

Technische Dokumentation COSA Webapplikation

zuletzt bearbeitet am 23. Juli 2009

erstellt durch

Bruno Wüest wueest@ipz.uzh.ch

Stefani Gerber stefani.gerber@zda.uzh.ch

Inhaltsverzeichnis

1	Anforderungen	3
1	Allgemeines	3
2	Adminbereich	4
2.1	Administrieren von Kampagnen	4
2.2	Verwaltung von Zeitungsartikeln und Akteur- und Themenlisten	4
2.3	On-Time-Update Akteur- und Themenlisten	4
3	Codierbereich (GUI)	5
3.1	Metadata	5
3.1.1	Artikeldaten	5
3.1.2	Kampagnendaten	5
3.2	Navigation	6
3.2.1	Sentence Navigation	6
3.2.2	Observation Navigation	6
3.3	Textfeld	7
3.4	Annotation	7
4	Zieldatensatz	9
2	Eingabefiles	11
1	XML	11
2	precosa	12
3	cosa	14
3	GUI Navigation	16
1	Anzeige	16
1.1	xml	16
1.2	precosa	17
1.3	cosa	17
2	Vorselektion	17
2.1	xml	17
3	Abspeichern	17

4	Technische Umsetzung	19
1	Datenbank	19
2	Model	19
3	Tests	19

Kapitel 1

Anforderungen

1 Allgemeines

Die Annotationsanwendung von COSA soll mit einer Webserver-Client-Lösung die simultane Erhebung von Daten zu elektronischen Zeitungsartikeln ermöglichen. Bis zu 10 codierende Personen sollten gleichzeitig über den Browser annotieren können. Folgende Funktionalitäten muss die Applikation leisten:

1. Administrieren von Kampagnen (einzelne Annotationspensen)
2. Verwaltung von Zeitungsartikeln
3. On-Time-Update Akteur- und Themenlisten
4. GUI mit Highlighting und dynamischen Listen
5. Abspeichern und Export des Zieldatensatzes

Benutzerrechte:

1. Admin: Administrieren von Kampagnen
2. User: Annotieren (Alles ausser Kampagnenadmin)

Zu verarbeitende Artikel:

1. *xml*-Files (ohne Texttags ausser `<p>` und `<title>`, ermöglichen weder Satznavigation noch Listenupdates).
2. *precosa.xml*-Files (ermöglichen nur Satznavigation).
3. *cosa.xml*-xml-Files (mit entities für Listen).

Bemerkungen: Bitte sparsamste, direkteste Lösungen in Betracht ziehen. Z.B. nicht zu stark auf Grafisches achten. Schnelligkeit ist entscheidend!

2 Adminbereich

2.1 Administrieren von Kampagnen

- Separates Login
- Bestimmung von:
 - Kampagne (Name und Code), z.B. Name: “uk_2005”, identifier: “603”
 - Codierende (Name, Code, Einloggen vorbereiten)
 - Zusätzliche Variablen (noch beiseite lassen für den Moment)
 - Artikel (Auswahl/Import, Ablage in Datenbank, Export, Löschen/Überschreiben)
 - Listen (Auswahl/Import, Ablage in Datenbank, Export, Löschen/Überschreiben)
 - Bestimmung, welche Artikel für welchen Codierenden (evtl. Tagen oder DB-Eintrag).
 - Export endgültiger Datensatz.

2.2 Verwaltung von Zeitungsartikeln und Akteur- und Themenlisten

- Daten für Listenergänzungen extrahieren (evtl. bereits beim Import).
- Nach erfolgter Codierung als *annotiert* kennzeichnen.

2.3 On-Time-Update Akteur- und Themenlisten

- Listen für Akteur- und Themeneingabefelder mit erkannten Entities (vorcodierte Wörter) ergänzen, wenn ein Satz aktiviert wird (An den Anfang der Liste setzen).
- Wenn Artikel gespeichert wird, Listen vom Client mit Listen auf Server abgleichen und gegebenenfalls beide Listen aktualisieren (dies evtl. auch bei der Eingabe eines neuen Akteurs bzw. Themas).

