Diplomarbeit

Entwicklung und Realisierung einer Strategie zur Syndikation von Linked Data

vorgelegt von
Raphael Doehring

Betreuer:

Prof. Dr. Ing. habil. Klaus-Peter Fähnrich
Dr. rer. nat. Dipl.-Math. Sören Auer

Institut für Informatik, Abteilung Betriebliche Informationssysteme
Universität Leipzig

Leipzig, im Februar 2010

Ich möchte mich vielmals bei meinem Betreuer, Sören Auer, für die Zeit die er sich jedes mal für mich genommen hat bedanken. Danke auch an die anderen Mitglieder der AKSW-Forschungsgruppe, für viele Antworten auf viele Fragen.

Ferner gilt mein Dank der Firma Netresearch, und im Besonderen Christian Weiske, für Unterstützung und zahlreiche sachdienliche Hinweise bezüglich PHP und darüber hinaus.

Raphael Doehring, im Februar 2010

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Das Semantic Web beginnt unsere Möglichkeiten zur Nutzung des Internet nachhaltig zu verändern. Suchmaschinen werten semantische Daten aus, viele Seiten werden mit semantischen Zusatzinformationen ausgezeichnet. Um die Entwicklung des Semantic Web voranzutreiben, wurde eine große Menge von Projekten zur Schaffung frei zugänglicher Resource Description Framework-Daten (RDF) gestartet. Diese waren sehr erfolgreich, so dass inzwischen über 13 Milliarden Datensätze[1] erreichbar sind. Die Veröffentlichung und Verwendung von freien Daten hat sich nach anfänglichem Erfolg jedoch nicht im großen Maße durchgesetzt. Dies liegt vor allem an der Qualität der veröffentlichen Daten und an der Zugänglichkeit für den Endnutzer.

Es gibt bereits viele wissenschaftliche Projekte und Programme zur Nutzung und Verarbeitung dieser Daten. Eine einfache Möglichkeit die freien RDF-Daten direkt weiterzuverwenden, sie zum Beispiel in eine Webseite oder einen Blog zu integrieren, fehlt jedoch bis jetzt. Es ist für einen Internetnutzer ohne genaue Kenntnisse der Technologie nicht möglich von der Masse an Daten zu profitieren. Ziel dieser Arbeit ist es, eine Strategie zur einfachen Wiederverwendung von RDF-Daten zu entwickeln und in einer prototypischen Implementierung zu realisieren.

Die hier entwickelte Webapplikation LESS ist eine Templateengine für Semantic Web-Daten. Sie bietet eine Plattform zur Bearbeitung, Veröffentlichung und Wiederverwendung dieser Templates. Daten können aus verschiedenen Quellen integriert und in ein beliebiges Textformat umgewandelt werden. Templates können für die flexible Darstellung von Daten, deren Umwandlung oder die gezielte Syndikation eingesetzt werden. So können

[1] http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/DataSets/%20Statistics

von einfachen HTML-Seiten, über RSS-Feeds bis hin zu Mashups, unter Verwendung von HTML und Javascript, unterschiedlichste Ergebnisse produziert werden.

Da ein wichtiges Ziel die Verbesserung der Benutzerfreundlichkeit der Semantic Web-Daten ist, wird der Anwender bei der Erstellung von Templates unterstützt, so dass nur ein geringes technisches Wissen für die Verwendung von LESS nötig ist. Weiterhin ist die Integration eines fertigen Templates in eine Webseite oder einen Weblog sehr einfach möglich.

1.2 Aufbau der Arbeit

Kapitel 2 erklärt die Grundlagen aus dem Bereich des Semantic Web. Es erläutert wichtige Begriffe und gibt einen Überblick über die bestehenden Technologien. Kapitel 3 entwirft verschiedene Anwendungszenarien, welche die Anforderungen an die Arbeit definieren. Das darauffolgende Kapitel 4 stellt die Lösungsidee vor. Im nächsten Teil, Kapitel 5, werden die Einzelheiten der technischen Umsetzung erläutert. Kapitel 6 zeigt die Anwendung der entwickelten Lösung anhand von umgesetzten Beispielen. Kapitel 7 stellt wichtige verwandte Arbeiten vor. Der vorletzte Abschnitt der Arbeit, Kapitel 8, erläutert Schlussfolgerungen und gibt einen Ausblick. Die Arbeit schließt in Kapitel 9 mit einer Zusammenfassung.

Kapitel 2

Grundlagen

Dieses Kapitel beschreibt die theoretischen und technischen Grundlagen für die hier vorliegende Arbeit. Darüber hinaus werden wichtige Begriffe eingeführt und erläutert.

2.1 Das Semantic Web

Das World Wide Web (WWW) ist ein Netz zur Veröffentlichung von Dokumenten. Jeder Mensch mit Zugang zu diesem Netz kann Dokumente der Öffentlichkeit zugänglich machen und durch Links miteinander verbinden. In vielen Fällen werden Daten für Webseiten in einer Datenbank vorgehalten. Bei Veröffentlichung in einem Webdokument verlieren diese Daten leider oft ihre eindeutige Struktur und damit einen Großteil ihrer Bedeutung (Semantik) [ADL+09]. Der Mensch kann die Zuordnung meist noch durch den Kontext der Webseite herstellen. Einer Maschine ist es allerdings nur mit großen Aufwand und hoher Ungenauigkeit möglich, diese Daten korrekt weiterzuverarbeiten.

Das Semantic Web (SW) verkörpert deshalb die Vision einer Erweiterung des WWW. Neben den eigentlichen Dokumenten werden explizit ausgezeichnete Daten direkt veröffentlicht. Die explizite Auszeichnung soll es ermöglichen, dass ein Computer die Bedeutung der Daten korrekt zuordnen kann. Diese können dann sowohl von Menschen als auch von Maschinen einfach weiterverwendet werden. Abgesehen von einer präziseren Suche im Web, können auf dieser Grundlage ganz neuartige Applikationen geschaffen werden.

2.2 Das Linked Data Web

Um neue SW-Applikationen erstellen und deren Funktionalität testen zu können, ist eine große Menge von echten, miteinander verbundenen Semantic Web-Daten notwendig [BHAR07]. Das *W3C SWEO6 Community*[1] Projekt *Linking Open Data*[2] (LOD) wurde ins Leben gerufen, um diese Daten zu schaffen und somit die Weiterentwicklung des Semantic Web zu gewährleisten. Erklärtes Ziel war es, frei lizensierte und öffentlich zugängliche Daten zu finden und in das Semantic Web-Format zu transformieren. Das daraus entstandene Netz an Daten wird auch als Linked Data Web (LDW) oder als LOD-Cloud, als Wolke von Linked Data bezeichnet. Das Projekt war sehr erfolgreich. Inzwischen ist das LDW auf eine Größe von 13 Milliarden (Stand 25.11.2009)[3] Datensätzen (siehe Abschnitt 2.2.2) angewachsen. Eine große Menge an Daten ist im Projekt DBpedia enthalten [LBK+09], das sich zur Aufgabe gemacht hat, automatisiert Informationen aus Wikipedia-Artikeln zu extrahieren und in das Semantic Web-Format (siehe Kapitel 2.2.2) umzuwandeln. Immer mehr Firmen (BBC[4], New York Times[5]), Institutionen (Library of Congress[6]) und sogar Regierungen (Großbritannien[7]) setzen LDW-Standards zur Bereitstellung ihrer öffentlichen Daten ein.

2.2.1 Linked Data Regeln und eindeutige Bezeichner

Die technische Grundlage für die Veröffentlichung der Daten bilden die vier Linked Data-Regeln, welche Tim Berners-Lee in [BL06] vorgeschlagen hat.

1. Verwende URIs, um Dinge zu benennen.
2. Verwende HTTP URIs, sodass Personen nachschauen können, was sich hinter einer URI verbirgt.
3. Wenn jemand die URL verwendet, biete hilfreiche Informationen an, verwende Standards (RDF, SPARQL).
4. Verwende Links zu anderen URIs. Das ermöglicht die Entdeckung von neuen Daten.

[1] http://www.w3.org/2001/sw/sweo/
[2] http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[3] http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/DataSets/%20Statistics
[4] http://welcomebackstage.com/
[5] http://data.nytimes.com/
[6] http://id.loc.gov/authorities
[7] http://data.gov.uk/

Bei der Erstellung von Metadaten sind eindeutige Bezeichner für alle Dinge, die beschrieben werden, unerlässlich. Regel 1 schlägt dafür die, ebenfalls im RDF-Standard (siehe Abschnitt 2.2.2) verwendeten, Uniform Resource Identifiers (URIs) vor. URIs sind eindeutige Bezeichner für konkrete oder abstrakte Dinge. Im Folgenden werden Dinge, die durch RDF-Aussagen beschrieben werden, als Ressourcen bezeichnet. URIs bestehen grundsätzlich aus einem Schemateil und dem eigentlichen Bezeichner. Bekannte Schemata sind `ftp`, `tel`, `http` und `mailto`. Beispiele für URIs sind:

```
ftp://ftp.heise.de/pub/ct/ctsi/hdtool11.zip
tel:+49-341-66778899
mailto:info@aksw.org.
```

Bei der Veröffentlichung von Linked Data werden laut Regel 2 nur HTTP URIs verwendet. Diese beginnen mit `http` und sind als URLs aus dem WWW bekannt. Die URL

```
http://raphas.net/projects/2009/thesis
```

z. B. ist der eindeutige Bezeichner für diese Diplomarbeit.

URLs haben hauptsächlich zwei Vorteile.

- Sie können gleichzeitig Bezeichner der Ressource und Lokation der Datenquelle sein.
- Sie lösen das Problem der Eindeutigkeit von Ressourcenbezeichnern sehr elegant durch Ausnutzung der weltweit exklusiven Zuteilung von Webadressen.

Regel 3 verlangt, dass nach Auflösung dieser URL hilfreiche Informationen über die Ressource oder ein Hinweis auf mögliche Datenquellen zur Ressource angeboten werden. Entweder sind die Daten zur Ressource direkt unter der URL erreichbar oder es wird mindestens ein Hinweis darauf gegeben, wo weitere Daten aufzufinden sind. Im nun folgenden Abschnitt wird das dazu verwendete Datenformat beschrieben.

2.2.2 Das Datenformat des Linked Data Web

Das zentrale Datenformat des LDW ist das Resource Description Framework (RDF). Dieses Rahmenwerk zur Beschreibung von Metadaten verwendet als Basis einfache Aussagen, die immer aus den drei Teilen Subjekt, Prädikat und Objekt bestehen. Eine solche Aussage wird als Tripel bezeichnet und beschreibt immer eine wahre Aussage zu einer Ressource. Prädikate werden auch als Properties (Eigenschaften) bezeichnet. Eine Aussage über diese Diplomarbeit ist zum Beispiel:

```
Diese Diplomarbeit hat den Autor Raphael Doehring.
```

RDF-Tripel lassen sich im Allgemeinen durch einen Graphen sehr gut verständlich visualisieren. Der Graph für die Aussage ist in Abbildung 2.1 zu sehen.

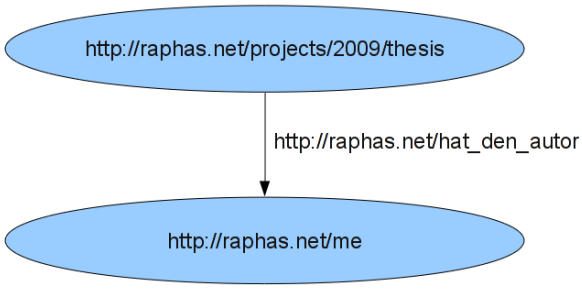


Bild 2.1: Repräsentation einer einfachen RDF-Aussage als Graph

Die URL `http://raphas.net/me` ist der eindeutige Bezeichner der mich, Raphael Doehring, repräsentiert.

RDF kann in verschiedenen Textformaten dargestellt werden. Die Notation 3 (N3) ist ein intuitiv verständliches Format, weshalb es zur Repräsentation von RDF in dieser Arbeit verwendet wird. N3 listet die einzelnen Teile einer Aussage hintereinander auf und schließt diese mit dem .-Symbol ab. Die N3-Repräsentation des oben beschriebenen Tripels ist:

```
<http://raphas.net/projects/2009/thesis>
     <http://raphas.net/hat_den_autor>
        <http://raphas.net/me> .
```

In RDF-Tripeln können neben Ressourcen auch Literale auftreten. Ressourcen werden durch eine URL repräsentiert und in N3 in spitze Klammern eingefasst. Literale hingegen sind konkrete Werte, wie z. B. Zahlen oder Zeichenfolgen. Sie können nur als Objekt eines Tripels auftreten und werden in N3 von Anführungszeichen umschlossen dargestellt.

Das folgende Beispiel zeigt die Darstellung eines Literals in N3

```
<http://raphas.net/me>
  <http://raphas.net/hat_den_namen>
    "Raphael Doehring" .
```

und in einem Graphen in Abbildung 2.2.

