogue\\_detail\\_ics.htm?csnumber=52142](http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=52142)
- Duda 2001: Duda, R. O., Hart, P. E., Stork, D. G. *Unsupervised Learning and Clustering*. In: *Pattern Classification* (2nd edition), Wiley, New York, 2001, ISBN 0-471-05669-3.
- Eckert 2010: Eckert, K., Niepert, M., Niemann, C., Buckner, C., Allen, C., Stuckenschmidt, H.: Crowdsourcing the assembly of concept hierarchies. In: *Proceedings of JCDL 2010*, Brisbane, Australia, 2010.
- Endres 1998: Endres-Niggemeyer, B. *Summarizing Information*. 1998. ISBN 3-540-63735-4. Facebook: <http://www.facebook.com>
- Fellbaum 1998: Fellbaum, C., editor. *WordNet: An Electronic Lexical Data-base*. MIT Press, Cambridge, MA, USA, 1998.
- FindIT: UBIO FindIT, [names.mbl.edu/tools/recognize](http://names.mbl.edu/tools/recognize)
- Freund 1999: Freund, Y., Schapire, R. E. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning* 37 (3), 1999. DOI: 10.1023/A:1007662407062.
- Garside1987: Garside, R., Leech, G., Sampson, G. *The Computational Analysis of English*. Longman, 1987.
- GATE: General Architecture for Text Engineering, <http://gate.ac.uk>
- GeoNames: GeoNames, <http://www.geonames.org>
- Gerner 2010: Gerner M., Nenadic, G. and Bergman, C. M. (2010) LINNAEUS: a species name identification system for biomedical literature. *BMC Bioinformatics* 11: 85.
- Gillingham 2006: E. Gillingham, Blackwell launches 3000 years of digitized journal backfiles. *Blackwell Publishing Journal News* 15, July 2006.
- Grishman 1996: Grishman, R., Sundheim, B. Message Understanding Conference - 6: A Brief History. In Proceedings of COLING 1996, Copenhagen, Denmark, 1996.
- Hertz 1990: Hertz, J., Palmer, R. G., Krogh. A. S. *Introduction to the theory of neural computation*. Perseus Books, 1990. ISBN 0-201-51560-1.
- Hirschman 2001: Hirschman, L., Gaizauskas, R. Natural Language Question Answering. *The View from Here. Natural Language Engineering* (2001), 7:4:275-300, Cambridge University Press.