3 Codierbereich (GUI)

Vier Frames, mit einfachen Reglern (flexible Frame-Kästen) einrichtbar:

3.1 Metadata (oben links)

3.2 Navigation (mitte)

3.3 Text (oben rechts)

3.4 Annotation (unten)

Alles über Tabs (Eingabefelder) und Shortcuts (Buttons) zugänglich machen!

3.1 Metadata

Besteht aus zwei Blocks:

3.1.1 Artikeldaten

ID, Country, Paper, Published, Section (rubric), Page, Length, Author, Country (countrycode).

Im Ausgangs-XML wie folgt getagt:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<article id="Nzz_2004_01_03_37945">
  <meta>
    <paper>Nzz</paper>
    <published>2004-01-03</published>
    <rubric>AUSLAND</rubric>
    <page>1</page>
    <length>1160</length>
    <author>-</author>
    <countrycode>CH</countrycode>
  </meta>
  ... hier Textblock ...
</article>
```

Code 1: Beispiel XML Meta Data

3.1.2 Kampagnendaten

Name Coder, Campaign, Date (of coding), Time (of coding), Observation (Nummer)

→ Nur Anzeige-, keine Eingabefenster, Observation, Date und Time bei jeder Kernsatzeingabe aktualisieren.

3.2 Navigation

8 Paare und ein einzelner an normalen Buttons:

- saveArticle
- getNextArticle (nächster nicht codierten Artikel laden)
- nextSentence und previousSentence (braucht beim ersten bzw. letzten Satz des Artikels evtl. eine Beschränkung)
- nextObservation previousObservation (braucht bei der ersten bzw. letzten Observation der Session eines Codierenden evtl. eine Beschränkung)
- nextActor und previousActor (Von einer erkannten Entität zur nächsten wechseln)

→ alle Buttons auch über Shortcuts bedienbar machen.

3.2.1 Sentence Navigation

- innerhalb des aktuellen Artikels
- beeinflusst highlighting im Preview
- kann die Annotations-Felder beeinflussen (je nach Artikeltyp)
- standardmässig ist der erste Satz markiert
- um die Performance zu verbessern werden die Sätze in der Session gespeichert.

3.2.2 Observation Navigation

- artikelübergreifend
- beeinflusst die Annotations-Felder
- standardmässig ist keine Observation ausgewählt, *previousObservation* geht zur letzten gespeicherten Observation zurück.
- aktuelle Observation wird NICHT implizit abgespeichert
- um die Performance zu verbessern werden die Observationen in der Session gespeichert.

3.3 Textfeld

1. Anzeige des Textes innerhalb des tags *annotation* der xml-Repräsentation des Artikels.
2. Ausgewählter Satz mit feinem Hintergrund (am Anfang der erste Satz) und in der Mitte des Scrollfenster.
3. Alle Entitäten auf mit feinem Hintergrund, farbliche Unterscheidung von Akteuren (rot) und Themen (blau). Diese Kennzeichnung basiert immer auf der Vorerkennung im XML, auch wenn bereits codierte Sätze für diesen Artikel vorhanden sind.

Logik der *id*-Attribute:

- Artikel (<article id="..."): Zeitung + Datum + Artikelnummer (alleine nicht eindeutig! Nur pro Zeitung und Datum...) z.B. thesun_2005_03_05_77
- Satz (<sentence id="..."): Artikel-ID + Satznummer (innerhalb des Artikels), z.B. thesun_2005_03_05_77-1 für den ersten Satz im Artikel.
- Token (einzelne Worte, <token id="..."): Satz-ID + Tokennummer, z.B. thesun_2005_03_05_77-1-1 für das erste Token im ersten Satz des Artikels.
- Entities fürs Highlighting (<entities><topic/actor ... listid="..." ...><tokenref ref="..."): topic-tac= es ist ein Thema, actor-tag=es ist ein Akteur, listid=Code in der Akteur- bzw. Themenliste, tokenref ref=auf dieses Wort im Text bezogen.

Xml-Kontext-Beispiel (siehe Code 2)

Das Sentence-Attribut type="..." zeigt den Satztyp an: Titel=title, Anfang eines Paragraphen=parastart, sonstige Sätze=normal. Restliche Attribute für Anzeige irrelevant.

3.4 Annotation

Ein Frame mit zwei Ansichten:

- Tabelleneingabe: Rollrahmen mit allen Felder einer Observation auf einer Zeile, dafür alle Observationen des Artikels anzeigen ("Excel-Style").
- strukturierte Eingabe: Felder sind logisch organisiert, dafür nur immer eine Observation anzeigen.

