XML Projekt - Dokumentation

Gruppe 3

Betriebssystem

Das Projekt wurde unter Xubuntu 64-Bit entwickelt und ausgeführt.

Installation und Konfiguration

Um das Programm zu installieren ist das Installationsskript als Root auszuführen

```
> sudo install_script.sh
```

Dabei werden alle nötigen Programme mit installiert und die Datenbank angelegt.
Standardmäßig ist der Webserver auf den Port 8345 und der Proxyserver auf den Port 8346 eingestellt. Bitte nehmen Sie die entsprechenden Einstellungen für den Proxyserver in Ihrem Browser vor (http://localhost:8346).

Ausführung

Um das Programm zu starten navigieren Sie in den Ordner `code` und führen Sie den Befehl

```
> python make run
```

aus. Damit starten der Web- und der Proxyserver.

Webserver:

Der Webserver kann im Browser unter http://localhost:8345 erreicht werden.
Hier steht die History bereit, sowie einige grafische Beispielanwendungen.

Proxyserver:

Nachdem der Proxyserver im Browser konfiguriert wurde, können Sie die 4 eingerichteten Datasets aufrufen. Dabei wird im Hintergrund vom Proxyserver das Eintragen der Daten vorgenommen.

Datasets

Die vier folgenden Datasets können angesurft und verarbeitet werden:

Twitter:

Bei Twitter ist es möglich die JSON Informationen aus der Seite heraus zu parsen.
Um eine Twitter Ressource anzusurfen, ist es nötig den unten stehenden Link zu benutzen. Sie werden dann automatisch an die Twitter-Seite des entsprechenden Users weitergeleitet. Um einen anderen User anzusurfen, ändern sie lediglich das Attribut `screen_name`, was den Twitter-Usernamen angibt.
Zu beachten: Unter Firefox können Probleme auftreten, da Twitter eine SSL-Verschlüsselung benutzt, welche der Proxy nicht verarbeiten kann. Mit dem Opera-Browser war es hingegen möglich die Ressource korrekt aufzurufen, da wir hier die SSL Verschlüsselung igonieren.

Beispiel Link:
http://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=sbahnberlin&count=1

Stackoverflow:

Bei Stackoverflow ist es möglich die Microdata Informationen aus der Seite heraus zu parsen.

Beispiel Link:
http://stackoverflow.com/questions/17435834/jquery-border-plugin

Entscholar:

Bei Entscholar ist es möglich die XML Informationen aus der Seite heraus zu parsen.

Beispiel Link:
http://www.entscholar.net/index.php/index/oai?verb=ListRecords&metadataPrefix=oai_dc

DBpedia:

Bei DBpedia ist es möglich die LinkedData Informationen aus der Seite heraus zu parsen.
Hierbei ist es wichtig die Ressource anzusurfen, anschließend wird man automatisch auf die data-Seite weitergeleitet.

Beispiel Link:
http://dbpedia.org/resource/Berlin

Visualisierung

Der Proxyserver arbeitet transparent, dass heißt der Benutzer erstmal nicht mitbekommt, dass die Metadaten wie auch Rohdaten in der 4store-Datenbank gesammelt werden. Möchte er diese untersuchen, so kann er dies über eine Weboberfläche tun.

Entsprechend haben wir uns bei unserer Visualisierung auf javascript-basierte Lösungen konzentriert, wobei die Wahl auf ***sgvizler*** als unsere Visualisierungs-API fiel, da diese RDF Graphen entgegen nehmen kann. Desweiteren standen weitere Visualiserungsmöglichkeiten zur Auswahl: ***MooWheel*** und ***infoVis***, wobei hier spezielle jSON Objekte nötig sind, wodurch zunächst nur sgvizler unterstützt wird.

Surft der Benutzer den Http-Server an, so sieht er eine Index-Seite, von welcher aus er die History aufrufen kann, interessante Query-Beispiele sich anschauen kann, und selbst Queries erstellen. Das zeichnen eines **s c h ö n e n** RDF-Graphen ist uns leider nicht gelungen, weshalb diese geplante Visualiserung, zum Beispiel als “Force directed Graph”, leider bis jetzt nicht eingebunden ist.