- HNS: Johnson, N. F. The Hymenoptera Name Server. Home Page: <http://atbi.biosci.ohiostate.edu:210/hymenoptera/nomenclator>
- Hütter 2008: Hütter, C., Kühne, C., Böhm, K. Peer production of structured knowledge - an empirical study of ratings and incentive mechanisms. In Proceedings of CIKM 2008, Napa Valley, CA, USA, 2008. doi: 10.1145/1458082.1458192
- Isozaki 2002: Isozaki, H., Kazawa, H. Efficient support vector classifiers for named entity recognition. In Proceedings of COLING 2002, Taipei, Taiwan, 2002.
- Kaelbling 1996: Kaelbling, L. P., Littman, M. L., Moore A. W. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* 4, 1996.
- KahleIA: B. Kahle. The internet archive, <http://www.archive.org>.
- Kim 2003: Kim, J.-D., Ohta, T., Tateisi, Y., Tsujii, J. GENIA corpus – a semantically annotated corpus for bio-text-mining. *Bioinform-matics*, pp. i180-i182, Oxford University Press, 2003.
- Knowtator: Ogren, P. V. Knowtator: a protégé plug-in for annotated corpus construction. In Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, New York, New York, USA, 2006. DOI: <http://dx.doi.org/10.3115/1225785.1225791>
- Kolawa 2007: Kolawa, A.; Huizinga, D. *Automated Defect Prevention: Best Practices in Software Management*. Wiley-IEEE Computer Society Press, 2007
- Koning 2005: D. Koning, N. Sarkar, T. Moritz, TaxonGrab: Extracting Taxonomic Names from Text, *Biodiversity Informatics*, 2005
- Koskenniemi 1990: Koskenniemi, K. Finite-state parsing and disambiguation. In Proceedings Computational Linguistics 1990. Morristown, NJ, USA, 1990.
- Lafferty 2001: Lafferty, J., McCallum, A., Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proceedings ICML, San Francisco, CA, USA, 2001.
- Lewis 1994: Lewis, D. D., Catlett, J. Heterogeneous Uncertainty Sampling for Supervised Learning. In Proceedings of ICML 1994, New Brunswick, NJ, USA, 1994.
- Li 2001: Li, X., Roth, D. Exploring evidence for shallow parsing. In Proceedings CoNLL 2001, Toulouse, France, 2001. doi: <http://dx.doi.org/10.3115/1117822.1117826>
- LingPipe: Alias-i LingPipe, [www.alias-i.com/lingpipe](http://www.alias-i.com/lingpipe)
- Linnaeus 1735: Linnaeus, C. *Systema naturæ, sive regna tria naturæ systematice proposita per classes, ordines, genera, & species*. Lugduni Batavorum. Haak, Leiden. 1735.
- Linnaeus 1753: Linnaeus, C. *Species Plantarum*. Laurentius Salvius, Stockholm. 1753.
- Lintott 2008: Lintott, C. J., Schawinski, K., Slosar, A., Land, K., Bamford, S., Thomas, D., Raddick, M. J., Nichol, R. C., Szalay, A., Andreescu, D., Murray, P. and Vandenberg, J. Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey. *Monthly Notices of the Royal Astronomical Society*, 389, 2008. doi: 10.1111/j.1365-2966.2008.13689.x
- Littlestone 1988. Littlestone, N. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4): 285. 1988. ISSN 0885-6125, DOI 10.1007/BF00116827.
- Lu 2008: Lu, X., Kahle, B., Wang, J. Z., Giles, C. L. A metadata generation system for scanned scientific volumes. In Proceedings of JCDL 2008, Pittsburgh, PA, USA, 2008.
- Magerman 1995: Magerman, D. M. Statistical decision-tree models for parsing. In Proceedings ACL 1995, Morristown, NJ, USA, 1995.
- Marcus 1994: Marcus, M. P., Santorini, B., Marcinkiewicz, M. A. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, Vol. 19, No. 2, pp. 313-330, 1994.
- Marsh 1998: E. Marsh, D. Perzanowski. MUC-7 Evaluation of IE Technology: Overview of Results. In Proceedings of MUC-7, 1998.
- MEDLINE: National Library of Medicine: MEDLINE (factsheet: <http://www.nlm.nih.gov/pubs/factsheets/medline.html>)