```
<article>
  <meta>
    ...
  </meta>
  <annotation>
    <sentence id="thesun_2005_03_05_77-11" type="parastart">
      <text>
        <token id="thesun_2005_03_05_77-11-1" lemma="Hut" type="normal">
          HuT
        </token >
      </text>
    </sentence>
    <entities>
      <topic id="top-5-5005056_16" listid="top-5-5005056">
        <tokenref ref="thesun_2005_03_05_77-2-16"/>
      </topic>
    </entities>
  </annotation>
</article>
```

Code 2: Beispiel XML Inhalt relevant für Annotation

4 Zieldatensatz

Nachfolgende Variablen müssen in einem zentralen Datensatz abgelegt und bereit zum Export gemacht werden. Organisationskriterium ist die Observation. Das heisst, für jeden Kernsatz muss eine Zeile mit allen Variablen verfügbar sein. Je nachdem kommen noch die zusätzlichen Variablen dazu. Für eine ausführlichere Beschreibung siehe Zieldatensatz.xls

- Campaign
- Coder
- Country
- Paper
- Observation
- articleID
- Relevant
- Published
- Rubric
- Length
- Page
- SentenceText
- SentenceID
- CoreType
- CoreTypeCode
- Quote1String, Quote1Code
- Quote2String, Quote2Code
- Subject1String, Subject1Code
- Subject2String, Subject2Code
- Quality

- Object1String, Object1Code
- Object2String, Object2Code
- Issue1String, Issue1Code
- Issue2String, Issue2Code
- Problem
- Date
- Time
- CoreCount
- Comment

Kapitel 2

Eingabefiles

Es gibt drei Typen von Eingabefiles. Sie unterscheiden sich im Ausmass der Vorverarbeitung. Dies beeinflusst, welche Einstellungen für den Codierer bereits übernommen werden können.

Gemeinsam ist allen drei Datentypen folgendes Gerüst:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<article id="thesun_2005_03_05_77">
  <meta>
    <paper>SU</paper>
    <published>2005-03-05</published>
    <rubric>-</rubric>
    <page>-</page>
    <length>165</length>
    <author>Neil Syson</author>
    <countrycode>2</countrycode>
  </meta>
  ...
</article>
```

Code 3: Metainformation, gleich bei allen drei Datentypen

1 XML

Dies ist "einfachste" Datenform. Der Artikel steht im Tag <text>, welches Überschriften (<title>) und Absätze (<p>) enthält. Beim Abspeichern als Sentence-Objekt werden folgende Eigenschaften gesetzt:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<article id="thesun_2005_03_05_77">
  <meta>
  ...
  </meta>
  <text>
    <title>UNIFORM-ROW MOB 'LIKE BNP'</title>
    <p>A RADICAL Muslim group which backed a girl's fight to wear
      head-to-toe dress at school has been branded as extreme
      as the BNP.</p>
    <p>Shabina Begum, 16, was supported by Hizb ut-Tahrir when
      she won a court victory to don a jilbab. </p>
    ...
  </text>
</article>
```

Code 4: Example XML file content

- sentence_id wird leer gelassen
- is_paragraph wird auf *true* gesetzt
- als Text wird alles zwischen *<p>* und *</p>* abgespeichert
- sentence_type entspricht dem Tag-Name (*title* oder *p*)

2 precosa

Der Artikel ist hier getagged nach Sätzen und Wörtern und beinhaltet den Wortstamm. Dies benötigt Software, die im Moment nur auf Deutsch und Englisch ausgerichtet ist. Die Struktur des Files ändert sich gegenüber dem plain xml, dass das *<text>* neu innerhalb von *<annotation>* und *<sentence>* liegt. Die *<syntax>*-Tags können samt Inhalt ignoriert werden.

Beim Abspeichern als Sentence-Objekt werden folgende Eigenschaften gesetzt:

- sentence_id wird auf den letzten Teil vom Attribut *id* des Tags *<sentence>* gesetzt. Die ID wird nach dem Zeichen “-” aufgespaltet. Beispiel: *<sentence id="thesun_2005_03_05_77-1" >*, sentence_id=1.
- sentence_type wird auf den Wert des Attributs *type* vom *<sentence>*-Tag gesetzt (*title*, *normal*, *parastart*).