Die History zeigt besuchte Seiten mit einem Timestamp an. Als interessante Query Beispiele haben wir einerseits ein Kuchendiagramm, welches verdeutlicht für wie viele Menschen ein Bürgermeister einer Stadt zuständig ist und andererseits das Hervorheben von deutschen Städten auf einer Landkarte, deren DBpedia-Artikel man besucht hat.

RDF Datenbank

Wir verwenden als interne Datenbank und Triplestore das Programm 4store. Dieses soll eine gute Skalierbarkeit wie auch allgemeine Performance und hohe Ausfallsicherheit gewährleisten. Wir benutzen die aktuellste Version, welche wir selbst kompilieren mussten.
4store bietet einen http-Daemon an, über den Anfragen an die Datenbank gestellt werden können, so zum Beispiel auch durch sgvizler.

Verwendete Technologie

Der größte Teil des Projektes wurde in Python entwickelt. Mit Python ist es relativ einfach einen Web- und Proxyserver zu implementieren. Zudem lassen sich die Parser mit der Skriptsprache relativ leicht entwickelt, da beispielsweise reguläre Ausdrücke leicht verwendet werden können. Außerdem stand uns ein XSLT Parser für die XML Quellen zur Verfügung, der ohne große Schwierigkeiten in Python benutzt werden kann.

Beispiel Queries

sMap

File Location: `code\server\visualizationx\sgvizler-0.5\tests\sMap.html`

Hinweis: Die Prefixes wurden im nachfolgenden Beispiel Query entfernt.

```
SELECT DISTINCT ?lat ?long ?name ?text ?url ?image WHERE
{
	{
	?url dbpedia-owl:country db:Germany .
	?url foaf:name ?name .
	?url geo:lat ?lat .
	?url geo:long ?long .
	}
UNION
	{
	?url dbpedia-owl:country db:Germany .
	?url dbpprop:name ?name .
	?url geo:lat ?lat .
	?url geo:long ?long .
	}
UNION
	{
	?url dbpedia-owl:country db:Germany .
	?url dbpprop:centre ?name .
	?url geo:lat ?lat .
	?url geo:long ?long .
	}
UNION
	{
	?url dbpedia-owl:country db:Germany .
	?url dbpprop:location ?name .
	?url geo:lat ?lat .
	?url geo:long ?long .
	}
}
```

Das Beispiel Query sucht alle über DBpedia angesurften deutschen Städte heraus. Diese werden in unserem Beispiel in einer Karte dargestellt. Da DBpedia bei der Bezeichnung der Städte leider nicht sehr konsequent ist, können eventuell nicht alle Städte durch das Query gefunden werden. Getestet wurden jedoch große deutsche Städte, wie Berlin, Hamburg oder München.