- Michalski 1983: Michalski, R. S., Carbonell, J. G., Mitchell, T.M. Machine Learning: An Artificial Intelligence Approach. Tioga Publishing Company, 1983. ISBN 0-935382-05-4.
- Mikheev 1999: A. Mikheev, M. Moens, C. Grover, Named Entity Recognition without Gazetteers, in Proceedings of EACL, Bergen, Norway, 1999
- Miller 2000: D. Miller, S. Boisen, R. Schwartz, R. Stone, R. Weischedel, Named Entity Extraction from Noisy Input: Speech and OCR, in Proceedings of ANLP 2007, Seattle, WA, USA, 2000
- MODS: Metadata Object Description Schema. <http://www.loc.gov/standards/mods/>
- Mössenböck 1996: Mössenböck, H., Koskimies, K. Active Text for Structuring and Understanding Source Code. Software: Practice and Experience 26 (7), 833-850, 1996.
- Mülle 2006: Mülle, J. A., Böhm, K., Röper, N., Sünder, T. Building conference proceedings requires adaptable workflow and content management. In Proceedings of VLDB 2006, Seoul, Republic of Korea, 2006.
- Müller 2004: M. Müller, F. Padberg, An Empirical Study about the Feelgood Factor in Pair Programming, Int. Symp. on Softw. Metr. 10 (2004) 151–158
- Newby 2003: Newby, G. B., Franks, C. Distributed proofreading. In Proceedings of JCDL 2003. Houston, TX, USA, 2003. doi: 10.1109/JCDL.2003.1204888
- Newell 1959: Newell, A., Shaw, J. C., Simon, H. A. Report on a general problem solving program. In Proceedings of International Conference on Information Processing, UNESCO, Paris, France, 1959.
- Ngai 2000: Ngai, G., Yarowsky, D. Rule writing or annotation: cost-efficient resource usage for base noun phrase chunking. In Proceedings of CoNLL 2000, Hong Kong, China, 2000. doi: <http://dx.doi.org/10.3115/1075218.1075234>
- Niu 2003: Niu, c., Li, W., Ding, J., Srihari, R. K. A Bootstrapping Approach to Named Entity Classification Using Successive Learners, In Proceedings ACL, 2003
- Olson 2008: Olson, D. L., Delen, D. Advanced Data Mining Techniques. Springer, 2008. ISBN 3540769161
- O'Neill 1978: O'Neill, T. J. Normal discrimination with unclassified observations. Journal of the American Statistical Association, 73, 1978.
- OpenNLP: The OpenNLP project, [www.opennlp.org](http://www.opennlp.org)
- OWL: <http://www.w3.org/TR/owl2-overview/>
- Oxygen: <Oxygen>, [www.oxygenxml.com](http://www.oxygenxml.com)
- Petri 1962: Petri, C.A. Kommunikation mit Automaten. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.
- Plazi: [Plazi.org](http://plazi.org) | Taking Care of Freedom, <http://plazi.org>
- Protégé: The Protégé Ontology Editor and Knowledge Acquisition System, <http://protege.stanford.edu/>
- Rabiner 1986: L. Rabiner, B. Juang An Introduction to Hidden Markov Models, in IEEE ASSP Magazine, Jan 1986, Volume 3, Issue 1, pp 4-16
- RDF: Resource Description Framework (RDF) Model and Syntax Specification, <http://www.w3.org/TR/PR-rdf-syntax/>
- RelaxNG: Eric van der Vlist, <http://books.xmlschemata.org/relaxng/>
- Reynar 1997: Reynar, J. C., Ratnaparkhi, A. A maximum entropy approach to identifying sentence boundaries. In Proceedings ANLP 1997, Morristown, NJ, USA, 1997.
- Rosenblatt 1958. Rosenblatt, F. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Cornell Aeronautical Laboratory, Psychological Review (65), 386–408. 1958. doi:10.1037/h0042519
- Sautter 2006: G. Sautter, K. Böhm, The Difficulties of Taxonomic Name Extraction and a Solution, in proceedings of BioNLP, New York, NY, USA, 2006
- Sautter 2006a: Sautter, G., Böhm, K., Agosti, D. A combining approach to find all taxon names (FAT). In Biodiversity Informatics, Vol. 3, [Online: <https://journals.ku.edu/index.php/jbi/index>], 2006.