```
<?xml version="1.0" encoding="utf-8"?>
<article id="thesun_2005_03_05_77">
  <meta>
    ...
  </meta>
  <annotation>
    <sentence id="thesun_2005_03_05_77-1" type="title">
      <text>
        <token id="thesun_2005_03_05_77-1-1" lemma="Uniform-row"
          type="normal">
          UNIFORM-ROW
        </token>
        <token id="thesun_2005_03_05_77-1-2" lemma="mob" type="normal">
          MOB
        </token>
        ...
      </text>
      <syntax>
        <dependency label="nchunk" governor="thesun_2005_03_05_77-1-4"
          dependant="thesun_2005_03_05_77-1-2"/>
      </syntax>
    </sentence>
    <sentence id="thesun_2005_03_05_77-2" type="parastart">
      <text>
        <token id="thesun_2005_03_05_77-2-1" lemma="a" type="normal">
          A
        </token>
        <token id="thesun_2005_03_05_77-2-2" lemma="radical" type="normal">
          RADICAL
        </token>
      </text>
      <entities>
        <topic id="top-5-5005056_16" listid="top-5-5005056">
          <tokenref ref="thesun_2005_03_05_77-2-16"/>
        </topic>
      </entities>
    </sentence>
  </annotation>
</article>
```

Code 5: Example precosa file content

- `is_paragraph` wird auf *false* gesetzt
- als Text werden die Inhalte der `<token>` innerhalb des `<sentence><text>`-Tags zusammengehängt.

3 cosa

Dieser Filetyp hat alle Eigenschaften von `precosa` und er ist zusätzlich schon vorcodiert. Dies ermöglicht die Vorauswahl der erkannten Akteure, bedingt aber auch, dass dieselben Listen die zur Vorcodierung verwendet wurden bereits im System importiert wurden.