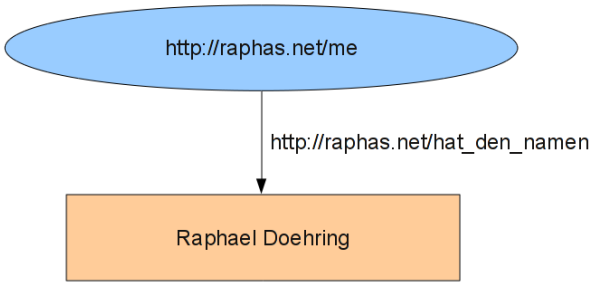


Bild 2.2: RDF-Aussage mit einem Literal

Die beiden Prädikate `http://raphas.net/hat_den_namen` und `http://raphas.net/hat_den_autor` sind für dieses Beispiel frei erfunden. Theoretisch kann in jedem Dokument ein neues Prädikat für Eigenschaften von Ressourcen verwendet werden. Eine gute Wiederverwendbarkeit der Daten ergibt sich aber erst, wenn alle Benutzer sie gleich interpretieren. Um dies zu ermöglichen, werden Ontologien entworfen. Sie beschreiben für eine bestimmte Domäne sowohl die darin vorkommenden Objekte, als auch deren Eigenschaften und Beziehungen. Ontologien lassen sich in RDF oder erweiterten Standards wie RDF-Schema (RDFS) oder der Web Ontology Language (OWL) erstellen. Das wohl bekannteste Beispiel für ein solches Vokabular ist *Friend of a friend (FOAF)*[8], welches Terme für die Beschreibung von Personen, deren Beziehungen und persönlichen Dokumenten enthält. Für die Beschreibung des Namens einer Person ist im FOAF-Vokabular der Bezeichner `http://xmlns.com/foaf/0.1/name` festgelegt. Die Verwendung der FOAF-Ontologie führt zu folgender Aussage:

```
<http://raphas.net/rdterms/me>
  <http://xmlns.com/foaf/0.1/name>
    "Raphael Doehring" .
```

Für Informationen zu Dokumenten wie Autor, Erstellungsdatum, Titel, usw. wird das Dublin-Core-Vokabular[9] verwendet. `http://purl.org/dc/elements/1.1/author` beschreibt darin die Relation, dass ein Dokument einen Autor hat.

Da die Basis-URL für unterschiedliche Properties aus einem Vokabular immer gleich ist, können Namensräume (namespaces) eingeführt werden, um die URLs zu verkürzen. Für FOAF wird in der Regel die Abkürzung `foaf` verwendet, für den Dublin Core `dc`.

Der folgende Graph (Abbildung 2.3) stellt die oben eingeführten Tripel und zusätzlich den Titel der Diplomarbeit dar. Auch dafür wird eine Element des Dublin Core verwendet.

[8] http://xmlns.com/foaf/0.1/
[9] http://dublincore.org/

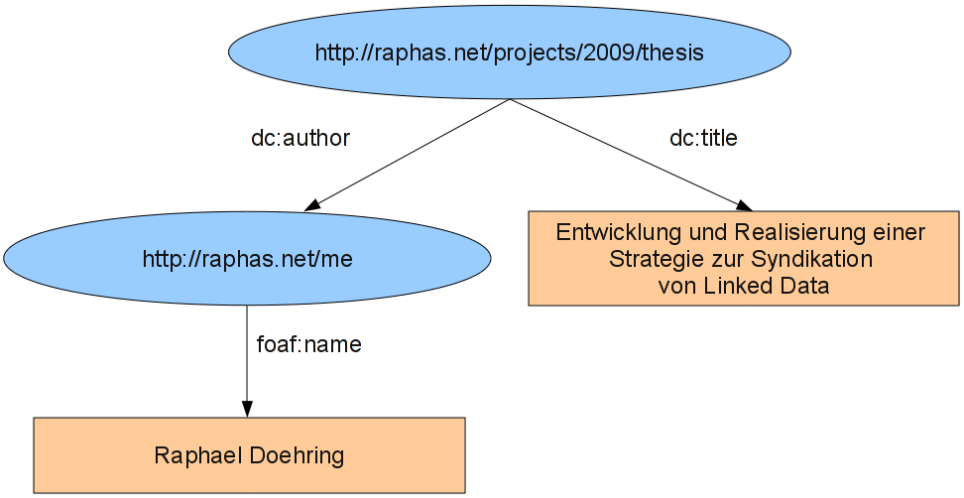


Bild 2.3: RDF-Graph mit drei Tripeln

Kernbestandteil des SW ist die Möglichkeit zur dezentralen Entwicklung von Vokabularen. Es gibt keine zentrale Instanz, die über die Verwendung von Vokabularen entscheidet. Es ist allerdings zu beobachten, dass sich für unterschiedliche Domänen bestimmte Vokabulare durchsetzen.

2.2.3 Speicherung von RDF-Daten

Kleinere Mengen von RDF-Tripeln können einfach in Textdateien gespeichert werden. Geht es nur darum, mithilfe des FOAF-Vokabulars persönliche Daten wie E-Mail, Telefonnummer oder aktuelle Projekte zu veröffentlichen, ist die Verwendung einer statischen Textdatei, die RDF enthält, völlig ausreichend. Bei größeren Datenmengen ist es sinnvoll die Tripel in einer speziellen Datenbank, einem so genannten Triplestore, zu speichern. Zugriff auf diese Daten ist dann i.d.R. auf zwei Wegen möglich. Viele Triplestores unterstützen die Veröffentlichung der Daten über Linked Data-URLs. Wenn eine URL angefragt wird, die im Namensbereich des Triplestores liegt, kann dieser überprüfen, ob Tripel mit dieser URL im Subjekt vorhanden sind. Ist das der Fall, werden die betreffenden Tripel in einer Textserialisierung ausgeliefert.

Eine andere Form des Zugriffs erfolgt über SPARQL (SPARQL Protocol and RDF Query Language). Dabei handelt es sich um eine SQL-ähnliche, graphbasierte Abfragesprache für Triplestores.

Im `PREFIX`-Teil der Abfrage werden die verwendeten Namensräume eingeführt. Der `SELECT`-Bereich definiert die Variablen, die im Ergebnis enthalten sein sollen. Der `WHERE`-Teil beschreibt die eigentliche Abfrage. Das folgende Beispiel zeigt eine solche SPARQL-

Abfrage:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpprop: <http://dbpedia.org/property/>

SELECT ?name ?website ?inhabitants
WHERE {
  ?city <rdfs:label> ?name .
  ?city <dbpprop:website> ?website .
  ?city <dbpprop;einwohner> ?inhabitants .
}
```

Ergebnis einer solchen Abfrage ist eine Tabelle, die für jede freie Variable eine Spalte mit den jeweiligen Werten enthält. Für diesen Fall entsteht die Tabelle, die in Abbildung 2.4 dargestellt ist.

name	website	inhabitants
Magdeburg	http://www.magdeburg.de/	230140
Stuttgart	http://www.stuttgart.de/	600038
Wiesbaden	http://www.wiesbaden.de/	275422
Düsseldorf	http://www.duesseldorf.de/	582222
Kiel	http://www.kiel.de/	236902
Mainz	http://www.mainz.de/	200234
Saarbrücken	http://www.saarbruecken.de/	180515

Bild 2.4: SPARQL-Ergebnistabelle

Kapitel 3

Problemstellung

Nach der Klärung der Grundlagen im letzten Kapitel umreißt das folgende Kapitel die Problemstellung und Motivation für diese Arbeit. Es werden im Weiteren konkrete Anforderungen an die Lösung daraus abgeleitet.

3.1 Ausgangssituation

Durch die Linking Open Data Initiative ist eine große Fülle von frei zugänglichen RDF-Daten entstanden. Diese werden oft für Forschungszwecke eingesetzt. Die Nutzung der Daten durch Menschen ohne Programmierkenntnisse ist nicht möglich. Dafür gibt es hauptsächlich zwei Gründe. Zum einen wissen viele Menschen außerhalb der Forschungsgemeinschaft nicht, dass es diese Fülle von freien Daten in dieser Form gibt. Zum anderen ist es im Moment ohne Programmierkenntnisse nicht möglich, die Daten in einfacher Form weiterzuverwenden. Selbst mit Kenntnis der nötigen Technologien ist der Aufwand zur Weiterverarbeitung der Daten relativ hoch. Außerdem müssen für jeden Anwendungsfall die Probleme des Datenzugriffs sowie der Datenverarbeitung neu gelöst werden.

Ziel dieser Arbeit ist es eine Strategie zur Wieder- und Weiterverwendung von Daten im Semantic Web zu schaffen, welche nur geringe technische Spezialkenntnisse verlangt. Als Ergebnis der Arbeit soll es möglich sein, mit grundlegenden Kenntnissen in HTML Daten aus dem semantischen Web weiterzuverarbeiten. Der Benutzer soll dabei möglichst einfach verständlich, visuell bei seiner Arbeit unterstützt werden.

3.2 Anwendungsszenarien

Die folgenden vier Anwendungsszenarien konkretisieren die verschiedenen Facetten der Anforderungen.

3.2.1 Flexible Darstellung von Linked Data Ressourcen

Daten im Linked Data Web liegen in einer festen Struktur im RDF-Format vor. RDF kann in XML oder Formaten wie N3 serialisiert werden. Auch eine Darstellung der Daten in einer Tripeltabelle ist einfach möglich. Sollen die vorhandenen Daten hingegen in ein benutzerdefiniertes Format gebracht werden, gibt es nur eingeschränkte Unterstützung durch Hilfsmittel. Besonders die flexible Darstellung einer bestimmten Ressource in einem HTML-Dokument ist nicht ohne weiteres möglich. Die hier entwickelte Lösung soll eine einfache und flexible Möglichkeit zur Darstellung von RDF-Daten bieten.

3.2.2 Integration von Darstellungen von RDF-Daten in Webanwendungen

Hat man eine solche Darstellung entworfen, stellt sich die Frage der Weiterverwendung und Integration. Eine solche Darstellung sollte als alleinstehende Webseite unter einer bestimmten URL abgerufen werden können. Eine wichtige Anforderung ist jedoch auch die einfache Integration in bestehende Webanwendungen. Es soll mithilfe der hier entwickelten Lösung ohne größeren Aufwand möglich sein, die erstellten Darstellungen in bestehende Webseiten oder Webanwendungen zu integrieren.

3.2.3 Syndikation von veränderlichen Inhalten

Die Weitergabe und Weiterverwendung (Syndikation) von sich verändernden Inhalten ist ein wichtiger Bestandteil des Web 2.0. News-Feeds (z. B. RSS-Feeds) haben eine hohe Verbreitung erlangt. Dabei handelt es sich um die Aggregation von sich verändernden Daten einer Webseite in einem festgelegten XML-Format. Dieser Mechanismus erlaubt es den Inhalt einer Webseite nicht mehr nur durch den Browser, sondern auch mithilfe spezialisierter Applikationen (Feedreader) zu betrachten. Es ist der Applikation einfach möglich festzustellen, ob auf einer bestimmten Webseite neue Beiträge existieren, die vom Benutzer noch nicht gelesen wurden.

Auch im Linked Data Web werden aktuelle, sich verändernde Daten veröffentlicht. Ein

Beispiel dafür sind die Programmdaten der BBC-Sendergruppe. Sie sind unter der Basis URL