- Sautter 2007: Sautter, G., Agosti, D., Böhm, K. Semi-automated XML Markup of Biosystematics Legacy Literature with the GoldenGATE Editor. In Proceedings of PSB 2007, Weilea, HI, USA, 2007.
- Sautter 2007a: Sautter, G., Böhm, K., Padberg, F., Tichy, W. Empirical Evaluation of Semi-Automated XML Annotation of Text Documents with the GoldenGATE Editor. In Proceedings of ECDL 2007, Budapest, Hungary, 2007.
- Sautter 2009: Sautter, G., Agosti, D., Böhm, K., Klingenberg, C. Creating Digital Resources from Legacy Documents - an Experience Report from the Biosystematics Domain, in Proceedings of ESWC, Heraklion, Greece, 2009.
- Sautter 2010: Sautter, G., Böhm, K., Mathäß, M. ProcessTron: Efficient Semi-Automated Markup Generation for Scientific Documents. In Proceedings of JCDL 2010, Gold Coast, Australia, 2010.
- Sautter 2011: Sautter, G., Böhm, K. High-Throughput Crowdsourcing Mechanisms for Complex Tasks. In Proceedings of SocInfo 2011, Singapore, 2011.
- SchemaTron: The Schematron Assertion Language  
<http://xml.ascc.net/resource/schematron/Schematron2000.html>
- Shieber 1985: Shieber, S. M. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8: 333-343, 1985.
- Sidner 1981: Sidner, C.L. Focusing for interpretation of pronouns. *American Journal of Computational Linguistics*, 7(4), 1981.
- Siorpaes 2007: Siorpaes, K., Hepp, M. OntoGame: towards overcoming the incentive bottleneck in ontology building. In Proceedings OTM 2007, Vilamoura, Portugal, 2007.
- Snow 2008: Snow, R., O'Connor, B., Jurafsky, D., Ng, A. Y. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In EMNLP 2008, Morristown, NJ, USA, 2008.
- StanfordNLP: The Stanford Natural Language Processing Group,  
<http://nlp.stanford.edu/software/index.shtml>
- Tennison 2002: Tennison, J., Piez, W. LMNL: the Layered Markup and Annotation Language. In Proceedings of EMLC 2002. Montreal, Canada, 2002.
- Tichy 2000: W. Tichy, Hints for Reviewing Empirical Work in Software Engineering, *Journal of Empirical Softw. Eng.* 5 (2000) 309-312
- Tonkin 2008: Tonkin, E., Muller, H. L. Semi automated metadata extraction for preprints archives. In Proceedings of JCDL 2008, Pittsburgh, PA, USA, 2008.
- UltraEdit: IDM Computer Solutions Inc., [www.ultraedit.com](http://www.ultraedit.com)
- Vapnik 2000: Vapnik, V. N. *The Nature of Statistical Learning Theory* (2nd Ed.). Springer Verlag, 2000.
- Van Der Aalst 2003: Van der Aalst, W. M. et al. Workflow Patterns, Distributed and Parallel Databases 14(1): pp. 5-51, 2003
- Van Der Aalst 2004: Van der Aalst, W. M., van Hee, K. *Workflow Management: Models, Methods, and Systems*. The MIT Press, Cambridge, Massachusetts, 2004.
- Van Der Aalst 2005: Van der Aalst, W. M. P., ter Hofstede, A. H. M. YAWL: yet another workflow language. *Information Systems* 30 (4), 245 - 275, 2005.
- Von Ahn 2003: Von Ahn, L., Blum, M., Hopper, N., Langford, J. CAPTCHA: Using Hard AI Problems for Security. *Advances in Cryptology - EUROCRYPT 2003*. Springer Berlin / Heidelberg, 2003. doi: 10.1007/3-540-39200-9\_18
- Von Ahn 2006: Von Ahn, L., Games with a Purpose. *IEEE Computer*, 2006. 29(6): p. 92-94.
- Von Ahn 2008: Von Ahn, L., Maurer, B., McMillen, C., Abraham, D., Blum, M. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science* 321 (5895), 2008. doi:10.1126/science.1160379
- WEKA: Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, J. H. *The WEKA Data Mining Software: An Update*; SIGKDD Explorations, Volume 11, Issue 1. 2009.

White 2004: White, S. A. Business Process Modeling Notation, specification of BPMN v1.0, for Business Process Management Initiative (BPMI), 2004, [http://www.bpmn.org/Documents/BPMN V1-0 May 3 2004.pdf](http://www.bpmn.org/Documents/BPMN_V1-0_May_3_2004.pdf)

Wilson2003: E. Wilson, The encyclopedia of life, Trends Ecol. Evol. 18: 77-80, 2003

WordFreak: WordFreak, <http://wordfreak.sourceforge.net/>

XHTML: <http://www.w3.org/TR/xhtml1/#xhtml>

XML: <http://www.w3.org/TR/REC-xml>

XmlSchema: XML Schema <http://www.w3.org/XML/Schema>

XmlSpy: Altova GmbH, <http://www.altova.com>

XPath: XML Path Language. <http://www.w3.org/TR/xpath>

XQuery: W3C XML Query, <http://www.w3.org/XML/Query/>

XQueryUpdate: XQuery Update Facility 1.0, <http://www.w3.org/TR/xquery-update-10/>

XSLT: Extensible Stylesheet Language Transformations, <http://www.w3.org/TR/xslt20/>



## Appendix

### A. The GAMTA Data Model

The Generalized Annotation Model for Text Analysis, GAMTA for short, is the document data model used throughout the work described in this thesis. It amalgamates the various document representations discussed in Section 6.4.1, adding the ability to edit the document text in addition to the markup elements.

Following the Tag-Array data model, a GAMTA document primarily is an ordered sequence of **tokens**, not one of plain characters, though it can enact the latter as well. A token essentially is an atomic semantic unit of text, i.e. a word, a number, or a punctuation mark, which in addition can have attributes. The token orientation facilitates to bind the **annotations** that represent the markup elements to token indices instead of character offsets, which reduces the computational effort for character level updates to the document text. This is an important feature when dealing with digitized legacy documents, namely for correcting OCR errors.