```
<?xml version="1.0" encoding="utf-8"?>
<article id="thesun_2005_03_05_77">
  <meta>
    ...
  </meta>
  <annotation>
    ...
  </annotation>
  <entities>
    <actor id="act-6009-0_4" listid="act-6009-0">
      <tokenref ref="thesun_2005_03_05_77-6-4"/>
    </actor>
    <topic id="top-5-5000506_25" listid="top-5-5000506">
      <tokenref ref="thesun_2005_03_05_77-6-25"/>
    </topic>
    <topic id="top-6-5006061_32" listid="top-6-5006061">
      <tokenref ref="thesun_2005_03_05_77-6-32"/>
    </topic>
  </entities>
  <cores>
    <at_core>
      <subjectRef ref="act-6009-0_4"/>
      <topicRef ref="top-5-5000506_25"/>
      <predicate>0</predicate>
    </at_core>
    <at_core>
      <subjectRef ref="act-6009-0_4"/>
      <topicRef ref="top-5-5000506_25"/>
      <predicate>0</predicate>
    </at_core>
  </cores>
</article>
```

Code 6: Example cosa file content

Kapitel 3

GUI Navigation

Die Codierschnittstelle muss so entwickelt sein, damit das codieren wo möglich erleichtert wird. Sie lässt sich in die Anzeige des Artikels und in die Vorselektion von Einstellungen unterteilen.

1 Anzeige

Statisch werden Informationen über die aktuelle Kampagne und Meta-Informationen des aktuellen Artikels angezeigt. Das Kernstück ist aber die Anzeige des Artikeltexts und eine Hervorhebung des aktuell bearbeiteten Satzes oder Abschnitts und allenfalls vorerkannter Akteure oder Themen. Was markiert werden kann hängt allerdings stark vom Format ab, in dem der Artikel vorliegt.

Textdarstellung:

- *normal* fortlaufend
- *title* fett