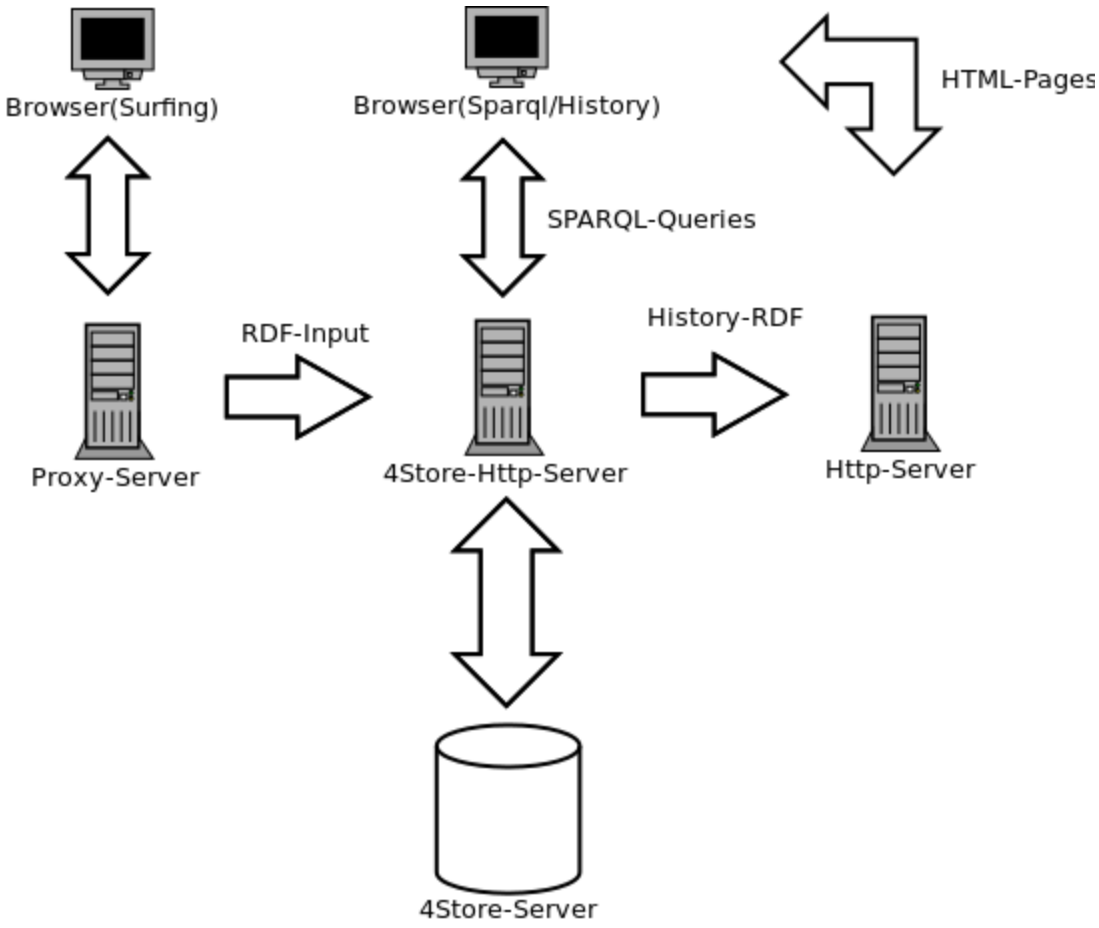
PieChart

File Location: `code\server\visualizationx\sgvizler-0.5\tests\pieChart.html`
Hinweis: Die Prefixes wurden im nachfolgenden Beispiel Query entfernt.