Like the data model underlying GATE, the GAMTA data model purely consists of Java™ interfaces in order to allow for arbitrary implementations. The package also includes a default implementation to reduce effort in deployment; a centralized abstract factory for the individual data model components facilitates to easily replace the default implementation with another one. In detail, the following interfaces constitute the GMATA data model:

- A **Character Sequence** is the basis of all further elements, representing a document as a sequence of plain characters. It has a length that indicates the number of characters it comprises, and it provides access to each individual one of them.
- A **Mutable Char Sequence** is a Character Sequence that supports edit operations, i.e., replacing, inserting, or removing sub sequences of characters.
- A **Char Span** marks a snippet of a Character Sequence, anchored to a start offset and an end offset, i.e., the offset of the character it starts at in the underlying Char Sequence and the offset of the character before which it ends; as a part of a Character Sequence, a Char Span also is a Character Sequence in itself.
- An **Attributed** object can have attributes, which are essentially pairs of an attribute name and value. Such an object also provides the functionality to read and manipulate its attributes.
- A **Token** is an Attributed Char Span that marks an atomic unit of semantic meaning in a document text, i.e. a word, a number, or a punctuation mark, or a sequence of identical punctuation marks.
- A **Token Sequence** is a tokenized Character Sequence that makes its individual tokens accessible by their index; it has a size that indicates the number of tokens it comprises.

- A **Mutable Token Sequence** incorporates the properties of a Token Sequence and a Mutable Char Sequence; namely, it supports edit operations on both character and token level.
- An **Annotation** is an Attributed marker for a part of a Token Sequence, anchored to a start index and an end index, i.e., the index of the token it starts at in the underlying Token Sequence and the index of the token before which it ends; as a part of a Token Sequence, an Annotation also is a Token Sequence in itself. By this definition, there can be Annotations that mark parts of other Annotations; the latter are referred to as nested in the Annotation they mark a part of. Further, an Annotation provides read access to document global properties, e.g. a specific character encoding to use for storage.
- A **Queryable Annotation** is an Annotation that allows retrieving its nested Annotations; among others, this facilitates the evaluation of XPath queries against them, thus the name.
- A **Mutable Annotation** integrates the features of a Queryable Annotation and a Mutable Token Sequence and in addition supports adding and removing nested Annotations. As such, it essentially is a (sub-) document in which both text and markup are editable.
- A **Document Root** is the Mutable Annotation that represents a given document as a whole and thus is the root of a hierarchy of Annotations; it allows for manipulating document global properties.

To simplify working with GAMTA documents, this data model also includes several function libraries that provide frequently used functions; namely, there are dedicated libraries for Attributed objects, Character Sequences, Token Sequences, and Annotations. Further, the GAMTA package includes an XPath implementation for executing respective queries on Queryable Annotations. Also included are wrappers that enable NLP components implemented on top of one of the data models presented in Section 6.4.1 to work with documents in the GAMTA representation. Finally, the package provides a slim interface for NLP components that work on GAMTA documents, be it by wrapping third-party NLP components or by implementing native ones. Connected to this interface are generic Java class loading facilities, which are used both for the NLP components and throughout the rest of the GoldenGATE system, see below.

In addition to the GAMTA core package, there are several extensions: The **GAMTA Imaging API** provides the functionality for extracting individual page images from PDF documents, which are often the pre-OCR result of scanning, and for manipulating and analyzing these images and the structure of the pages they were taken from. The **GAMTA Feedback API** readily provides implementation of the specific visualizations for correcting the results of the four classes of NLP tasks (see Section 8.3.3). The respective document views can be rendered both in Java Swing and in HTML and JavaScript; this dualism facilitates to make selected corrections browser based, a feature extensively used in the implementation of the crowdsourcing scenario. This package also includes the infrastructure components that handle individual correction dialogs between the NLP components and the answering users, and it implements the data quality enforcement mechanisms from Chapter 10. Finally, the **GAMTA to GATE Wrapper** emulates the GATE data model based on a GAMTA document; this facilitates the integration of GATE-based NLP components.

## B. The GoldenGATE Editor

The GoldenGATE Editor is the platform that implements the integration of assisted XML editing and deployment of NLP tool described in this thesis upon the GAMTA data model. The following section describes its basic architecture.