- *parastart* hat zwei Zeilenumbrüche vorangestellt

1.1 xml

Hier ist keine Satznavigation möglich sondern nur eine Navigation auf Paragraphenebene. Es wird also immer der aktuelle Paragraph markiert (und mit der Observation abgespeichert).

1.2 precosa

Markiert wird der aktuelle Satz.

1.3 cosa

Markiert werden zusätzlich zum aktuellen Satz die erkannten Entitäten (=Akteure) und Themen.

2 Vorselektion

Die vorhandenen Informationen sollen wo möglich in die Dropdowns des Annotierbereichs geladen werden.

2.1 xml

nicht möglich

3 Abspeichern

Observation

- Artikel
- Objekt
- Subjekt
- Quote
- Thema
- Typ
- Qualität
- Coder Kommentar
- Problematisch
- Gelöscht

Satz (bei xml der Paragraph)

- Text
- Satz-ID (falls vorhanden)
- Type (title,normal, parastart)

Kapitel 4

Technische Umsetzung

Die Webapplikation wurde mithilfe des PHP-Frameworks Prado¹ umgesetzt. Als Datenbank wurde MySQL² verwendet.

1 Datenbank

Die Applikation basiert auf dem Datenbankschema aus Abbildung 4.1.

2 Model

Das Model (vgl. Abbildung 4.2) basiert zu grossen Teilen auf dem Datenbankschema, hat aber noch einige zusätzliche, nicht-persistente Objekte. Letztere sind hauptsächlich im Codierbereich nötig, weil sich persistente Objekte schlecht in der Session speichern lassen und diese Objekte bei jedem Laden der Seite neu aus der Datenbank zu holen würde die Performance zu stark beeinträchtigen.

