Projektvertrag

Inhaltsverzeichnis

1 Allgemein

Unser Programm stellt eine Erweiterung zur Funktionalität von BOA dar.
Es ist in der Lage bestimmte Einheitentypen im Satz zu finden und diese anschließend zu annotieren. Ein Label kann auf verschiedene Weisen in einem Satz dargestellt werden. Typische Beispiele sind hierbei die verschiedenen Möglichkeiten ein Datum oder ein Gewicht in einem Satz anzugeben.

January 1, 2003	1,5 kg
January first, 2003	1.5 kg
1.1.2003	1500 grams
January 1st, 2003	1500 g
...	...

Unser Framework soll außerdem in der Lage sein, für ausgewählte Datentypen und Einheiten diese verschiedenen Formen eigenständig von einer gegebenen Oberflächenform abzuleiten. Die Produktivität unseres Programmes soll unter anderem durch die Verwendung von mehreren Threads, Config-Dateien, Scoring Funktionen und Logging gesteigert werden. Letzentliches Ergebnis unseres Projektes ist ein GUI-Prototyp, der die grundlegende Funktionalität anhand einiger ausgewählter Beispielsätze verdeutlicht.

2 Produktübersicht

2.1 Erstellung von Annotationen

Zur Annotationserstellung wird ein Satz benötigt, der einen von uns definierten Datentyp enthält. Die Sätze könnten beispielsweise in einer `txt`-Datei gespeichert sein.
Zunächst müsste dieser Satz mittels *StanfordCore NLP* tokenisiert werden, um anschließend unser Programm nach vorkommenden Datentypen suchen zu lassen. Findet unser Programm einen zu annotierenden Datentyp, so wird in der `xml`-Datei, in dem der Satz nach erfolgreicher Bearbeitung gespeichert wird, unter dem Reiter `<annotation>` die entsprechende Annotation eingetragen (genauere Erläuterungen dazu kommen in 4.4 Speicherstruktur). Bei der Suche nach den Datentypen sollen mögliche Vorkommensweisen, nach denen der Satz im Anschluss durchforstet wird, in einer Config-Datei hinterlegt sein. Dies hat zur Folge, dass sich mögliche Ergänzungen leicht vornehmen lassen.

2.2 Ermitteln von Oberflächenformen

Bei der Erstellung der verschiedenen Oberflächenformen wird unserem Programm ein Label und der Datentyp übergeben. Diese können auch durch die in 2.1 angedeutete Annotationserstellung erstellt worden sein. Auf Basis der einzelnen Datentypen definieren wir die verschiedenen Möglichkeiten in denen ein Datentyp auftreten kann. Bei Einheiten wie Masse, Länge oder Temperatur ist das Umrechnen in andere Einheiten ebenfalls Bestandteil unseres Programms. Die Umrechnungsalgorithmen sind in Config-Dateien hinterlegt, um später in der Lage zu sein, Verbesserungen oder Ergänzungen vornehmen zu können, ohne den Quellcode dabei bearbeiten zu müssen.

3 Anforderungen

3.1 Zwingende Anforderungen

Die zwingenden Anforderungen an unser Programm gliedern sich einerseits in eine lauffähige Umsetzung der oben genannten Funktionen für die Datentypen Datum, Masse, Längeneinheit und Temperatur. Die verwendeten Einheiten, welche gefunden und umgerechnet werden sollen, sind:

Einheitentypen	**Einheiten**
weight	μg, mg, g, kg, t, lbs, Kt (Karat), oz (Unze)
linear measure	μm, mm, cm, dm, m, km, ft, inch, yard, miles, seamiles
temperature	°C, °F , K

Diese sind selbstverständlich jederzeit durch weitere Einheiten erweiterbar.
Andererseits möchten wir neben der eigentlichen Funktionalität sowohl die Produktivität, als auch die Variabilität unseres Programms erhöhen. Diese Aspekte werden wir mittels Multithreading, der Verwendung von Config-Dateien und dem Scoring aller Teilfunktionen erreichen. Die Scoring Ergebnisse sind Bestandteil der wöchentlichen Abgabe.
Die zu erstellenden Oberflächenformen bestehen bei den Datentypen Gewicht, Längeneinheit und Temperatur aus der Standardangabe in allen anderen Einheiten des Datentyps und den verschiedenen Schreibweisen, in denen die eigentliche Einheit auftauchen kann. Ein Vorkommen von $7\,°C$ würde beispielsweise die Ermittlung folgender Oberflächenformen ermöglichen:

- 44,6 °F
- 280 K
- 7 degree centigrade
- 7 degree C

Bei Datumsangaben ist keine Umrechnung in andere Einheiten möglich, dennoch gibt es verschiedenartige Möglichkeiten für das Auftreten von Daten. Die Angabe *July 1, 1969* kann auch folgendermaßen in einem Satz vorkommen:

- July 1st 1969
- July first 1969
- 07/01/1969
- 1st July 1969
- first July 1969
- 1.July 1969
- 01.07.1969
- 1969/07/01
- 1969 July 1st

3.2 Optionale Anforderungen

Eine unserer möglichen Erweiterungen stellt das Hinzufügen der Größe Geschwindigkeit dar. Der Aufwand würde bei 10% liegen. Die angegeben Prozente stehen hierbei immer im Verhältnis zum Aufwand, den wir für die Umsetzung aller zwingenden Anforderungen planen.
Die Möglichkeit einzelne Einheiten Plug-Ins auszuwählen oder abzuschalten, um die Leistung des Programms zu optimieren, ordnen wir mit einem Mehraufwand von 5% ein. Des Weiteren wäre es möglich dem Benutzer die Mittel zu geben, selber neue Einheiten zu einem bereits vorhandenen Einheitentyp hinzuzufügen. Dafür müsste er lediglich die Bezeichnung und die Umrechnung in unsere Standardeinheit angeben. Die Implementierung dieser Anforderung liegt bei 5%. In unseren zwingenden Anforderungen ist bereits eine einfache GUI mit integriert. Eine mögliche Erweiterung dieser sehen wir bei etwa 5% Mehraufwand.

Umsetzung für Geschwindigkeiten:	10%
Ein-/Abschalten einzelner Plug-Ins:	5%
Möglichkeit für das Hinzufügen von neuen Einheiten für den Benutzer:	5%
Erweiterung der GUI:	5%

4 Technologien und Vorgehensweisen

4.1 Multithreading

Um die Kapazitäten des CPUs geeignet auszunutzen und pro Zeiteinheit mehrere Aufgaben gleichzeitig ausführen zu können, werden wir Multithreading nutzen.
Dabei werden wir Teile unseres Programms dazu befähigen, nebeneinander abzulaufen, ohne sich dabei gegenseitig zu behindern, Ergebnisse zu verändern, oder den Gesamtablauf extrem zu verlangsamen. Ein Thread wird dabei genau eine Art von Maßeinheit bearbeiten. Diese Einteilung wird die Sychronisation wesentlich vereinfachen. `Java` liefert bereits alle nötigen Klassen und Methoden um dies umzusetzen. Wir werden die Möglichkeit nutzen, eine Klasse von `Thread` abzuleiten und `start()` sowie `run()` geeignet zu überschreiben. Dies hat den Vorteil, dass wir alle Methoden der Klasse `Thread` nutzen können. Unsere Threads laufen über einen Satz und werden dabei zu jeweils einem Ergebnis kommen. Diese Ergebnisse müssen dabei für jeden Satz übergeordnet zusammengeführt werden. Teilen sich mehrere Threads eine Ressource oder ein Objekt, können Codesequenzen durch das Schlüsselwort `synchronized` markiert werden, um Fehler zu vermeiden. Für das Objekt eines Satzes, der annotiert werden soll, wäre dies eventuell sinnvoll. Um die Kommunikation zwischen den Threads geeignet zu managen, werden die Funktionen `wait()` und `notify()` genutzt. Zum festgesetzten Zeitpunkt wird ein Thread dabei in Ruhe versetzt, oder geweckt.Unnötig häufige Abfragen von Threads, die auf Ergebnisse warten, werden damit vermieden. Gesondert zu beachten, ist das Ende der Threads. Allgemein ist ein Thread beendet, wenn die `run()`-Methode ohne Fehler beendet wurde, eine *RuntimeException* auftritt oder der Prozess von außen beendet wurde. Letzteres wäre potenziell problematisch. Jeder unserer Threads wird den Satz überlaufen und alle Annotationen, die zu einer Kategorie gehören sammeln. Dabei wird der Thread an sich, also theoretisch, unendlich laufen, solange bis eine Abbruchbedingung angezeigt wird, dann gibt es keine weiteren Annotationen. Wenn keine Annotationen mehr zu finden sind, die diese Kategorie betreffen, wird das Ergebnis an die übergeordnete Instanz gemeldet und der Thread automatisch beendet, da alle Operationen ausgeführt wurden. Um einen fehlerhaften Thread zu behandeln, nutzen wir eventuell das Interface `Callable`. Dabei wird es möglich sein, einen Fehler bzw. ein Problem beim Ausführen einer Aktion abzufragen. Damit ist abgesichert, dass keine unnötigen Prozesse auflaufen. Auf die

Verwendung der Methoden `stop()` oder `destroy()` werden wir verzichten, da nicht abgesichert werden kann, das deren Benutzung nicht zu unvorhergesehenen Problemen führen könnten.