The core of the GoldenGATE Editor is a plug-in host that brings together the individual groups of components involved, and a respective loading mechanism for these plug-ins. Special components that provide the plug-ins are so-called **Configurations** that can load them from multiple sources; configurations are also responsible both for which plug-ins are available, and for where to load them from; this is to facilitate the GoldenGATE Editor to be available in many different setups.

The basis of the plug-in mechanism in the GoldenGATE Editor is the Java interface **Golden Gate Plug-in**. This interface includes the lifecycle management functions for the individual plug-ins, i.e., functions for initialization and shutdown. It further includes specific mounting points for the plug-ins to integrate themselves in the GoldenGATE Editor user interface, i.e., getter functions from which the editor core retrieves menu items that make the plug-ins accessible in the various menus. The editor core provides a central registry through which individual plug-ins can access each other. All other types of plug-ins used in semantic markup generation derive from this interface.

**Document IOs** are plug-ins that access different storage locations for documents, be it files on the local system or network, URLs, databases, or a GoldenGATE Server (see Appendix D). Closely related to Document IO plug-ins are **Document Format Providers**, which are responsible for decoding different document formats into the unified internal representation as GAMTA documents. Document Format Plug-ins exist for HTML, various XML schemas, and plain text, and there is one for handling different character encodings, for use in combination with one of the others.

**Document Editor Extensions** are plug-ins that add special visualization functionality to the main document display, e.g. a viewer for page images. Related to Document Editor Extensions are **Document Viewer** plug-ins, which open specialized document views in sub dialogs of the main window; the List View and the Slide View (see Section 7.4.2) fall into this category.

**Resource Managers** are plug-ins that provide the functionality for NLP; they handle individual NLP tools as **Resources**; one resource manager can provide multiple resources of the same type. There are three special types of resources, and respective managers:

- **Annotation Filter Providers** are responsible for handling **Annotation Filters**, e.g. preconfigured XPath queries.
- **Annotation Source Providers** make available so-called **Annotation Sources**, i.e., basic means of extracting specific phrases from documents; most prominent here are gazetteer lists and regular expression patterns.
- **Document Processor Managers** are responsible for handling **Document Processors**, i.e., small components that manipulate both text and markup elements of a document; the Analyzer Manager and the Pipeline Manager (see Section 7.4.2), for instance, and also the ProcessTron plug-in used in Section 9.2 fall into this category, and there are several others.

The **Custom Function Manager** is a Resource Managers that is an integral part of the GoldenGATE Editor core. Its responsibility is to provide the special buttons for the user interface that make arbitrary Document Processors one-click accessible, thus saving users going through menus, which is especially helpful for frequently used functionality. Likewise integrated in the core is the **Custom Shortcut Manager**, a Resource Managers providing custom-configured shortcuts that help with manually creating individual markup elements; namely, they react to key strokes and generate a markup element of a configured type, optionally running this markup element through a configured Document Processor afterward.

The **graphical user interface** (GUI) of the GoldenGATE Editor is also exchangeable, and the editor core with all its plug-ins can exist and work without having a GUI on top of them at all. It is the responsibility of the **startup program** to set the GUI, which facilitates using the GoldenGATE Editor core in a variety of applications and scenarios, with respectively appropriate GUIs on top. The second responsibility of the startup program is to specify the configuration for the editor core to load the plug-ins from. There are startup programs for using the GoldenGATE Editor as a desktop application and as an Applet in a browser, respectively.

### C. GoldenGATE Evaluation Questionnaire

Table C.1 (on the right) lists the questions asked to the participating users in the laboratory experiment described in Section 7.5. Group A used XML Spy and then the GoldenGATE Editor, for Group B the order was reverse. The users answered Questions 1 – 19 by checking off one out of four boxes, the fourth / rightmost box representing the most positive answer. For Question 20, there were three boxes, the first / leftmost representing the most positive answer. For Question 22, finally, there were six boxes, the first / leftmost again representing the most positive answer.

Question 21 is not listed in Table C.1 because it referred to the length of the documents used in the experimental tasks, not to the tools they had worked with; on average, the users judged the length of the documents as ‘appropriate’ to ‘a little too long’.