```
http://www.bbc.co.uk/programmes
```

abrufbar. Um diese Daten zum Beispiel in einem RSS-Feed zu veröffentlichen, ist bisher eine programmatische Lösung von Nöten. Durch diese Arbeit soll es möglich sein, dieses Problem auch ohne Verwendung einer Programmiersprache zu lösen.

3.2.4 Mashups

Mashups sind ein weiterer bekannter Bestandteil des Web 2.0. Die grundlegende Idee bei einem Mashup ist die Verbindung unterschiedlicher Daten oder Applikationen zur Schaffung einer neuen Applikation. Die Erstellung solcher Mashups soll von der hier entwickelten Lösung unterstützt werden.

3.3 Weitere Anforderungen

Die mit der hier zu entwickelnden Applikation erstellten Darstellungen und Mashups sollen möglichst effizient wiederverwendet werden können. Insbesondere sollte es eine Möglichkeit geben nicht nur eigene Inhalte weiter zu verwenden, sondern auch auf der Arbeit anderer aufzubauen.

Da das Linked Data Web noch eine relativ junge Entwicklung istund einige Komponenten eher prototypischen Charakter haben, ist die Erreichbarkeit der veröffentlichten Daten nicht immer gewährleistet. Insbesondere Linked-Data-Endpunkte sind gelegentlich nicht oder nur schlecht erreichbar. In der hier entwickelten Lösung sollte dieses Problem speziell adressiert werden können. Insbesondere soll eine Weiterverwendung bestimmter semantischer Daten auch dann möglich sein, wenn die ursprüngliche Datenquelle nicht erreichbar ist.

Kapitel 4

Lösung

Bezugnehmend auf die im letzten Kapitel formulierten Anforderungen wird in diesem Kapitel eine grundlegende Idee zur Lösung der Problemstellung entworfen. Es werden dabei die zu entwickelnden Bestandteile in ihren wichtigsten Eigenschaften beschrieben.

4.1 Templatesprache

Als Methode zur Verarbeitung und flexiblen Umformatierung von bestehenden Daten haben sich Templatesprachen in der Programmierwelt durchgesetzt. Das Konzept ist ausgereift und es gibt viele Implementierungen, die auch kommerziell eingesetzt werden. Sie bieten die Möglichkeit Vorlagen (Templates) in bestimmten Formaten zu entwerfen und diese später mit Daten aus verschiedenen Datenquellen zu füllen (rendern). Für eine jeweilige Anwendung müssen lediglich die Vorlage und die anzuwendende Datenquelle genannt werden, um eine Repräsentation der Daten im neuen Format zu erhalten. Außerdem unterstützen Templates das Konzept der Wiederverwendung in hohem Maße, da für eine bestimmte Zielstellung i.d.R. nur einmal ein Template erstellt werden muss. Anschließend kann es unter Angabe unterschiedlicher Datenquellen immer wieder verwendet werden.

Die oben beschriebenen Eigenschaften bieten gute Voraussetzungen zur Lösung der zuvor beschriebenen Problemstellung. Kern der hier entwickelten Lösung wird eine Templatesprache sein, die es erlaubt Daten aus einer Linked Data-Quelle oder dem Ergebnis einer SPARQL-Abfrage zu adressieren, zu verarbeiten und in einem beliebigen Textformat anzuzeigen.

Templatesprachen haben gegenüber visuellen Editoren den Vorteil, dass sie flexibler und mächtiger sind. In einer Templatesprache können deutlich komplexere Ausdrücke unterstützt werden. Auf der anderen Seite ist die Verwendung einer Templatesprache immer

mit einem höheren Lernaufwand verbunden, während bei visuellen Editoren die Lernkurve deutlich flacher ist. Aus diesem Grund soll die Templatesprache zwar der Kern der hier entwickelten Lösung sein, jedoch in einen Editor integriert werden, der dem Benutzer das Erlernen der Sprache erleichtert.

4.2 Entwicklungsumgebung

Um die Entwicklung von Templates in einer bestimten Sprache zu unterstützen, bedarf es verschiedener Hilfsprogramme. Auf diese Programme soll im nun folgenden Abschnitt eingegangen werden.

Zu Beginn wird ein Werkzeug erstellt, welches den Benutzer bei der Entwicklung von Templates für RDF-Daten unterstützt. Es ermöglicht die Bearbeitung des Templatecodes. Es werden dabei Erleichterungen im Zugriff auf die Daten bzw. Sprachkonstrukte angeboten. Außerdem soll es eine Vorschaumöglichkeit geben, um eine Kontrolle und Fehlersuche bei der Entwicklung zu unterstützen.

Es wird weiterhin ein Programm zum Rendern der Templates entwickelt. Um eine möglichst niedrige Hemmschwelle für die Benutzung durch nicht technisch versierte Nutzer zu erreichen, ist es sinnvoll dazu einen Webservice anzubieten, statt den Benutzer eine Software herunterladen und installieren zu lassen. Der Webservice nimmt die Templatedaten sowie die Datenquelle entgegen, und liefert das fertig gerenderte Template zurück. Das Ergebnis wird einfach in eine bestehende Webseite und Applikation integriert werden können.

Um eine hohe Wiederverwendung von Templates zu erreichen, wird eine Plattform geschaffen, auf welcher Templates verwaltet werden können. Es wird einem Benutzer möglich sein die vorhandenen Templates zu durchsuchen und ihren genauen Inhalt anzuschauen. Desweiteren wird die Plattform alle notwendigen Informationen bereithalten, um das gefundene Template weiterzuverwenden.

Die explizit benannten Probleme des Linked Data Web (siehe Kapitel 3.3) werden besonders adressiert. Der Umstand der unzuverlässigen Erreichbarkeit von Datenquellen kann durch ein Zwischenspeichern bestimmter Daten erreicht werden. Gleichzeitig bietet dieser Mechanismus eine Zeitersparnis bei der Datenverarbeitung, weil u. U. kein neuer Zugriff auf die Datenquelle über das Netz oder kein neues Verarbeiten des Templates notwendig ist.

Des Weiteren wird beim Entwurf der Lösung eine einfache Möglichkeit zur Parametrisierung von Templates vorgesehen. Das heißt neben der Datenquelle und dem

Template wird es einfach möglich sein, benutzerdefinierte Parameter an das Template zu übergeben. Diese werden innerhalb des Templates ausgewertet und können so unterschiedliches Verhalten abhängig vom Wert des Parameters ermöglichen. Damit wird die Flexibilität zum Einsatz eines bestimmten Templates deutlich erhöht. So muss z. B. für den Fall eines Templates in mehreren Sprachen lediglich der Parameter geändert werden. Es ist nicht notwendig ein von der Struktur identisches Template anzulegen.

Kapitel 5

Realisierung

Im letzten Kapitel wurde die Lösung der Problemstellung dieser Arbeit theoretisch beschrieben. Dieses Kapitel dokumentiert nun ihre praktische Umsetzung. Es wird mit den eingesetzten Technologien begonnen. Anschließend werden die entwickelten Komponenten der Lösung vorgestellt und erläutert. Es wird sowohl auf Architektur als auch auf API des Programmes, sowie auf Tests, Sicherheit und aufgetretene Probleme eingegangen.

Der Name der hier entwickelten Software ist LESS. Er steht für Leipzig Semantic Syndication. Vor Abschluss der Programmierung im Rahmen dieser Arbeit gab es keinerlei Veröffentlichung oder offizielle Versionierung der Komponenten, außer der implizierten im Versionierungssystem. Deshalb ist der Entwicklungsstand am Ende dieser Arbeit der erste veröffentlichte und bekommt die Versionsnummer `0.1`.

Es gibt eine frei zugängliche Testinstallation von LESS, die unter `http://less.aksw.org` erreichbar ist.

Auf Grundlage dieser Arbeit ist Paper entstanden, welches in Anhang A zu finden ist. Der Anhang B enthält die Hilfeseite, welche ebenfalls im Rahmen der Entwicklung von LESS entworfen wurde.

5.1 Eingesetzte Technologien

Aufgrund der großen Verbreitung im Bereich der Webapplikationen fiel die Wahl bei der Programmiersprache auf PHP[1], die in Version 5.2.10 verwendet wird, sowie der Zend-Bibliothek[2] in der Version 1.9.2. Als Templateengine wird Smarty[3] Version 3Alpha einge-

[1] http://php.net
[2] http://www.zend.com/de/community/framework
[3] http://www.smarty.net/

setzt. Smarty ist die am weitesten verbreitete Templateengine für die Programmiersprache PHP. Das Smartyprojekt ist Open Source, es gibt eine aktive Entwicklergemeinschaft und das Projekt wird ständig verbessert. Die Bibliothek Erfurt[4], Revision 388e755b42, zum Zugriff und zur Verarbeitung von RDF-Daten wurde ausgewählt, da sie ebenfalls von der Forschungsgruppe *AKSW* entwickelt wird. Als Rahmenwerk für das Testen ist PHPUnit[5] in Version 3.4.2 ausgewählt worden. PHPUnit ist ebenfalls Open Source und wird aktiv weiterentwickelt. Außerdem gibt es eine Integration für das Testen von Zend Komponenten. Als Datenbank kommt MySQL in der Version 5.1.37 zum Einsatz.

Der Programmquelltext wurde mithilfe von *git* in Version 1.6.3.3 verwaltet.

5.2 Entwickelte Templatesprache

Mit Smarty liegt eine sehr umfangreiche, bereits lange erprobte Templatesprache vor. Die eingesetzten Funktionen sind für die Verarbeitung von Daten gut geeignet und die Mächtigkeit der Templatesprache ist sehr groß. Die grundlegenden Anforderungen an die Sprache zur Lösung der Aufgabenstellungen dieser Arbeit werden von Smarty erfüllt. Allerdings gibt es in Smarty keinerlei Unterstützung für den Zugriff auf RDF-Daten. Die Entscheidung fiel deshalb auf eine Erweiterung der Smartytemplatesprache um die notwendigen Konstrukte für den Datenzugriff. Die Neuentwicklung einer Templatesprache hätte einen weitaus größeren Aufwand bedeutet als eine Erweiterung von Smarty.

In Smarty werden neue Platzhalter eingeführt, die den besonderen Ansprüchen der Adressierung von Ressourcen in Linked-Data-Quellen genügen. Darüber hinaus wird ein Mechanismus geschaffen, um Templates mit den speziellen Platzhaltern in korrekte Smartysyntax zu überführen, um schließlich Smarty für das Rendern der Templates verwenden zu können. Siehe dazu Kapitel 5.3.3.

Die Templatesprache wird als LESS Template Language oder LeTL bezeichnet.

Zunächst wird nun die Datenadressierung für die unterschiedlichen Datenquellen erklärt. Danach wird auf die aus Smarty verwendeten Sprachkonstrukte eingegangen. Abschließend werden zusätzlich eingeführte Funktionen und Parameter erläutert.

[4] http://aksw.org/Projects/Erfurt
[5] http://phpun.it/

5.2.1 Datenadressierung

Die hier entwickelten Platzhalter sollen eine möglichst hohe Lesbarkeit und Wiedererkennbarkeit für die Anwender bieten. Die Form der Platzhalter orientiert sich an unterschiedlichen Serialisierungen von RDF-Ressourcen. Für die beiden Arten von Datenquellen ist die Behandlung der Templatesprache unterschiedlich, weshalb sie im Folgenden getrennt beschrieben werden.

Linked Data

Für die Verdeutlichung der Funktionalität der LESS-Templatesprache ist es hilfreich, sich die Grundstruktur einer Aussage in RDF, das Tripel, noch einmal in Erinnerung zu rufen. Es besteht aus den drei Teilen Subjekt, Prädikat und Objekt. In einer Datenquelle sind eine beliebige Anzahl dieser Tripel enthalten. Im Template erreichbar sind alle Tripel deren Subjekt mit der URI der Datenquelle übereinstimmt. Etwas anders verhält sich die Applikation bei Verwendung des Parameters `instances` (siehe Abschnitt 5.5), folgt allerdings immer noch der prinzipiellen Funktionsweise. Das Objekt eines Triples enthält den eigentlichen Wert der jeweiligen Aussage. Diese Information soll im Template erscheinen. Um nun ein Objekt, oder eine Menge von Objekten, in einem Template eindeutig adressieren zu können, muss das zugehörige Prädikat festgelegt werden. Die Templatesprache sollte es also ermöglichen Prädikate in Platzhaltern so zu kodieren, dass der Wert (Objekt) für dieses Prädikat eingesetzt werden kann. Mit der Datenquelle (Subjekt) und dem Platzhalter (Prädikat) kann die Templateengine das zugehörige Objekt, oder die Objekte, bestimmen. Aufgabe der Templatesprache ist es nun, einen Weg zu schaffen, innerhalb von Quelltext, der ausgegeben werden soll, Platzhalter so zu integrieren, dass sie von der Templateengine erkannt und durch deren Inhalt ersetzt, jedoch nicht selbst angezeigt werden.

Im Fall von Linked-Data-Templates müssen URIs von Prädikaten in den Text eingefügt werden. Um diese vom restlichen Text abzugrenzen, werden zwei geschweifte Klammern als Abgrenzungssymbole zu Beginn und zu Ende der URI verwendet. Ein Beispiel für einen solchen Platzhalter mit einer URI ist der folgende:

```
{{http://dbpedia.org/property/population}}.
```

Da zum Zwecke der Übersichtlichkeit in RDF die Definition von Namensräumen zur Verkürzung von URIs erlaubt ist, soll dies auch in der Templatesprache möglich sein. Nach Definition des Namensraumes

```
dbpprop => http://dbpedia.org/property/
```

ist auch die Schreibweise

```
{{dbpprop:population}}
```

gültig.

Die RDF-Spezifikation sieht eine explizite Auszeichnung der verwendeten Sprache für Literale vor. Ein bestimmtes Literal kann als in Englisch verfasst beschrieben werden. Zur Repräsentation dieser Information wurde hier eine Form in Anlehnung an die sehr intuitive Beschreibung aus der RDF-Serialisierung Turtle[6] verwendet. Soll ein Textliteral mit einem Sprachschlagwort versehen werden, wird das Sprachkürzel mit einem vorgefügten „@“-Zeichen an das Literal angehangen. Möchte man also den Kommentar zu einer bestimmten Ressource auf dänisch erhalten, kann folgender Platzhalter angewendet werden:

```
{{rdf:comment@da}}.
```

Neben der Sprache kann für ein Literal in RDF auch der genaue Datentyp festgelegt werden. Auch dafür wird die Notation der Turtle-Spezifikation verwendet. Literal und Datentyp, der auch durch eine URI repräsentiert wird, werden durch zwei „^“-Zeichen verbunden. Möchte man das Geburtsdatum einer Person im Datumsformat laut XML-Schema[7] erhalten, verwendet man den Platzhalter

```
{{dbpedia-owl:birthDate^^xsd:date}}.
```

SPARQL-Datenquelle

Für den Fall eines SPARQL-Endpunktes als Datenquelle ist die Adressierung der Ergebnisdaten etwas einfacher. In einer SPARQL-Abfrage werden Variablen definiert. Werden diese im `SELECT`-Teil der Abfrage verwendet, so sind sie als Namen für Ergebnisspalten im Resultat der Abfrage enthalten. Für diesen Fall ist lediglich eine Adressierung der einzelnen Spalten der Ergebnistabelle notwendig. Für die Beispielabfrage

```
SELECT DISTINCT ?city ?name ?website ?inhabitants
WHERE {
 ?city <skos:subject> <dbpedia:Category:German_state_capitals> .
 ?city <rdfs:label> ?name .
 ?city <dbpprop:website> ?website.
```

[6] http://www.w3.org/TeamSubmission/turtle/
[7] http://www.w3.org/XML/Schema

```
 ?city <dbpprop:einwohner> ?inhabitants .
 FILTER ( lang(?name) = "de" )
}
```

ergibt sich die Ergebnistabelle in Abbildung 5.1.

SPARQL results:

city	name	website	inhabitants
:Magdeburg	"Magdeburg"@de	<http://www.magdeburg.de/>	230140
:Stuttgart	"Stuttgart"@de	<http://www.stuttgart.de/>	600038
:Wiesbaden	"Wiesbaden"@de	<http://www.wiesbaden.de/>	275422
:D%C3%BCsseldorf	"Düsseldorf"@de	<http://www.duesseldorf.de/>	582222
:Kiel	"Kiel"@de	<http://www.kiel.de/>	236902
:Mainz	"Mainz"@de	<http://www.mainz.de/>	200234
:Saarbr%C3%BCcken	"Saarbrücken"@de	<http://www.saarbruecken.de/>	180515
:Bremen	"Bremen"@de	<http://www.bremen.de/sixcms/detail.php>	548477
:Erfurt	"Erfurt"@de	<http://www.erfurt.de>	202929
:Munich	"München"@de	<http://www.muenchen.de/>	1356594
:Dresden	"Dresden"@de	<http://www.dresden.de/>	512234
:Hanover	"Hannover"@de	<http://www.hannover.de/>	522944
:Potsdam	"Potsdam"@de	<http://www.potsdam.de/>	151725

Bild 5.1: Ergebnistabelle für eine Sparql-Anfrage

Um nun die Ergebnisse der einzelnen Spalten zu erhalten, kann einfach der Spaltenname als normale Smartyvariable der Form `{$inhabitants}` verwendet werden. Besonders in Funktionsköpfen ist die Verwendung der Variable in nativer Smartysyntax erlaubt. Es ist keine zusätzliche Syntax notwendig.

Die Resultattabelle kann abhängig von der SPARQL-Anfrage keine, eine oder mehrere Zeilen enthalten. Der Benutzer hat innerhalb des Templates keinen Zugriff auf die einzelnen Zeilen des Ergebnisses. Vielmehr wird das Template für jede Ergebniszeile erneut aufgerufen und die resultierenden Zeichenketten werden verkettet zurückgegeben. Das heißt, es ist durch die Benutzer in der SPARQL-Anfrage festzulegen, ob ein oder mehrere Ergebniszeilen zurückgeliefert werden sollen.

5.2.2 Angepasste Smartyfunktionen

Häufiges Wiederverwenden der selben Zeichenkette erhöht den Aufwand ihrer Aktualisierung deutlich. Um den Aufwand und die Gefahr von Fehlern niedrig zu halten, werden Variablen eingesetzt. Smarty bietet dazu die Funktion `assign`. Auch diese kann in der LESS-Templatesprache verwendet werden. `assign` erlaubt es dem Nutzer eine bestimmte

Zeichenkette, oder auch den Inhalt aus einer Datenquelle, einem Variablennamen zuzuweisen. Der Zugriff auf den Inhalt ist dann immer einfach durch Verwendung der Variablen möglich. Die Zuweisung eines bestimmten Inhalts, in diesem Fall eine URL, könnte zum Beispiel so aussehen:

```
{assign var="url" value="http://blog.aksw.org/2009/eswc10-in-use-track/"}.
```

Soll auf Daten aus einer Linked-Data-Datenquelle zugegriffen werden, kann dies durch einen Aufruf der folgenden Form geschehen:

```
{assign var="zip" value="dbpedia-owl:postalCode"|getRdfValue}.
```

Nach der Zuweisung eines Inhalts zu einer Variablen kann darauf in der Form `{$<name>}`, im Beispiel `{$url}`, zugegriffen werden. `getRdfValue()` ist der Name einer an Smarty registrierten Funktion, die mit dem davor stehenden Wert als Parameter aufgerufen wird. `getRdfValue()` und andere spezielle Funktionen werden im folgenden Abschnitt (5.2.4) näher erläutert. Neben den selber erstellten Variablen gibt es noch spezielle Variablen, die dem Benutzer zur Verfügung stehen. Zum einen sind dies die SPARQL-Ergebnis-Variablen. Zum anderen sind das so genannte benutzerdefinierte Parameter. Näheres dazu im Abschnitt 5.2.3.

Eine häufig benötigte Funktionalität in Templates ist die der bedingten Darstellung von Inhalten. Der in Smarty enthaltene `if`-Ausdruck wird in der LESS Templatesprache erlaubt. Ein solcher Ausdruck enthält eine Bedingung und einen Inhalt.

```
{if <Bedingung>} <Inhalt> {/if}
```

Der Inhalt wird nur in das Template integriert, wenn die Bedingung wahr ist. Für LeTL wurde die Smartysyntax so erweitert, dass auch die hier eingeführten Platzhalter in der Bedingung verwendet werden können. Ein wichtiges und im Templatebereich häufig vorkommendes Beispiel für die Verwendung dieses Ausdrucks ist die Anzeige einer Tabellenzeile nur dann zuzulassen, wenn für den verwendeten Platzhalter tatsächlich Daten vorhanden sind. Im folgenden Beispiel würde die Zeile mit dem Namen nur angezeigt, wenn für diese Eigenschaft ein Wert vorliegt.

```
{if {{foaf:name}} != ''}
 <tr>
  <td>
    Name:
  </td>
```

```
  <td>
    {{foaf:name}}
  </td>
 </tr>
{/if}
```

Wie bereits erwähnt, können mit Subjekt und Prädikat ein Objekt oder eine Menge von Objekten adressiert werden. Zu einem bestimmten Subjekt und einem Prädikat können beliebig viele Objekte existieren. In *LeTL* wird ein Platzhalter immer zu einem einzelnen Objekt ausgewertet. Sollten mehrere vorhanden sein, wird lediglich zufällig einer der Werte zurückgegeben. Es gibt allerdings auch eine Möglichkeit über die Menge der Objekte zu iterieren und somit auf jedes einzelne Zugriff zu erhalten. Dies geschieht mit der `foreach`-Schleife in Smarty. Sie besteht aus einem Schleifenkopf und einem Schleifenkörper.

```
{foreach <Menge> as <Schleifenvariable>} <Schleifenkörper> {/foreach}
```

Im Schleifenkopf wird die Objektmenge definiert, über die iteriert werden soll. Auch in diesem Fall wurde die Syntax so angepasst, dass Linked-Data-Platzhalter eingesetzt werden können. Die Schleifenvariable gibt den Namen der Variable an, welche im Schleifenkörper in jedem Durchlauf den Wert des aktuellen Objekts enthält. Um nun zum Beispiel die Menge aller RDF-Klassen einer Ressource in einer Liste auszugeben, kann folgendes Beispiel verwendet werden.

```
<ul>
 {foreach {{rdf:type}} as $var}
   <li>{$var}</li>
 {/foreach}
</ul>
```

5.2.3 Benutzerdefinierte Parameter

Um der Anforderung an die Templates zu genügen, den Inhalt flexibel an unterschiedliche Parameter anzupassen, werden so genannte benutzerdefinierte Parameter eingeführt. Damit sollte es möglich sein, ein Template in unterschiedlichen Einsatzszenarien wiederzuverwenden, wenn die Anforderung lediglich einen geringen Unterschied in der Darstellung des Inhalts verlangt. Ein Benutzerparameter ist unter seinem Namen als Variable registriert.

Soll ein Template zum Beispiel in einer Webseite und auf der Kommandozeile ausgegeben werden, müssen die Zeilenumbrüche unterschiedlich codiert werden. Mithilfe einer

bedingten Anweisung kann das Problem bei Übergabe eines Parameters, der die verlangte Umgebung codiert, in einem Template gelöst werden. Für dieses Beispiel würde dem Template der benutzerdefinierte Parameter `lineBreak` übergeben. In ihm ist entweder durch den Wert *console* ein Zeilenumbruch für die Kommandozeile, oder durch den Wert *web* ein Zeilenumbruch für eine Webseite verlangt. Das zugehörige Template könnte dann wie folgt aussehen.

```
{if $lineBreak == 'web'}
  {assign var="line_break" value="<br>"}
{else}
  {assign var="line_break" value="\n"}
{/if}

<Daten> {$line_break}
<mehr Daten> {$line_break}
```

5.2.4 Spezielle Funktionen

Um einen höheren Grad an Modularisierung zu erreichen, ist es in einem Template möglich ein anderes Template einzubinden. Natürlich wäre dies über Verwendung von Javascript und einem Aufruf des *Template Processors* möglich, doch die hier eingeführte Funktion verringert den Aufwand deutlich. Durch den Aufruf der an Smarty registrierten Funktion `template()` kann unter Angabe der benötigten Parameter direkt das Ergebnis eines anderen Templates integriert werden. Zwei Beispiele für Aufrufe dafür könnten wie folgt aussehen:

```
{template id="3"
          uri="http://dbpedia.org/resource/Smarty"}
```

oder für eine SPARQL-Datenquelle

```
{template id="7"
          sparql="SELECT DISTINCT ?city ?name ?website ?lat ?long ?inhabitants
            WHERE {
              ?city <http://www.w3.org/2004/02/skos/core#subject>
                  <http://dbpedia.org/resource/Category:German_state_capitals> .
              ?city <http://www.w3.org/2000/01/rdf-schema#label> ?name .
              ?city <http://www.w3.org/2003/01/geo/wgs84_pos#lat> ?lat .
```

```
        ?city <http://www.w3.org/2003/01/geo/wgs84_pos#long> ?long .
        ?city <http://dbpedia.org/property/einwohner> ?inhabitants .
        FILTER ( lang(?name) = 'de') }"
    endpoint ="http://dbpedia.org/sparql" }.
```

Zwei weitere spezielle Funktionen stehen dem Benutzer zur Verfügung. `getRdfValue()` und `getRdfArray()` sind ursprünglich für den internen Gebrauch bei der Umwandlung der speziellen LeTL-Syntax in Smartysyntax entwickelt worden. Sie können allerdings auch direkt im Template aufgerufen werden, sollte dies gewünscht sein. Die Funktionen ermöglichen den direkten Zugriff auf Inhalte von RDF-Properties aus einer Linked-Data-Datenquelle. Die beiden Funktionen unterscheiden sich ausschließlich im Datentyp des Ergebnisses. Während `getRdfValue()` genau eine Zeichenkette zurück gibt, ist das Ergebnis bei `getRdfArray()` eine Kollektion von gefundenen Werten. Ein relevanter Fall für den direkten Aufruf könnte dieses Beispiel aus dem vorigen Kapitel sein:

```
{assign var="zip" value="dbpedia-owl:postalCode"|getRdfValue}.
```

5.3 Anwendungskomponenten

In der vorliegenden Beispielimplementierung sind die drei Hauptkomponenten eng miteinander verwoben. In diesem Fall wird aus dem *Template Repository* direkt auf den *Template Builder* zugegriffen. Außerdem ist für Vorschauzwecke ein *Template Processor* integriert.

Für andere Anwendungen ist es jedoch denkbar und u. U. sogar notwendig die Komponenten systematisch verteilt einzusetzen. Ein Beispiel dafür wird im Ausblick (Kapitel 8) bei der Verwendung in der Software Ontowiki [HDMA09] vorgeschlagen.

Die Hauptkomponenten werden im Folgenden einzeln erläutert.

5.3.1 Template Repository

Aufgabe dieses Teils der Anwendung ist die Verwaltung der bestehenden Templates. Nutzer haben die Möglichkeit hier Templates zu veröffentlichen, zu suchen und anzuschauen. Zentral für das *Template Repository* ist die Seite, die alle verfügbaren Templates auflistet. Sie ist in Abbildung 5.2 zu sehen.

Ein Überblick über jedes einzelne Template wird in einem extra Block dargestellt. Zur Darstellung gehört der Name, die Version und der Name des Autors. Als Zusatzinforma-

Bild 5.2: Template Repository

tionen sind die Schlagworte und die Anzahl der Aufrufe mit vermerkt. Darüber hinaus ist die Durchschnittsbewertung des Templates durch die anderen Benutzer sowie eine Liste von Kommentaren sichtbar. Außerdem zeigt ein Vorschaufenster das fertig gerenderte Ergebnis. Zu jedem Eintrag wird ein direkter Link zum Anschauen des Templates im *Template Builder* angeboten. Dadurch ist es möglich sich die genauen Details anzusehen. Ist der Benutzer Besitzer des Templates kann er es verändern. Ist er es nicht, kann er einfach eine Kopie davon anlegen.

Der Benutzer hat verschiedene Möglichkeiten, seine Suche in den Templates zu verfeinern. Dies kann mithilfe von frei vergebenen Schlagworten (Tags) geschehen, um bestimmte Templates herauszufiltern. Dabei ist es möglich verschiedene Schlagworte aus der Schlagwortwolke zu kombinieren, um die Anzahl der angezeigten Templates auf eine überschaubare Menge zu reduzieren. Eine zweite Möglichkeit der Einschränkung bietet die RDF-Klassenwolke. Dabei können analog der Schlagworte RDF-Klassen kombiniert

werden. Diese gelten jedoch nur für die Templates, die eine Linked-Data-URI als Datenquelle angegeben haben. Für SPARQL-Datenquellen kann keine Ressource eindeutig definiert werden, deren Klassenzugehörigkeit als Merkmal zu verwenden wäre. Eine dritte Möglichkeit der Einschränkung bei der Suche bietet das Auswahlkriterium „Only my templates“. Dabei werden nur Templates angezeigt, die vom aktuellen Benutzer angelegt wurden.

Darüber hinaus ist noch eine unterschiedliche Sortierung der Templates möglich. Sie können alphabetisch nach Namen sortiert werden. Es gibt auch die Möglichkeit die Templates nach Anzahl ihrer Abrufe in einem bestimmten Zeitraum zu sortieren. Des Weiteren ist eine Sortierung nach Bewertung durch andere Nutzer möglich. Die beiden letzten Sortierungskriterien sollen eine benutzergenerierte Dynamik in die Applikation bringen. Aus diesem Grund ist die Standardsortierung auch nach Benutzerbewertung eingestellt. Ein Template, welches häufig positiv von anderen Benutzern bewertet oder häufig verwendet wird, sollte also eine möglichst hohe Sichtbarkeit bekommen.

Templates können außerdem kommentiert werden. Dies ist ein Mechanismus um z. B. sehr schlechte oder sehr gute Bewertungen zu erklären bzw. Veränderungswünsche oder Verbesserungsvorschläge auszutauschen.

5.3.2 Template Builder

Durch das Klicken auf den im *Template Repository* beschriebenen Link gelangt der Benutzer zum *Template Builder* der in Abbildung 5.3 zu sehen ist. Hier kann nun der konkrete Quelltext eines Templates angeschaut und ggf. geändert werden. Auch wenn der Benutzer nicht angemeldet ist, gelangt er in diesen Bereich, um sich einen genauen Eindruck vom Inhalt des Templates verschaffen zu können. Ist der Benutzer angemeldet und Besitzer des Templates, kann er die Daten und Eigenschaften des Templates verändern und speichern.

Der erste Block im *Template Builder* bietet die Möglichkeit, die Datenquelle für das Template festzulegen. Dabei gibt es grundsätzlich zwei Möglichkeiten. Die Daten können entweder aus einer Linked Data-Ressource stammen oder von einer SPARQL-Abfrage an einen SPARQL-Endpunkt. Im Fall der Linked-Data-Quelle muss die URI der gewünschten Ressource angegeben werden (siehe Bild 5.4). Im Fall von SPARQL müssen sowohl die URL des Endpunktes als auch die Anfrage angegeben werden (siehe dazu Bild 5.5). Es ist während der Arbeit an einem Template jederzeit möglich die Datenquelle zu wechseln.

Beim Klick auf „Update Properties“ wird die angegebene Datenquelle abgefragt. Im Linked-Data-Fall wird versucht die RDF-Daten, welche bei Dereferenzierung der URI an-

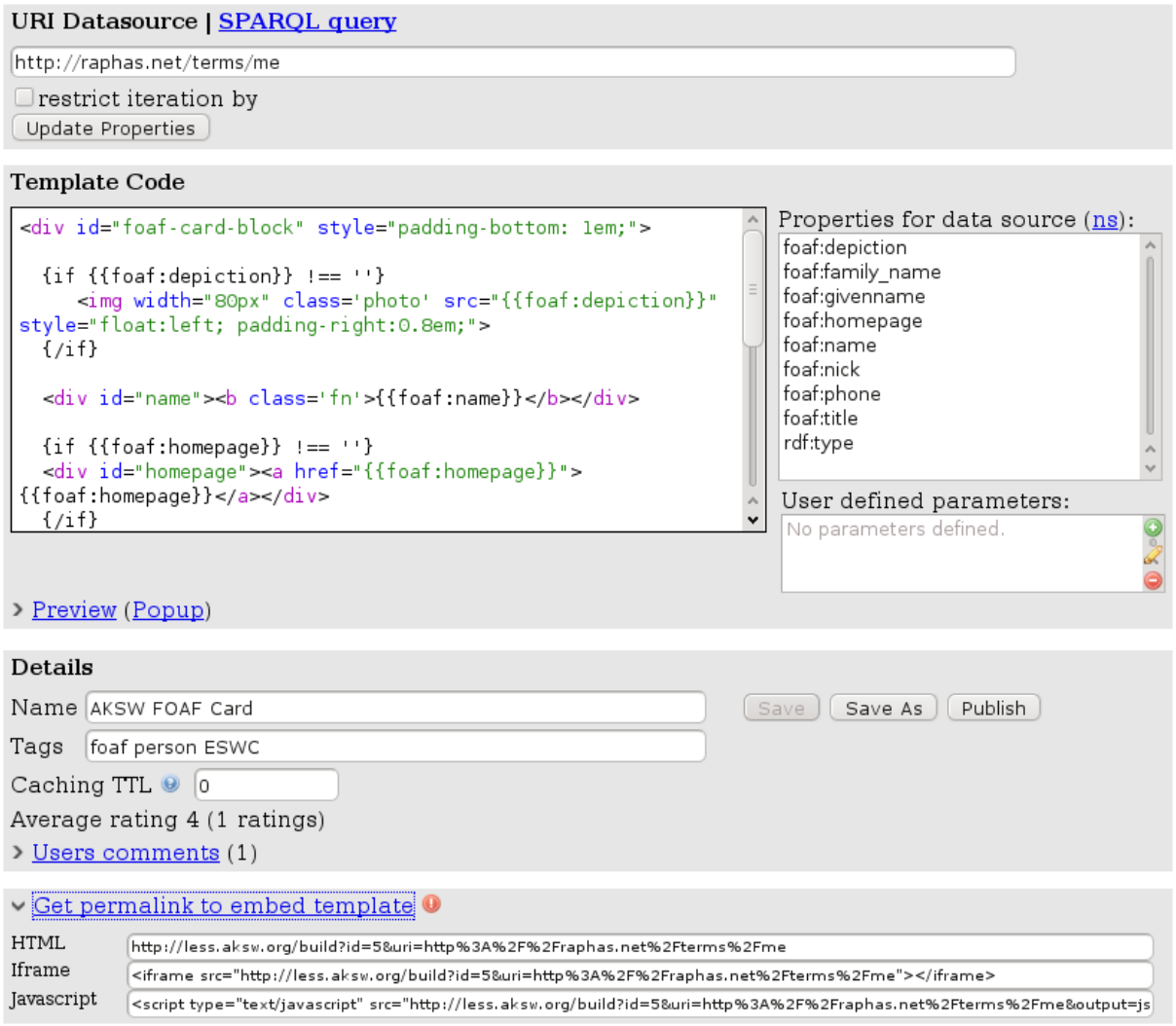


Bild 5.3: Template Builder

geboten werden, herunterzuladen. Ist das Caching in der Anwendung aktiviert, wird der Cache vorher überprüft, ob er eine gültige Version der Daten vorhält. Sollte das der Fall sein, werden diese verwendet. Die lokal gespeicherten RDF-Daten werden danach durchsucht. Es werden solche Tripel gefunden, die direkte Aussagen über die durch die URI adressierte Ressource machen. Für den Benutzer werden nun die Prädikate dieser Tripel extrahiert. Dabei wird unterschieden, wieviele Objekte für ein Prädikat gefunden werden. Außerdem werden die Objekte nach eventuell vergebenen Sprachtags sowie Datentypen sortiert. Am Ende entsteht eine Liste von Prädikatnamen mit eventuellen Sprach- oder Datentyperweiterungen, welche dem Benutzer in der so genannten „Properties Recommender“ angezeigt werden. Die Struktur dieser Platzhalter wird ausführlich im Kapitel 5.2 beschrieben. Sie repräsentieren die in der Datenquelle verfügbaren Attribute für die ausgewählte Ressource. Beim Schweben mit dem Mauszeiger über einer bestimmten Zeile in der Auswahlbox werden die in der Datenquelle gefundenen Objektwerte in einer kleinen Informationsblase (tooltip) angezeigt.

Im SPARQL-Fall wird die Anfrage an den SPARQL-Endpunkt abgesetzt. Das Ergebnis wird im Speicher gehalten. Die Variablen aus der SPARQL-Anfrage werden dem Benutzer in der „Properties Recommender“ angeboten. Auch in diesem Fall erhält man durch Verweilen mit dem Mauszeiger über einer Ergebniszeile eine Vorschau auf die für diese Variable gefundenen Werte. Durch Klick auf eine der Zeilen wird nun der ausgewählte Platzhalter in den links daneben befindlichen Editorbereich an der aktuellen Position der Schreibmarke eingefügt.

Der Editorbereich erlaubt die Bearbeitung des Quelltextes. Er kann HTML-Text farblich hervorheben. Der Benutzer wird zum Beispiel durch besondere Färbung auf nicht geschlossene HTML-Elemente im Text hingewiesen. Bei der Erstellung von HTML-Templates und der Fehlersuche in den selbigen ist diese Eigenschaft sehr hilfreich. Im Quelltext kann der Benutzer alle in der Templatesprache erlaubten Ausdrücke verwenden (siehe Kapitel 5.2). Sie werden in diesem Moment als reiner Text gespeichert und erst beim Rendern eines Templates überprüft sowie verwendet.

Die Auswahlbox mit dem Titel „User defined parameters“ beinhaltet benutzerdefinierte Zusatzparameter für das Template. Es ist an dieser Stelle möglich, einfache Schlüssel-

Bild 5.4: Angabe einer Linked-Data-Quelle

Bild 5.5: Angabe einer SPARQL-Quelle

Wert-Paare anzulegen und zu speichern. Der Schlüssel eines solchen Paares kann als Variable im Template verwendet werden. Dieser Mechanismus wird im Kapitel 5.2.3 ausführlich erläutert. Die Werte aus dieser Eingabe werden in dem Fall nur gespeichert, um eine Vorschau für das Template auch ohne konkret angegebene Konfigurationswerte zu ermöglichen. Theoretisch können im Templatequelltext beliebige Variablennamen verwendet werden, welche erst beim Rendern einen Wert erhalten.

Um eine effiziente Entwicklung von Templates zu ermöglichen Bedarf es einer Vorschaufunktion. Im Moment gibt es zwei Wege einer Vorschau: integriert in den *Template Editor* und als externes Popupfenster. Der Zugriff auf beide befindet sich in Form von Verweisen direkt unter dem Eingabebereich für den Templatecode.

Der nächste logische Block beinhaltet Zusatzinformationen zum Template. Hier befindet sich ein Eingabefeld, welches es erlaubt den Namen des Templates zu ändern. Im nächsten Eingabefeld können die mit diesem Template assoziierten Schlagworte getrennt durch Leerzeichen eingegeben werden. Außerdem sind hier die Knöpfe zum Speichern des Templates platziert. Sobald ein Benutzer am Editor angemeldet ist, hat er die Möglichkeit mit „SaveAs“ eine Kopie des Templates anzulegen. Gehört das Template dem Benutzer selber und er darf es editieren, erscheint auch ein Button mit der Aufschrift „Save“. Der letzte Button, „Publish“, dient der Veröffentlichung des Templates im *Repository*. Auch hier können ähnlich wie im *Template Browser* eine Bewertung und ein Kommentar abgegeben werden. Das letzte Eingabefeld in diesem Bereich widmet sich der Einstellung für die Gültigkeit der Daten im Zwischenspeicher der Applikation. Der Mechanismus wird ausführlich im Abschnitt 5.6 beschrieben.

Um ein Template letztendlich zu verwenden, muss der *Template-Processor*-Webservice mit einer bestimmten Menge an Parametern angefragt werden. Um die Handhabung des Templates zu vereinfachen, werden, sobald das Template gespeichert ist, drei Eingabefelder mit permantenten URLs ausgefüllt. Das erste Feld beinhaltet die URL, welche das Template als reines HTML-Ergebnis vom *Template Processor* anfordert. Alle Parameter sind so in der URL zu finden, wie sie in diesem Moment im *Template Builder* gesetzt

sind. Insbesondere die aktuelle Datenquelle sowie die beutzerdefinierten Parameter sind darin integriert. Zur direkten Einbettung eines Templates in eine bestehende HTML-Seite sind die beiden folgenden Felder geeignet. Das Erste bietet die oben beschriebene URL integriert in den HTML-Text einer iframe-Umgebung. Das Zweite verwendet Javascript und ermöglicht eine direkte Integration des vom Webservice gelieferten Ergebnisses in den Text einer Webseite.

5.3.3 Template Processor

Der Webservice zum Rendern von Templates ist unter der URL

```
http://<basis-url>/build
```

zu erreichen. Im Abschnitt 5.5 ist die genaue Definition der Parameter zu finden. Die Webservice-Klasse nimmt die Parameter entgegen und überprüft diese. Die eigentliche Geschäftslogik ist auch hier wieder in einer extra Klasse gekapselt. In diesem Fall gehört diese nicht zum Modell der Applikation, sondern ist als externes Modul anzusehen, da eine Installation der Applikation theoretisch auch ohne integrierten *Template Processor* denkbar ist.

Die Klasse *SmartyTemplateBuilder* wird mit einem Datenfeld, das die Template Parameter beinhaltet, instanziiert. Dabei werden die Parameter auf Vollständigkeit überprüft. Danach kann durch Aufruf der Funktion `getHtmlCode()` das Rendern des Templates erreicht werden.

Das eigentliche Rendern wird komplett in Smarty durchgeführt. Bevor das Rendern gestartet wird, werden allerdings als Erstes die Datenquellen angefragt und die Ergebnisdaten gespeichert. Damit Smarty Inhalte für Variablen ersetzen kann, müssen diese am Smartyobjekt registriert werden. In jedem Fall werden als Erstes eventuell übergebene, benutzerdefinierte Parameter als Variablen angemeldet. Im Falle einer SPARQL-Datenquelle werden anschließend noch die Variablen aus der SPARQL-Anfrage registriert. Für jedes Template wird noch die spezielle Funktion `template()` hinzugefügt, die eine vereinfachte Integration von Templates in ein anderes Template erlaubt (siehe Abschnitt 5.2.4). Danach wird das Rendern durch Smarty angestoßen. Das Ergebnis der Smarty-Umwandlung wird nun direkt als Ergebnis an den Webservice zurückgegeben. Nur im Falle eines Ausnahmefehlers beim Rendern in Smarty gibt es eine Fehlermeldung.

Bei Daten aus einer Linked Data-Quelle sind noch weitere vorbereitende Schritte notwendig. Hier geschieht die bereits erwähnte Zurückführung der *LeTL*-Syntax auf die von Smarty. Da die Template-Platzhalter für diese Applikation unter dem Gesichtspunkt der

Verständlichkeit für den Benutzer in Anlehnung an die Syntax von RDF entwickelt wurden, ist eine direkte Verwendung dieser Platzhalter in Smarty nicht möglich. Insbesondere die Zeichen „@“, „^“ sowie „:“ haben in Smarty eine spezielle Bedeutung. Smarty bietet eine einfache Möglichkeit für die Vorverarbeitung von Templatetext durch den Mechanismus der „Prefilter“. Am Smartyobjekt kann eine Funktion registriert werden, welche vor dem Rendern des Templates aufgerufen wird. Dabei wird der Text des jeweiligen Templates als Parameter übergeben. In der Funktion wird nun mithilfe von regulären Ausdrücken nach Vorkommen der speziellen Platzhalter gesucht. Jedes Auftreten wird durch den Aufruf einer speziellen Smartyfunktion ersetzt, welche den ursprünglichen Platzhalter als Parameter übergeben bekommt. Bei der Übergabe von Parametern als Zeichenkette wird der Inhalt durch Smarty nicht ausgewertet und die speziellen Zeichen stellen kein Problem dar. Aus

```
{{rdfs:comment@en}}
```

wird also

```
{getRdfValue placeholder="rdfs:comment@en"}.
```

Bei dieser Ersetzung werden noch zwei Sonderfälle beachtet. Die Syntax von Smarty erlaubt keine Aufrufe von Funktionen in Smartyfunktionsköpfen. Eine für die Verarbeitung sehr wichtige Funktion ist `foreach`. Sie ermöglicht es über die einzelnen Elemente einer Elementsammlung zu iterieren. Der Benutzer hat die Möglichkeit eine Eingabe in der intuitiven Form