4.2 Config-Dateien

In den Config-Dateien werden zum einen die Eigenschaften zum Finden einer Einheit in einem Satz festgelegt und zum anderen werden die Umrechnungsalgorithmen hinterlegt. Dies geschieht, um von außen leichter Veränderungen oder Ergänzungen vornehmen zu können, ohne dabei in den eigentlichen Klassen arbeiten zu müssen.

Eine config-Datei in dem Oberflächenformen hinterlegt sind, sähe wie folgt aus:

```
weight_surfaceforms=mg;milligram;g;gram;kg;kilogram;t:tones;pounds;lbs;ounze...
weightFalse_surfaceforms=graam;killogram...

temperature_surfaceforms=degree Celsius;degree Fahrenheit;Kelvin;K;°C;°F...
temperatureFalse_surfaceforms=F;C;Kalvin;degre;...

linearemeasure_surfaceforms=cm;m;km;ft;feet;inch;yard...
linearmeasureFalse_surfaceforms=meetre;foots;feed;...
```

Zum Auslesen der Surfaceforms müssten die jeweils im Programm benötigten Surfaceforms als Array gespeichert werden, um anschließend den Text danach abzusuchen.

Eine Config-Datei, die dazu dient die Umrechnungsfaktoren zu speichern, ist folgendermaßen aufgebaut:

```
weight_conversion_standard=g,0.001;t,1000;Kt,0.0002
weight_conversion_units=g,1000;t,0.001;Kt,5000

temperature_conversion_standard=°C,1+273.15;°F,(1+459.67)/1.8
temperature_conversion_units=°C,1-273.15;°F,(1-273.15)*1.8+32

lineare_measure_conversion_standard=cm,0.01;km,1000;yard,0.9144
lineare_measure_conversion_units=cm,100;km,0.001;yard,1.0936133
```

Zum Einlesen dieser Config-Dateien müssten 2 zweidimensionale Arrays eingelesen werden. Ein Array dient dazu, von einer Einheit in die jeweilige SI-Einheit des Datentyps zu gelangen. Das andere dient dazu, um von der SI-Einheit in alle anderen zu gelangen. Die verwendeten SI-Einheiten sind:

weight:	kg
temperature:	K
linear measure:	m

4.3 Scoring

Das Scoring nimmt eine wichtige Rolle im Bereich des Testings und der Effizienz unseres Programms ein. Durch die ermittelten Größen *precision*, *recall* und *f-score* erhalten wir Aussagen über die Trefferquote sowie die Genauigkeit, mit der unser Programm arbeitet. Diese ermöglichen einen leichten Einblick in den Fortschritt unserer Implementierung. Nach jeder implementierten Teilfunktion ist ein Scoring durchzuführen und am Ende der Woche mit abzugeben. Im Scoring

werden die *Goldstandard-XML* und die von unserem Programm erstellte `xml`-Datei Annotation für Annotation verglichen. Anhand dieser Vergleiche ergeben sich die Werte:

- `kp` – Anzahl korrekter Annotationen
- `fp` – Anzahl falscher Annotationen
- `kn` – Anzahl nicht gemachter Annotationen, an einem Label in dem im Goldstandard auch keine ist
- `fn` – Anzahl nicht gemachter Annotationen, an einem Label in dem im Goldstandard Annotation ist

Diese Werte bilden wiederum die Grundlage zur Ermittlung von *precision*, *recall* und somit auch *f-score*.

- *recall* (Trefferquote):

$$\frac{Anzahl\ korrekter\ Annotationen}{Anzahl\ Annotationen\ im\ Goldstandard}$$

- *precision* (Genauigkeit):

$$\frac{Anzahl\ der\ korrekten\ Annotationen}{Anzahl\ aller\ Annotationen}$$

- *f-score*:

$$2 \cdot \frac{recall \cdot precision}{recall + precision}$$

4.4 Speicherstruktur

Als Speicherformat für unsere annotierten Sätze verwenden wir eine `xml`-Struktur basierend auf den von *Stanford CoreNLP* erstellten `XML`s. Übernommen wurde der Grobaufbau mit indizierten Sätzen und Tokens. Außerdem erhält jeder Satz das mehrwertige Attribut `Annotation`, welches sich aus den referenzierten Tokens und dem Typ der jeweiligen Annotation zusammensetzt.
Hier eine Übersicht über unsere Dokumentstruktur anhand des Beispielsatzes: *The Mount Everest is 8,848 metres (29,029 ft) high.*

```
<root>
  <document>
    <sentences>
      <sentence id="1">
        <tokens>
          <token id="1">
            <word>The</word>
            <CharacterOffsetBegin>0</CharacterOffsetBegin>
            <CharacterOffsetEnd>3</CharacterOffsetEnd>
          </token>
          ...
          <token id="5">
```

```
                <word>8,848</word>
                <CharacterOffsetBegin>21</CharacterOffsetBegin>
                <CharacterOffsetEnd>26</CharacterOffsetEnd>
            </token>
            <token id="6">
                <word>metres</word>
                <CharacterOffsetBegin>27</CharacterOffsetBegin>
                <CharacterOffsetEnd>33</CharacterOffsetEnd>
            </token>
            <token id="7">
                <word>-LRB-</word>
                <CharacterOffsetBegin>34</CharacterOffsetBegin>
                <CharacterOffsetEnd>35</CharacterOffsetEnd>
            </token>
            <token id="8">
                <word>29,029</word>
                <CharacterOffsetBegin>35</CharacterOffsetBegin>
                <CharacterOffsetEnd>41</CharacterOffsetEnd>
            </token>
            <token id="9">
                <word>ft</word>
                <CharacterOffsetBegin>42</CharacterOffsetBegin>
                <CharacterOffsetEnd>44</CharacterOffsetEnd>
            </token>
            ...
        </tokens>
        <annotation id="1">
            <token>5</token>
            <token>6</token>
            <type>LINEAR_MEASURE</type>
        </annotation>
        <annotation id="2">
            <token>8</token>
            <token>9</token>
            <type>LINEAR_MEASURE</type>
        </annotation>
    </sentence>
  </sentences>
 </document>
</root>
```

4.5 Logging

Das Logging wird hauptsächlich mit der Bibliothek *log4j* realisiert. Logging Nachrichten werden dabei in die log-Datei `boa-framework.log` abgelegt. Zusätzlich wird die Bibliothek *slf4j* verwendet, um die Möglichkeit zu geben, verschiedene Logging-Tools zu nutzen und gegebenenfalls auszutauschen.
Die Form der log-Ausgaben wird mit Hilfe einer Konfigurationsdatei `log4j.properties` fest-

gelegt. Dabei werden unter anderem das Datum, die Uhrzeit, das log-Level sowie die genaue Angabe, in welcher Klasse bzw. Methode und Quellcodezeile eine log-Nachricht aufgerufen wird.

Bsp.: `2012-04-01 17:32:41,721 WARN «LogTest.java:16» [main] - Test`

4.6 Verwendete Pakete und Funktionen

Die zu implementierenden Funktionen gliedern sich größtenteils nach den beiden Hauptfunktionen des Programms. So sind für jeden Datentyp, Funktionen zu implementieren, die die Einheiten des Datentyps in einem Satz finden und eine Annotation machen sowie jene die für die Umrechnung bzw. Umformung eines gefundenen Labels mit Datentyp zuständig sind.
Die ersten zu implementierenden Elemente stellen jedoch die Scoring-Funktionen sowie die benötigten Funktionen für eine laufende Demo-GUI dar. Die Demo-GUI ermöglicht es von außen, jederzeit den Stand unserer Implementierung an selbst gewählten Beispielsätzen zu überprüfen.

- zu implementierende Funktionen:
  - Scoring
  - Demo-GUI
  - Einheiten:

Gewicht	**Länge**	**Temperatur**
Finden von μg	Finden von μm	Finden von K
Finden von mg	Finden von mm	Finden von °C
Finden von g	Finden von cm	Finden von °F
Finden von kg	Finden von dm	Finden aller Temperaturen
Finden von t	Finden von m	
Finden von lb	Finden von km	
Finden von Kt	Finden von ft	
Finden von oz	Finden von in	
Finden aller Gewichtseinheiten	Finden von yard	
	Finden von miles	
	Finden von seamiles	
	Finden aller Längeneinheiten	

Datum	**Umformungen**
Finden aller Datumsangaben	Umformung aller Gewichtseinheiten
	Umformung aller Längeneinheiten
	Umformung aller Temperaturen
	Umformung aller Daten

5 Deadlineplanung

Die Deadlineplanung richtet sich nach der Storyplanung (*siehe Aufgabenblatt 5*). Wie in der Storyplanung vereinbart wurde, nehmen die Erstellung der Demo-GUI, der Scoringfunktionen und der grundlegenden Funktionen, die anfänglichen Wochen ein, um eine schnelle sowie effektive Einsicht in den Fortschritt unseres Projekts zu ermöglichen. Daraus folgt, dass in den nachfolgenden Wochen die Präsentation unserer Ergebnisse mit verhältnismäßig wenig Aufwand verbunden sein wird.