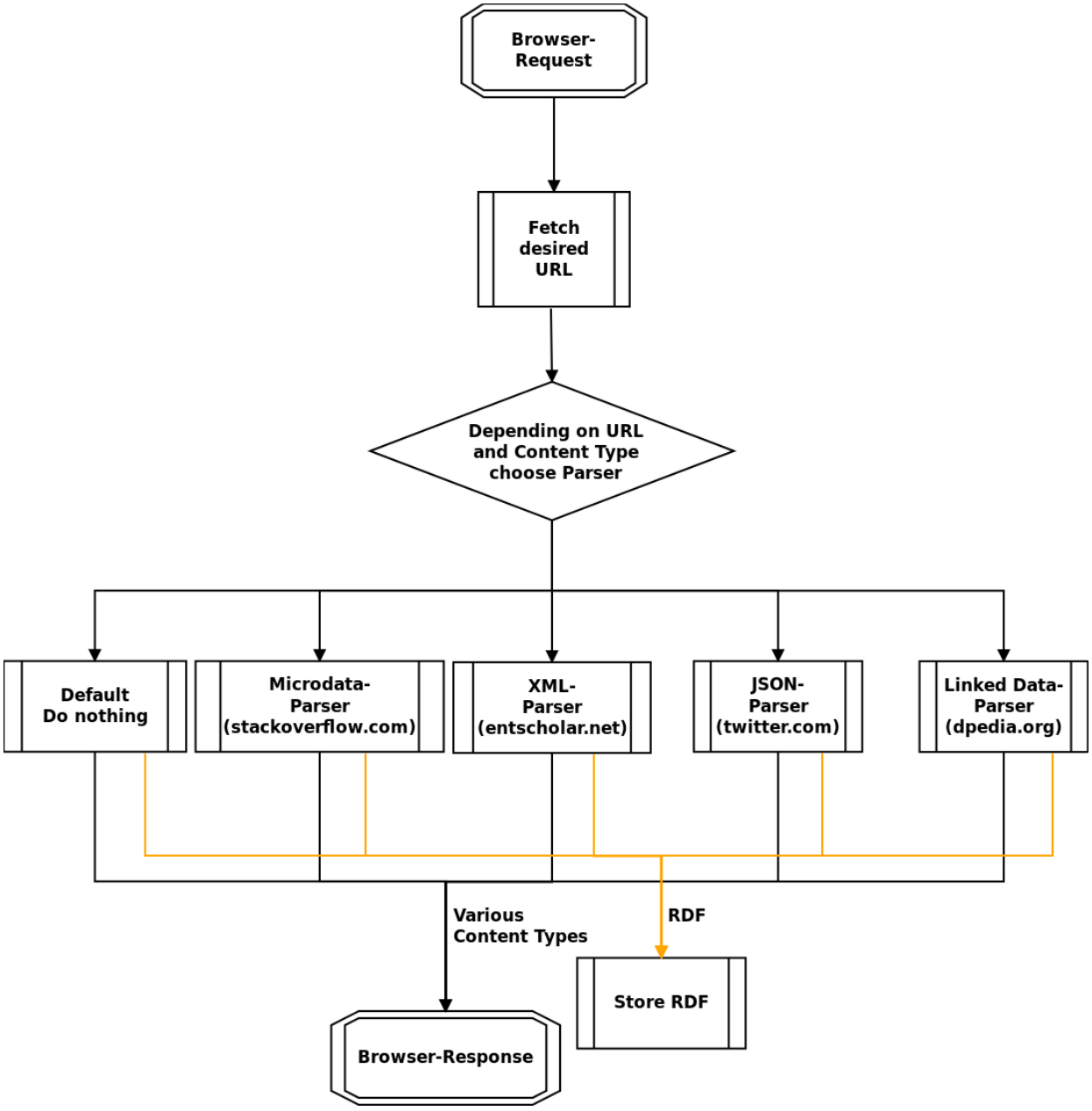
```
SELECT DISTINCT ?mayor ?population WHERE
{
      {
      ?town dbpedia-owl:leader ?mayor .
      ?town dbpedia-owl:populationTotal ?population .
      }
UNION
      {
      ?town dbpedia-owl:leaderName ?mayor .
      ?town dbpedia-owl:populationTotal ?population .
      }
UNION
      {
      ?town <http://dbpedia.org/property/b%C3%BCrgermeister> ?mayor .
      ?town dbpedia-owl:populationTotal ?population .
      }
FILTER (?population > 1000000)
} ORDER BY DESC(?population) LIMIT 10
```

Das Beispiel Query sucht die Bürgermeister aller besuchten Städte, welche mindestens 1 Million Einwohner haben. Anschließend wird dies in einem PieChart dargestellt.
Auch hier ist wieder zu beachten, dass aufgrund von Inkonsistenzen bei DBpedia eventuell nicht alle Daten mit dem Query erfasst werden, beziehungsweise auch Daten erfasst werden, die eigentlich nicht mit dazu gehören, da beispielsweise ein Subjekt mit dem Prädikat `dbpedia-owl:leaderName` nicht nur auf den Bürgermeister zeigt, sondern auch auf andere Objekte.

Server-Architektur:



Proxy-Flow


Browser-
Request
Fetch
desired
URL
Depending on URL
and Content Type
choose Parser
Default
Do nothing
Microdata-
Parser
(stackoverflow.com)
XML-
Parser
(entscholar.net)
JSON-
Parser
(twitter.com)
Linked Data-
Parser
(dpedia.org)
Various
Content Types
RDF
Store RDF
Browser-Response