**Table C.1: Users’ detail assessment of the GoldenGATE Editor**

Participant		Group A					Group B					Overall Avg	
		2	4	6	8	9	Avg	1	3	5	7		Avg
Question (Best Score - Worst Score)													
1	How easy is the user interface to oversee? (4-1)	2	3	3	2	2	2,4	3	2	3	3	2,75	<b>2,56</b>
2	How easy are the essential functions to access? (4-1)	3	2	3	2	3	2,6	4	3	3	3	3,25	<b>2,89</b>
3	How easy is it to create annotations? (4-1)	4	3	4	3	4	3,6	4	4	4	4	4,00	<b>3,78</b>
4	How easy is it to edit the document text? (4-1)	1	1	1	2	2	1,4	2	2	1	2	1,75	<b>1,56</b>
5	How easy is it to edit or remove annotations? (4-1)	4	4	3	4	4	3,8	4	4	4	4	4,00	<b>3,89</b>
6	How natural is it to move and edit annotations instead of removing old ones and creating new ones? (4-1)	1	2	1	3	3	2	3	0	3	4	3,33	<b>2,50</b>
7	How helpful is it to create XML syntax automatically? (4-1)	4	3	4	4	4	3,8	4	4	4	4	4,00	<b>3,89</b>
8	How sensible is it to annotate on word-level as opposed to character-level? (4-1)	2	3	4	4	4	3,4	4	4	4	3	3,75	<b>3,56</b>
9	How well is the support for correcting OCR errors? (4-1)	1	1	1	4	1	1,6	3	2	4	0	3,00	<b>2,13</b>
10	How helpful is automated paragraph flow text conversion? (4-1)	4	3	4	4	2	3,4	4	4	4	4	4,00	<b>3,67</b>
11	How helpful is merging and splitting of annotations? (4-1)	4	4	2	3	4	3,4	4	3	4	4	3,75	<b>3,56</b>
12	How helpful is the page-by-page view? (4-1)	3	2	3	2	4	2,8		3	4	4	3,67	<b>3,13</b>
13	How helpful is showing / hiding annotations in reading the document? (4-1)	4	4	0	4	4	4	4	4	4	4	4,00	<b>4,00</b>
14	How helpful is the availability of both semantic coloring and XML tags and the option to switch between them? (4-1)	4	4	4	4	4	4	4	4	4	4	4,00	<b>4,00</b>
15	How helpful is global annotating? (4-1)	4	4	4	4	4	4	4	4	4	4	4,00	<b>4,00</b>
16	How easy is it to use the integrated NLP tools? (4-1)	4	2	3	3	4	3,2	3	4	4	4	3,75	<b>3,44</b>
17	How helpful is the tabular view for correcting detail-level annotations? (4-1)	3	0	0	2	4	3	0	4	4	0	4,00	<b>3,40</b>
18	How well did the editor support you in the overall completion of your task? (4-1)	3	3	3	4	3	3,2	3	4	4	3	3,50	<b>3,33</b>
19	How do you perceive the performance of the editor? (4-1)	2	2	3	4	3	2,8	3	3	3	3	3,00	<b>2,89</b>
20	How many problems did you have in using the editor, and which? (1-3)	2	2	3	2	2	2,2	2	2	1	2	1,75	<b>2,00</b>
22	How do you grade the editor overall, on a 1-6 scale?	4	2	2	2	2	2,4	2	2	2	2	2,00	<b>2,22</b>

## D. The GoldenGATE Server System

The GoldenGATE Server is an application server that forms the back-end of the Facebook Application (see Section 10.6), among others; it is deployed in other roles as well, e.g. as the storage facility and retrieval system for the documents from the Madagascar Corpus (see Section 8.1) and the ZooTaxa Corpus (see Section 9.2.3). Like the GoldenGATE Editor core, it is primarily a host for plug-ins that implement the actual functionality. It provides a console interface for administration, and a network interface through which it makes available the functionality of its plug-ins. Further, it includes a centralized event notification system through which individual plug-ins can monitor each other's activities. Finally, the server core provides a centralized access point to the underlying database for the plug-ins to use; this simplifies configuration because the database access parameters are stored in a dedicated central location, not in the individual plug-ins.