```
{foreach {{rdf:type}} as $var}{/foreach}
```

vorzunehmen. Bei einfacher Ersetzung durch die oben beschriebene Regel würde Quelltext der Form

```
{foreach {getRdfValue placeholder="rdf:type"} as $var}{/foreach}
```

entstehen. Der hier abgebildete Aufruf der Funktion `getRdfValue()` im Schleifenkopf von `foreach()` ist nicht erlaubt. Aus diesem Grund wird als Erstes nach Vorkommen der Platzhalter innerhalb von `foreach`-Schleifenköpfen gesucht. Das erste Beispiel zur `foreach`-Funktion würde nach der Ersetzung in folgenden Code umgewandelt:

```
{assign var="__internal__rdfArray" value="rdf:type"|getRdfArray}
{foreach $__internal__rdfArray as $var}{/foreach}.
```

Hier wird die zu Smarty gehörende Zuweisungsfunktion `assign()` verwendet um eine temporäre Variable mit den Daten, welche durch den Platzhalter kodiert sind, zu füllen. Der Grund für eine gesonderte Behandlung der `foreach`-Funktion liegt in der Tatsache begründet, dass für `foreach()` kein einfacher Wert benötigt wird, sondern eine Kollektion von Werten. Für alle anderen Fälle, in denen lediglich ein einzelner Wert erwartet wird, wird also eine andere Codeersetzung vorgenommen. Dabei findet statt der Funtion `getRdfArray()` die bereits bekannte Funktion `getRdfValue()` Anwendung. Nachdem die Ersetzungen durch den Filter geschehen sind, werden die oben erwähnten speziellen Funktionen an Smarty registriert und das Rendern des Templates kann angestoßen werden.

Ein weiterer Sonderfall gilt für Templates, die eine Linked-Data-Datenquelle verwenden. Sind keine weiteren Parameter angegeben, wird das Template für die Ressource aufgerufen, deren URI als Datenquelle angegeben wurde. Dieses Verhalten kann durch zwei weitere Parameter beeinflusst werden. Da RDF-Quellen gelegentlich eine Menge von Ressourcen des selben Typs zusammenfassen ohne diese einer gemeinsamen Hauptressource zuzuordnen, muss eine Möglichkeit gefunden werden, um Zugriff auf diese Daten zu erhalten. Dies geschieht hier unter Verwendung des `instances`-Parameters. Er erwartet die URI einer RDF-Klasse als Wert. Wird der *Template Processor* mit solch einer RDF-Klasse aufgerufen, findet der Aufruf des Templates für alle Ressourcen aus der Datenquelle statt, welche zu der entsprechenden Klasse gehören. Somit ist es insbesondere möglich bestimmte Ressourcen aus einer Datenquelle zu selektieren, sollten verschiedene vorhanden sein. Da die Reihenfolge der Ressourcen in RDF variieren kann, wurde der Parameter `sortBy` implementiert, welcher als Wert ein Property des ausgewählten Ressourcentyps erwartet. Die Ressourcen werden dann vor der Verarbeitung nach den Werten eben jenes Properties sortiert.

5.4 Architektur der Anwendung

Das Model-View-Controller-Pattern (MVC) ist ein sehr verbreitetes Strukturmuster für die Architektur von Software. Vor allem im Bereich der Webanwendungen hat sich das Muster als Standard etabliert. MVC wurde das erste Mal 1979 von Reenskaug [Ree79] beschrieben. Für viele Programmiersprachen gibt es ein Rahmenwerk, das dieses Strukturmuster unterstützt. Für PHP bietet die Zend-Bibliothek[8] die Grundlagen für die Verwendung des MVC-Musters.

Die konsequente Trennung der Programmcodes in die Bereiche Präsentationsschicht (View), Geschäftslogik (Model) und der dazwischenliegenden Steuerung (Controller) führt

[8] http://www.zend.com/de/community/framework

zu einer klaren Strukturierung des Anwendungscodes. Eine solche Aufteilung der Software ermöglicht eine eindeutige Zuordnung von Zuständigkeiten und erleichtert somit die Fehlersuche während der Entwicklung. Außerdem wird eine Erweiterung der Applikation, sowie die Wiederverwendung von Komponenten ohne größeren Aufwand ermöglicht. Aus diesem Grund wurde das MVC-Muster für die Architektur der beiden Anwendungsmodule *Template Repository* und *Template Builder* ausgewählt. Da die Zend-Bibliothek bereits Bestandteil der Anwendung war und die volle MVC-Funktionalität unterstützt, ist die Wahl auf diese Implementierung gefallen.

Für die einzelnen Hauptaufgaben der Anwendung hat jeweils ein anderer Controller die Verantwortung. Für das Durchsuchen der Templates im *Template Repository* ist der *BrowseController* zuständig. Des Weiteren gibt es den *EditController* für den *Template Builder*. Für die Funktionalität des Editors sind im Hintergrund noch weitere Controller notwendig. Der *SaveController* kümmert sich um die Speicherung von Veränderungen, welche am Template vorgenommen wurden. Der *GetController* liefert verschiedene Daten als Webservice an die Anwendung zurück. Unter anderem stellt er die Liste von Properties für den „Properties Recommender“ im *Template Editor* zur Verfügung. Der *LoginController* und der *RegisterController* sind für die Aufgaben Benutzeranmeldung und Registrierung zuständig.

Die Zend-Bibliothek unterstützt das Prinzip der „convention over configuration“. Dabei werden bestimmte Standardvorgehensweisen für die Applikation erwartet. Solange man sich an diese hält, müssen viele Festlegungen für das Zusammenspiel der Komponenten nicht explizit konfiguriert werden. So liegen alle Controller zum Beispiel standardmäßig im Ordner

```
/application/controllers/.
```

Erst wenn man Controller in einem anderen Verzeichnis ablegen möchte, muss diese Position explizit angegeben werden. Dies hat den Vorteil, dass bei Verwendung der vorgeschriebenen Standards keinerlei Konfiguration notwendig ist. Die Erstellung des Grundgerüstes einer Applikation ist dadurch sehr schnell und unkompliziert möglich.

Ähnlich der Controller selbst sind die Definitionen der Darstellung für die einzelnen Controller im festen Ordner

```
/application/views/scripts/<Name des Controllers>
```

zu finden. Sie enthalten das HTML-Gerüst für die Darstellung der im jeweiligen Controller verarbeiteten Inhalte.

Die Geschäftslogik der Anwendung ist in den Modellklassen implementiert. Dabei sind nach klassischem objektorientierten Design die meisten Ressourcen als Klasse abgebildet, welche im Aufgabenbereich der Applikation eine wichtige Rolle spielen. So sind unter

```
/application/models/
```

zum Beispiel die Klassen *Template*, *User*, *Tag* und *RdfClass* zu finden. Sie enthalten die jeweils für das Objekt erwartete Funktionalität. Die Klasse *Template* zum Beispiel bietet Methoden zum Erhalten oder Setzen des Templatecodes oder zum Speichern eines Templates.

5.5 API

Zugriff auf die Daten und Dienste von LESS sind über zwei prinzipielle Wege möglich. Im folgenden werden die genauen Zugangsmodalitäten für beide ausführlich erläutert.

5.5.1 Webservices

Der *Template Processor* liefert auf Anfrage ein fertig gerendertes Template zurück. Der Webservice ist unter der URL

```
http://<basis-url>/build
```

zu erreichen, wobei „¡basis-url¿“ durch die URL der jeweiligen Installation ersetzt werden muss. Er verlangt mindestens die folgenden Parameter:

- *id* - ID des Templates

und Konfiguration der verwendeten Datenquelle. Wird eine Linked-Data-Datenquelle verwendet ist folgender Parameter ausreichend:

- *uri* - URI der Datenquelle.

Wird eine SPARQL-Datenquelle verwendet werden zwei Parameter erwartet:

- *sparql* - die SPARQL-Anfrage
- *endpoint* - der SPARQL-Endpunkt

Abgesehen von den verlangten Parametern können noch einige optionale Parameter hinzugefügt werden

- *output*
  - js - Template wird in Javascript-Code eingebettet ausgegeben
  - app - Das Ergebnis wird in einem JSON-Array zurückgeliefert, diese Option wird z. B. vom *Template Builder* eingesetzt.
  - Wird dieser Parameter weggelassen, wird das Template direkt als Text ausgegeben.
- *debug* - aktiviert die Ausgabe von eventuell vorhandenen Fehlermeldungen
- *instances* - beschränkt die Ausführung des Templates auf Ressourcen des angegebenen Typs
- *sortBy* - nur im Zusammenhang mit *instances* - sortiert die verwendeten Ressourcen nach dem Wert des angegebenen Properties

Des Weiteren gibt es den Webservice für externe Anwendungen, welche den Zugriff auf Templates erleichtern sollen, wie zum Beispiel die Typo3-Erweiterung (siehe Kapitel 6.2.2). Er ist unter

```
http://<basis-url>/service/<typ>
```

abrufbar. Es gibt zwei Typen für diesen Dienst:

- list - Dieser gibt die Liste von Templates dieser LESS-Instanz zurück. Er benötigt keine Parameter.
- template-versions - Dieser Service verlangt als Parameter die ID des Templates (`templateId`) um die für dieses Template vorhandenen Versionsnummern zurück zu liefern.

Ein weiterer Zugriffpunkt auf die Applikation ist der *GetController*. Er wird im Moment nur applikationsintern verwendet, ist aber bei einer verteilten Installation der Komponenten als Schnittstelle zum *Template Repository* unbedingt notwendig. Er liefert die Templatedaten zu einem bestimmten Template in Rohform aus. Zu erreichen ist dieser Dienst unter

```
http://<basis-url>/get/template
```

und erwartet ebenfalls eine Template-ID (`templateId`) sowie die Version (`revision`), wenn nicht die aktuellste gewünscht wird.

Alle Parameter für diese Dienste können als GET-Parameter direkt an die URL angehangen werden.

5.5.2 Linked Data

Alle Daten für LESS werden in einer MySQL-Datenbank gehalten. Die relationale Datenbank wird statt direkter Speicherung in RDF eingesetzt, da der Zugriff aus PHP mit Zend sehr einfach und effizient möglich ist. Die Datenbank ist nur für die Applikation erreichbar. Um nicht private Daten dennoch als Linked Data zu veröffentlichen, wird triplify eingesetzt, welches diese Aufgabe sehr unkompliziert löst [ADL+09].

Die Einstiegs-URI ist

`http://<basis-url>/triplify.`

Unter dieser URL sind Basisinformationen über die vorhandenen Objekte für diesen Endpunkt enthalten. Im Moment werden zwei Typen von Objekten publiziert. Templates und Kommentare können als Linked Data abgerufen werden. Die Templateobjekte enthalten Informationen wie Name des Erstellers und Schlagworte. Private Daten wie z. B. E-Mailadressen der Benutzer werden nicht veröffentlicht.

Mit der URL

`http://<basis-url>/triplify/template/`

erhält man eine Liste von existierenden Templates inklusive der jeweiligen URI. Eine Template-URI könnte dann so aussehen:

`http://<basis-url>/triplify/template/2.`

Ruft man diese URI auf, erhält man die RDF-Daten für das Template mit der ID '2' in N3-Serialisierung.

```
<http://less/triplify/template/2>
    <http://purl.org/dc/elements/1.1/title> "City Fact Box" .
<http://less/triplify/template/2>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://less/triplify/template> .
```