Das Framework (PRADO) arbeitet mit dem ActiveRecordPattern. Da die Funktionalität der Objekte komplex ist, wurden zusätzliche Objekte erstellt, die den ActiveRecord verwalten und die zusätzlichen Funktionalitäten modellieren. Abbildung 4.2 zeigt letztere Objekte, nicht die ActiveRecord.

3 Tests

Diese Applikation hat ausführliche Unit Tests, auf funktionale Tests wurde verzichtet.

¹<http://pradosoft.com/> Version 3.1.3

²<http://www.mysql.com> Version 5.0.67

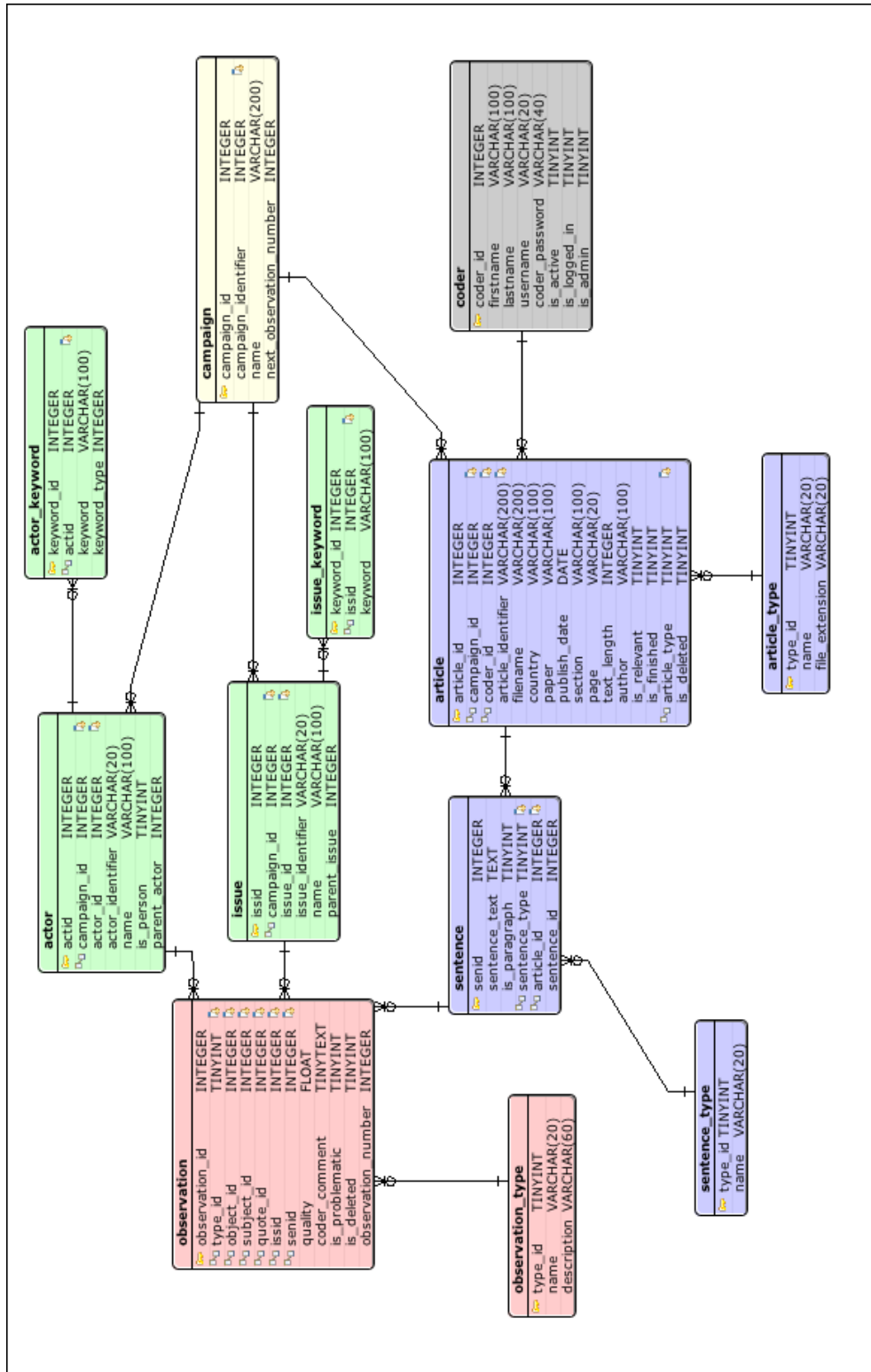


Abbildung 4.1: Datenbankschema

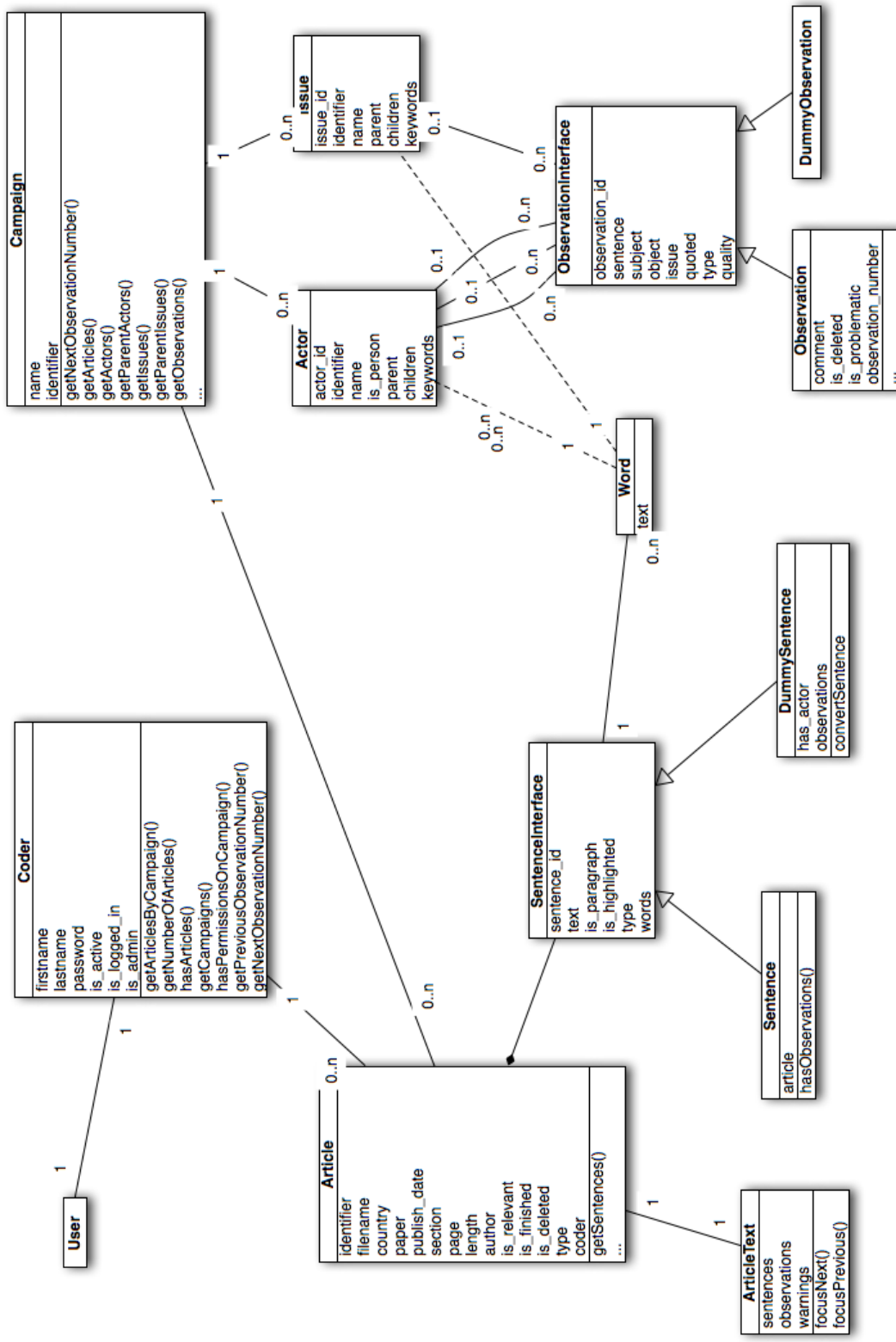


Abbildung 4.2: Model

Die Unit Tests befinden sich im Verzeichnis *tests/unit*. Sie können in einem Webbrowser ausgeführt werden. Dazu muss das Verzeichnis *tests* von einem Webserver bedient werden. Das Skript *unit.php* listet alle Testfälle auf, die durch Klicken auf den jeweiligen Namen ausgeführt werden.

Damit die Tests ausgeführt werden können benötigen sie eine “saubere” Datenbank³: diese beinhaltet die Struktur und einzelne Datensätze (hauptsächlich in Systemobjekten). Die Datenbankstruktur kann mit der Datei *docs/database/create.sql* erzeugt und mit der Datei *docs/database/insert.sql* gefüllt werden. Falls ein Test fehlschlägt, werden unter Umständen nicht alle Einträge wieder aus der Datenbank gelöscht. Vor einem erneuten Aufruf der Tests muss deshalb die Datenbank zurückgesetzt werden, indem man die Dateien *docs/database/truncate.sql* und *docs/database/insert.sql* ausführt. Einige Tests müssen eine Datei erzeugen können, deshalb muss der Ordner *tests/server* für den Webserver schreibbar sein.

³Es empfiehlt sich, eine separate Datenbank für die Tests zu erstellen