The basis of the plug-in mechanism in the GoldenGATE Server is the Java interface **Golden Gate Server Component**. This interface includes the lifecycle management functions for the individual plug-ins, i.e., functions for initialization, linking, and shutdown. It further includes the getter function from which the server core retrieves the actions that make the plug-in accessible through the network and console interfaces. The server core provides a central registry through which individual plug-ins can connect to each other on startup, namely in the linking phase. There are several plug-ins worth mentioning here because they are deployed in the back-end of the Facebook Application or the GoldenGATE Server installations that host the documents of the Madagascar Corpus and the ZooTaxa Corpus:

- The **User Access Authority** (UAA) handles user accounts and permission, and it manages user sessions; it has a mounting interface for advanced permission managers, through which the **User Permission Authority** (UPA) integrates the role based access control it implements.
- The **Document IO Service** (DIO) manages the centralized document store and makes it accessible locally inside the server and over the network interface. The access functions implement a 'global read / authenticated write' policy, backed by the user access authority for authentication.
- The **Document Image Service** (DIS) provides page images for the document stored in the DIO, backed by the GAMTA Imaging API. It observes the addition of new document to DIO by means of the central event notification system, and then obtains the respective page images from and makes them available.
- The **Editor Configuration Service** (ECS) manages and provides different configurations for the GoldenGATE Editor, both locally inside the server and over the network and Internet.
- The **Remote Feedback Service** (RFS) ships correction tasks from Document Processors running inside the server to users over the network, and the results back to their source; it builds upon the GAMTA Feedback API.
- The **Document Processing Service** (DPS) is the source of the correction tasks in the Facebook Application. Backed by a GUI-less GoldenGATE Editor based on a configuration provided by the ECS, controlled by ProcessTron, and obtaining user corrections via the RFS, the DPS implements server based semantic

markup generation. In particular, it observes the addition of new document to the DIO-managed collection and runs these documents through its markup process.

- The **Remote Event Service** (RES) observes the central event notification service and makes the events available over the network. In addition, it periodically fetches events from remote installations of GoldenGATE Server and issues respective notifications locally. This facilitates, for instance, the replication of document updates that occurred on a remote server to the local DIO's document collection, a functionality implemented in the **Document Replication Service** (DRS).
- The **User Scoring Service** (USS) is the basis of the reward system in the Facebook Application; it observes the RFS for completed correction tasks and adds a respective score to the accounts of the users who answered the task.
- The **User Competence Service** (UCS) also observes the RFS for completed correction tasks and computes the errors of the individual answering users. This forms the basis for the deployment of Vote Boosting (see Section 10.4).
- The **Search and Retrieval Service** (SRS) indexes documents and makes them available for retrieval. The indexed documents can either be entire documents from the local DIO, or parts of such documents, e.g. individual taxonomic treatments. The indices are maintained by respective **Indexer** plug-ins; except for the full text index, they are based on the semantic markup contained in the documents. Both the documents and the content of the individual indices are accessible through the network interface.

The **GoldenGATE Server Web Front-End** is a collection of Java Servlets that integrate the functionality of a backing GoldenGATE Server and its plug-ins into a web site. In addition, a dedicated **Proxy Servlet** provides an HTTP tunnel for accessing the network interface of the backing server over the Internet. There are several Servlets and groups of Servlets worth mentioning here because they are deployed in the web front-end of the Facebook application or the GoldenGATE Server installations that host the documents of the Madagascar Corpus and the ZooTaxa Corpus:

- The **SRS Search Portal** consists of several Servlets. One offers a semantic search portal based on a backing SRS and its Indexer plug-ins, for instance the treatment search portal of Plazi; the search portal provides both the indexed documents and the content of individual indices. Another one makes the indexed documents available in a variety of XML schemas. A third one generates RSS feeds and a Sitemap, so harvesters can receive a notification of updates and ingest the newly added documents.
- The **Page Image Servlet** provides images of document pages from a backing DIS, also building on the GAMTA Imaging API.
- The **Feedback Servlet** handles crowdsourced correction tasks to users in the form of HTML forms; its embedded feedback engine is the host for the various data quality enforcement mechanisms deployed in the Facebook Application. Related Servlets display user rankings and other community functionality.
- The **Authenticated Web Front-end** forms a common host for parts of the web site that require authentication to access, for instance administrative functions. The individual parts are integrated via plug-in **Modules**, for instance for user management or role and permission management.