```
<http://less/triplify/template/2>
    <http://rdfs.org/sioc/ns#has_reply> <http://less/triplify/comment/24> .
<http://less/triplify/template/2>
    <http://www.holygoat.co.uk/owl/redwood/0.1/tags/taggedWithTag>
    "city" .
<http://less/triplify/template/2> <http://purl.org/dc/terms/modified>
    "2010-01-16T12:02:07"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<http://less/triplify/template/2> <template_code>
    "<table [...] </tr>\n</table>\n" .
<http://less/triplify/template/2> <http://rdfs.org/sioc/ns#has_creator>
    "admin" .
```

In diesem Ausschnitt sieht man wie die Templatedaten in RDF kodiert werden. Das erste Tripel beschreibt den Namen des Templates. Das zweite definiert, dass die beschriebene Ressource vom Typ Template ist. Das folgende Property `<http://rdfs.org/sioc/ns#has_reply>` verweist auf eine weitere URI unter welcher ein Kommentar zum aktuellen Template zu erreichen ist. Die weiteren Tripel beschreiben ein Schlagwort, den Modifikationszeitpunkt, den Templatecode sowie den Namen des Autors.

5.6 Wichtige Entwurfsentscheidungen

Die Wiederverwendung von bestehenden Templates ist eines der Grundanliegen bei der Entwicklung von LESS. Dafür wurde dem Benutzer eine einfache Möglichkeit gegeben bereits bestehende Templates als Grundlage für seine eigene Arbeit zu verwenden. Bei Betrachtung jedes veröffentlichten Templates im *Template Builder* kann mithilfe des „Save As“-Knopfes der aktuell betrachtete Templatecode als neues Template gespeichert werden, welches dann dem Benutzer gehört und verändert werden kann.

Ein Ziel der Applikation ist es, vielen unterschiedlichen Benutzern die Möglichkeit zu geben eigene Templates anzulegen und bestehende Templates wiederzuverwenden. Dazu ist es im Besonderen keinem Nutzer möglich, bestehende Informationen eines anderen Benutzers zu löschen oder zu überschreiben. Das heißt, jeder Benutzer hat Lesezugriff auf den Quelltext von veröffentlichten Templates. Allerdings kann dieser nur vom Besitzer selbst verändert werden.

Die hohe Zuverlässigkeit bezüglich des erwarteten Templateinhaltes ist eine weitere wichtige Eigenschaft der Applikation. Dieser Anforderung wird dergestalt Rechnung getragen, dass ein Template, welches veröffentlicht wurde in dieser Version auch vom Besitzer nicht

mehr verändert werden kann. Der Besitzer kann selbstverständlich aus der veröffentlichten Version eine neue private Version generieren, die er weiter verändern kann. Dieser Mechanismus garantiert Benutzern, die fremde Templates verwenden, dass sich eine einmal verwendete Version eines Templates nicht mehr verändern kann. Bei allen Zugriffsvarianten auf Templates ist es einfach möglich, durch Weglassen der Versionparameters, die aktuellste Version zu erhalten.

Dem Prinzip der bestmöglichen Informationserhaltung folgend, wird bei einer Transformation des Templates in einen anderen Zeichensatz die `translit`-Methode angewandt. In diesem Fall werden Zeichen, welche im Zielzeichensatz nicht vorhanden sind möglichst informationserhaltend ersetzt. Beispielsweise wird das „`€`“-Zeichen bei einer Umwandlung von *UTF-8* in *ISO-8559-1* durch „`EUR`“ ersetzt.

Nicht gefundene Variablen werden im Template einfach übergangen. Es soll vermieden werden, dass die Auslieferung eines Templates fehlschlägt, weil eine Variable eventuell nicht gefüllt werden kann.

Nur im Falle einer schwerwiegenden Ausnahme schlägt das Rendern des Templates fehl und es wird ein leeres Template ausgeliefert. Dies passiert beispielsweise, wenn eine Datenquelle nicht erreichbar ist und sich auch kein Datensatz im Zwischenspeicher befindet. Normalerweise werden keine Fehlermeldungen angezeigt um unerwarteten Inhalt zu vermeiden. Es ist immer möglich durch Hinzufügen des *debug*-Parameters die aufgetretenen Fehlermeldungen zu erhalten.

5.7 Caching

In LESS ist ein Zwischenspeicher (Cache) implementiert, den sich die drei Hauptkomponenten der Applikation teilen. In diesem werden RDF-Daten und SPARQL-Ergebnisse, die in einer der Komponenten verwendet wurden, zwischengespeichert. Zum einen wird dadurch die Netzlast an der Datenquelle massiv verringert. Zum anderen besteht die Möglichkeit im Falle der Nichterreichbarkeit der Datenquelle ein Template dennoch auszuliefern. Für den Benutzer passiert all das völlig transparent.

Eine Anfrage an ein Linked-Data-Dokument wird von der zentralen Helfer-Klasse *RdfDownloadHandler* in der Applikation verarbeitet. Dadurch kann ein gemeinsamer Cache realisiert werden. Wird ein bestimmtes Dokument, welches durch seine URI eindeutig adressiert werden kann, angefragt, geschieht als Erstes eine Überprüfung des Caches. Wird ein passender Datensatz zu dieser URI gefunden, wird die Aktualität überprüft. Ist der Datensatz noch aktuell (cache hit), wird er direkt verwendet, ohne dass Netzwerk-

verkehr erzeugt werden muss. Sollte die Lebenszeit überschritten sein, wird ein negatives Ergebnis der Anfrage (cache miss) zurückgegeben. In diesem Fall wird versucht die Daten direkt, unter Verwendung der URI, von der Datenquelle aus dem Netz zu erhalten. Sollte dies fehlschlagen, wird der Cache ein zweites Mal überprüft. In diesem Fall wird auch Inhalt mit abgelaufener Lebenszeit akzeptiert um eine Anzeige der gewünschten Daten zu ermöglichen.

Für SPARQL-Datenquellen wird ein analoger Mechanismus angewandt.

Die implementierte Lösung erhöht die Stabilität des angebotenen Templatedienstes deutlich. Ist ein Datensatz einmal im Cache der Applikation gespeichert, können Templates die diese Daten verwenden mit hoher Sicherheit ausgeliefert werden.

5.8 Unit Tests

Neben manueller Tests wurde für die Applikation bereits während der Entwicklung eine Suite von Unit Tests für die meisten der programmierten Klassen angelegt und gepflegt. Dabei ist aktuell eine Test-Abdeckung von 70 % der Codezeilen erreicht. Ein besonderes Augenmerk lag dabei auf der Entwicklung der Tests für die Modell- sowie die Helperklassen, da diese die Geschäftslogik enthalten.

5.9 Während der Entwicklung erkannte Probleme

Verschiedene Probleme kristallisierten sich während der Entwicklung der Applikation heraus, die mit den Details der Verarbeitung von semantischen Daten in der Praxis zu tun haben. Teilweise konnten die Probleme direkt gelöst werden, wie im Folgenden beschrieben wird.

So vielfältig die Datenquellen für RDF-Inhalte im Internet sind, so unterschiedlich ist auch die Qualität und Vollständigkeit der dort angebotenen Daten. Bei Tests mit dem Projekt DBpedia ist es häufig vorgekommen, dass bestimmte Properties nur bei einer gewissen Instanz der Klasse vorhanden waren. So z. B. das `foaf:depiction`-Property bei Städten. Mit dieser Form von Datenunsicherheit kann mithilfe der bedingten Anweisungen, welche bereits implementiert sind, umgegangen werden. Bei vollständiger Überprüfung jedes einzelnen der verwendeten Platzhalter ist der Aufwand jedoch sehr hoch. Außerdem kommt es zu häufiger Wiederholung immer der gleichen Konstrukte. Dieses Problem soll in einer der nächsten Versionen der Anwendung adressiert werden.

Ein häufig angetroffenes Phänomen und verwandt mit dem eben besprochenen, ist die

Vielfältigkeit der Möglichkeiten ein bestimmtes Prädikat für eine Ressource zu benennen. Häufig sind verschiedene Bezeichner aus unterschiedlichen Ontologien synonym oder können als solche verwendet werden. Dies führt allerdings auch dazu, dass bei Verwendung des einen Bezeichners in einem Template, das u. U. alternative Vorkommen eines anderen Bezeichners nicht gefunden würde. Dieses Problem ist mit den Mitteln der aktuellen Applikation durchaus lösbar. Ein erster Ansatz, wenngleich wieder mit hohem Aufwand verbunden, ist die Verwendung der durch Smarty implementierten bedingten Anweisungen *if,else*. Durch diese ist es möglich das Vorhandensein bestimmter Inhalte zu überprüfen und in Abhängigkeit des Ergebnisses anzuzeigen. Eine zukünftige Lösung um den Bearbeitungsaufwand für den Benutzer gering zu halten, könnten Äquivalenzklassen bestimmter Properties sein. Es wäre dann möglich die Elemente einer solchen Äquivalenzklasse zu definieren. Danach könnte eine Funktion angeboten werden, welche, wenn gefunden, den Wert eines der Elemente zurück gibt.

5.10 Sicherheitsaspekte der Implementierung

Im Sicherheitskonzept gibt es eine klare Trennung von Zugriffsrechten auf Templates unterschiedlicher Benutzer. So ist es jedem Benutzer möglich alle veröffentlichten Templates zu lesen und zu kopieren. Eine Veränderung oder gar Löschung der Templates ist jedoch nicht möglich.

Smarty bietet eine große Flexibilität in der Benutzung von Funktionen. Dies kann u. U. zu einem Sicherheitsproblem werden, wenn der Benutzer z. B. Funktionen in PHP aus dem Template aufrufen kann und somit Zugriff auf die Laufzeitumgebung der Applikation erhält. Smarty unterstützt einen Sicherheitsmodus, der insbesondere den Aufruf von PHP-Code aus Templates verbietet. Der Sicherheitsmodus ist für diese Applikation aktiviert.

In Templates kann jede Form von Textformaten geschrieben werden. Sehr häufig wird das HTML-Format verwendet, welches im Browser dargestellt werden kann. In HTML-Templates lässt sich allerdings auch Javascript-Code schreiben, was für manche Zielstellung sogar unablässig ist. Javascript kann allerdings auch für Angriffsszenarien auf die Privatsphäre der Templatebenutzer verwendet werden. Somit ist es möglich in Javascript private Daten, welche im Browser gespeichert sind, an bestimmte Rechner im Internet zu schicken. Diese Attacke wird gewöhnlich als *Cross-Site-Scripting* bezeichnet. Im Moment gibt es in LESS keine automatische Filterung oder Klassifizierung von Templates die Javascript verwenden. Es ist dem Benutzer selbst überlassen jedes Template, welches er verwendet, auf schädlichen Code zu überprüfen. Dieses Problem soll in der nächsten Version der Software behandelt werden. Eine mögliche Lösung könnte die redaktionelle

Kontrolle von Templates sein, eine andere wäre die automatische Klassifikation der Templates, die keinen Javscript-Code enthalten. Diese könnten dann ohne Weiteres verwendet werden und der Kontrollaufwand wäre auf die potentiell gefährlichen Templates reduziert.

Kapitel 6

Anwendung und Einsatz

Im vorigen Kapitel wurde die Realisierung der Implementierung genauer beleuchtet. Im nun folgenden Kapitel wird auf die Anwendung der entwickelten Komponenten bezogen auf die Anforderungen aus Kapitel 3 eingegangen.

Als Referenz für die Evaluation der Anforderungen wird die Testinstallation der Applikation unter `http://less.aksw.org` verwendet. Alle hier vorgestellten Beispiele sind mit der Testinstallation realisiert. Die aufgezeigten Templates können mit dem Schlagwort „ESWC“ gefunden werden.

6.1 Flexible Darstellung von Linked Data-Ressourcen

Transformation von Daten in ein anderes Format zu deren Darstellung ist eines der ursprünglichen Aufgabengebiete von Templates. Da in LESS eine Templateengine für Semantic Web-Daten umgesetzt wurde, sind die Anforderungen bereits grundlegend erfüllt. Als Beispiel zur Evaluation dieser Anforderung wurde die Darstellung einer Faktensammlung zu einer Stadt ausgewählt. Im DBpedia-Projekt gibt es zu sehr vielen Städten strukturierte Informationen. Sollen diese nun ähnlich wie in Wikipedia in einer Tabelle dargestellt werden, kann das „City Fact Box“-Template verwendet werden. Das Ergebnis des gerenderten Templates sowie seine Integration in einen Wordpress-Blog können in Abbildung 6.2 angesehen werden. Ein Ausschnitt aus dem benötigten Templatecode zeigt Abbildung 6.1.

```
<table width="330">
  <tr>
    <th align="center" colspan="2" style="background-
color:#f2f2f4;"><font size="+1">{{rdfs:label@de}}</font>
    </th>
  </tr>
  <tr>
    <th colspan="2"><a href="{{foaf:depiction}}"><img
border="0" width="330" src="{{foaf:depiction}}"></a>
    </th>
  </tr>
  <tr>
    <th colspan="2" style="background-
color:#f2f2f4;">Data</th>
  </tr>
  <tr style="background: #ffffff;">
    <td>Area:</td><td>{{dbpprop:area}}km²</td>
  </tr>
  <tr style="background: #ffffff;">
    <td>Inhabitants:</td>
    <td>{{dbpprop:population}}
 ({{dbpprop:popDate}})</td>
  </tr>
```

Bild 6.1: Auszug aus dem Templatecode für das „City Fact Box“-Template

Die im Beispiel verwendete URL zum Abruf des Templates ist

```
http://less.aksw.org/build?id=6
      &uri=http%3A%2F%2Fdbpedia.org%2Fresource%2FBerlin.
```

Soll das Template nun für eine andere Stadt wiederverwendet werden, muss lediglich der URI-Parameter für die Linked Data-Ressource zur gewünschten Stadt angepasst werden und der Benutzer erhält eine aktuelle Darstellung.

6.2 Integration von Darstellungen von RDF-Daten in Webanwendungen

6.2.1 Integration über eine Webservice-URL

Hat der Benutzer ein Template angelegt, ist die Integration in eine bestehende Webapplikation sehr einfach. Im *Template Builder* werden eine Reihe von URLs angeboten, mit denen ein Zugriff auf dieses Template möglich ist. Diese so genannten Permalinks werden ausführlich in Kapitel 5.3.2 erklärt. Eine Repräsentation des reinen Textinhaltes kann über einen einfachen Aufruf des *Template Processors* (siehe Kapitel 5.3.3) geschehen.

Soll das Template zum Beispiel in eine bestehende Webseite oder einen Blog eingefügt werden, bietet sich der zweite Permalink des *Template Builders* an. Er nutzt Javascript um den Inhalt des Templates in eine HTML-Seite zu schreiben. Wird das oben genannte Städtetemplate in den Artikel eines Wordpress-Blogs eingefügt, kann der Javascriptcode exakt so verwendet werden, wie er im *Template Builder* angeboten wird. Der verwendete Wordpresscode und das Ergebnis in der Weblogseite ist in der Abbildung 6.2 dargestellt.

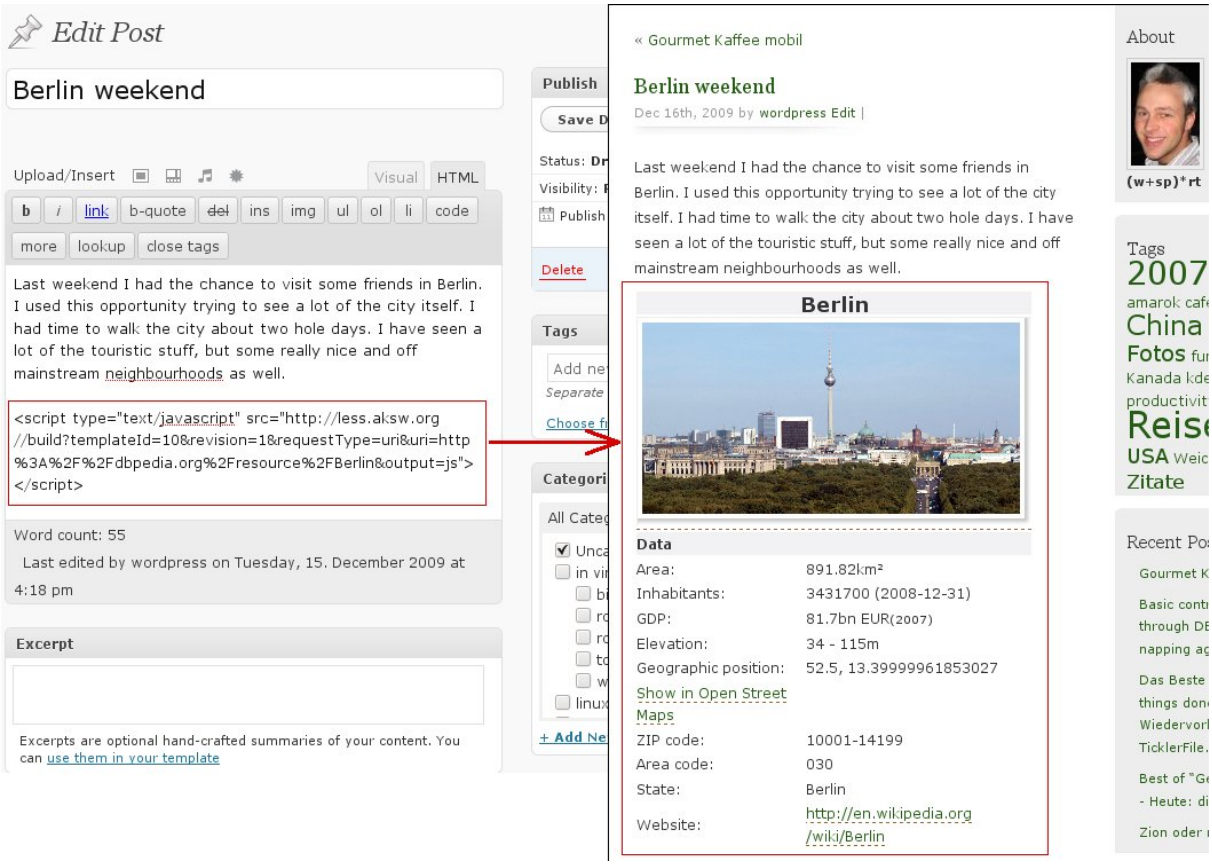


Bild 6.2: Integration des Städtetemplates in Wordpress

Um die Integration vieler Templates in die selbe Webapplikation zu erleichtern, sind Erweiterungen geeignet. Dies ist für verschiedene Plattformen denkbar. Im folgenden Kapitel wird eine Erweiterung für Typo3 erläutert.

6.2.2 Integration in Typo3

Im Rahmen dieser Arbeit wurde eine Erweiterung des verbreiteten Content Management Systems (CMS) Typo3 entwickelt. Nach Installation der Erweiterung kann im CMS ein neuer Typ eines Frontendplugins verwendet werden. Das Plugin „Semantische Templates“ bringt eine Oberfläche mit, welche die Integration von bestehenden Templates aus einem *Template Repository* in eine Typo3-basierte Webseite erleichtert. Das Plugin bietet dem

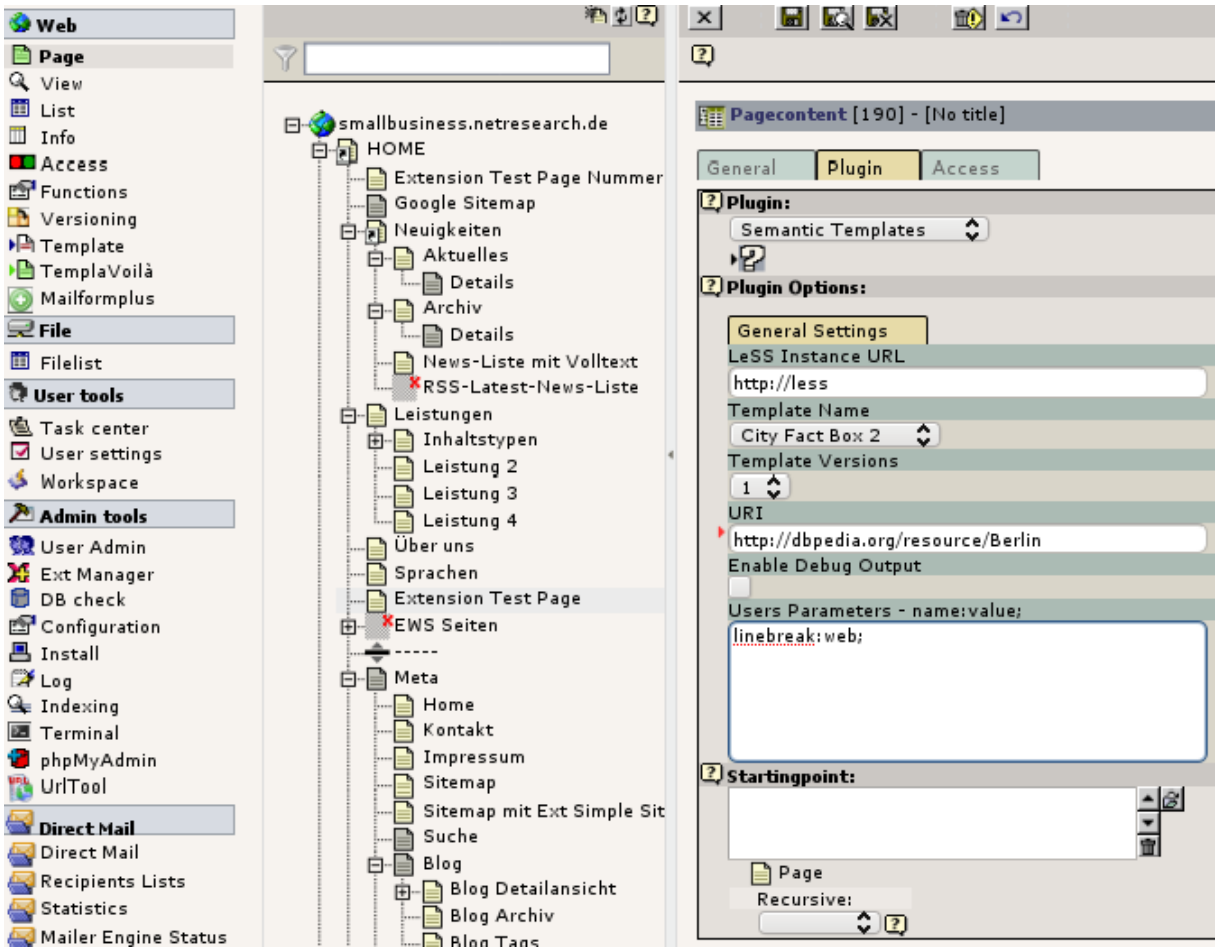


Bild 6.3: Typo3-Erweiterung zur Integration von LESS-Templates

Benutzer Eingabefelder für alle wichtigen Konfigurationsdaten, die für die Verwendung eines konkreten Templates notwendig sind. Abbildung 6.3 zeigt die Eingabemaske der Erweiterung auf der rechten Seite.

Nach Eingabe der LESS-Instanz wird vom Webservice der Applikation eine Liste mit verfügbaren Templates geholt. Aus dieser Liste kann der Benutzer das gewünschte Template auswählen. Danach wird die Liste der verfügbaren Versionen des Templates aktualisiert. Auch hier kann die gewünschte Version nun aus einer Drop-Down-Box ausgewählt werden. Abhängig von der Art des Templates werden Eingabefelder für die Datenquellendetails, eine URI oder eine SPARQL-Anfrage sowie ein SPARQL-Endpunkt angeboten. Sollte es notwendig sein, können in einem weiteren Eingabefeld noch benutzerdefinierte Parameter angegeben werden. Das letzte Eingabefeld ist eine Checkbox, welche zur Fehlersuche den Debugmodus für dieses Template aktivieren kann. Wird das Feld ausgewählt, werden im Templatecode im Falle eines Fehlers die Fehlermeldungen direkt ausgegeben.

6.3 Syndikation von veränderlichen Inhalten

RDF wird auch zur Speicherung aktueller sich verändernder Inhalte verwendet. Ein Beispiel sind die Programmdaten der BBC. Diese werden in RDF gespeichert und sind vollständig als Linked Data und teilweise über einen SPARQL-Endpunkt abrufbar. Im Moment gibt es nur zwei offizielle SPARQL-Endpunkte, welche nicht live aktualisiert werden. Dies ist allerdings für die nahe Zukunft geplant.

Die Programmdaten können vom SPARQL-Endpunkt abgerufen und einfach in einem Template verarbeitet werden. Als Anwendungsszenario wurde ein Template gewählt, welches die nächsten Sendungen einer bestimmten Sendereihe anzeigt. Das Template heißt „BBC Favorites“ und ist ebenfalls auf der LESS-Testinstallation zu finden. Im Template werden alle Sendungen der Sendereihe „Mountain“ vom SPARQL-Endpunkt abgefragt. Sie werden dann als Liste angezeigt. Da die Daten im Talis-SPARQL-Endpunkt[1] nicht aktuell sind, werden im Moment Sendungen angezeigt, die nach einem festgelegten Datum liegen. Es ist allerdings überhaupt kein Problem, das jeweils aktuelle Datum zu verwenden, sobald die Endpunkte Livedaten enthalten. Somit kann das Template dann zukünftige Sendungen anzeigen. Das Ergebnis könnte in die Webseite eines Fanclubs zur Sendereihe integriert werden und es wären immer die folgenden Sendungen sichtbar. Das Template kann in Abbildung 6.4 angesehen werden. Auch hier ist es mit einer einfachen Aktualisierung der SPARQL-Abfrage möglich das Template für eine andere Sendung zu verwenden.

Series: Mountain

Griff Rhys Jones explores the great mountain ranges of Britain, from Scotland southwards. Next show times on BBC One:

- **2009-05-30** 20:00 - 21:00 - Northern Scotland
  Griff Rhys Jones explores the wilderness of the northwest highlands of Scotland.
- **2009-09-25** 20:00 - 21:00 - Northern Scotland
  Griff Rhys Jones explores the wilderness of the northwest highlands of Scotland.
- **2009-10-02** 22:00 - 23:00 - The Lakes
  Griff visits the Lake District.
- **2009-10-02** 20:00 - 21:00 - The Lakes
  Griff visits the Lake District.

Bild 6.4: Template zur Anzeigen von Sendungen einer bestimmten Sendereihe

`data.gov`[2] ist ein Projekt der amerikanischen Regierung welches Daten und Statistiken

[1] http://api.talis.com/stores/bbc-backstage/services/sparql
[2] http://www.data.gov/

veröffentlicht. Darunter sind aktuelle Daten zu Erdbeben. Das Projekt Data-gov Wiki[3] ist bestrebt die veröffentlichten Daten in RDF umzuwandeln. Aus diesem Grund gibt es eine Linked-Data-Datenquelle, welche alle weltweit registrierten Erdbebendaten der letzten sieben Tage enthält. Diese werden im Template „Mashup Google Map EQ-Locations“ verwendet und mithilfe der Google-Maps-API in einer interaktiven Karte (siehe Abbildung 6.5) dargestellt. Für den Zugriff auf die Daten dieser Datenquelle ist der im Abschnitt 5.5 beschriebene Parameter `instances` notwendig, da das RDF eine Reihe von Ressourcen enthält, die jeweils ein einzelnes Erdbeben beschreiben.

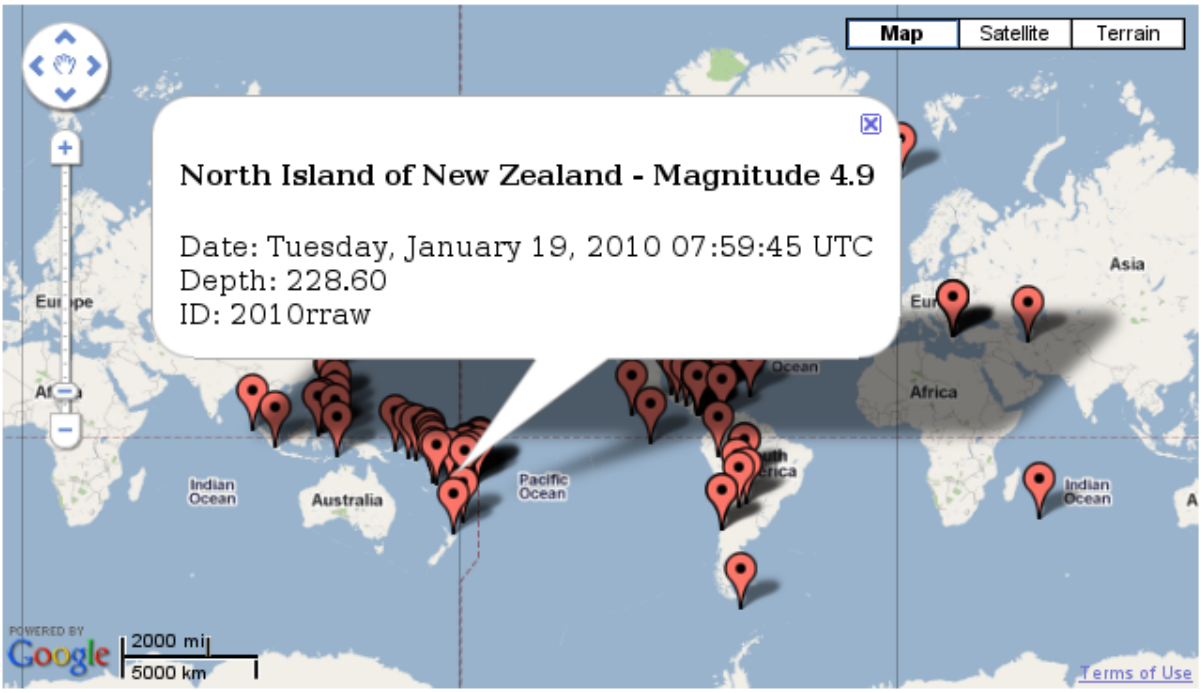
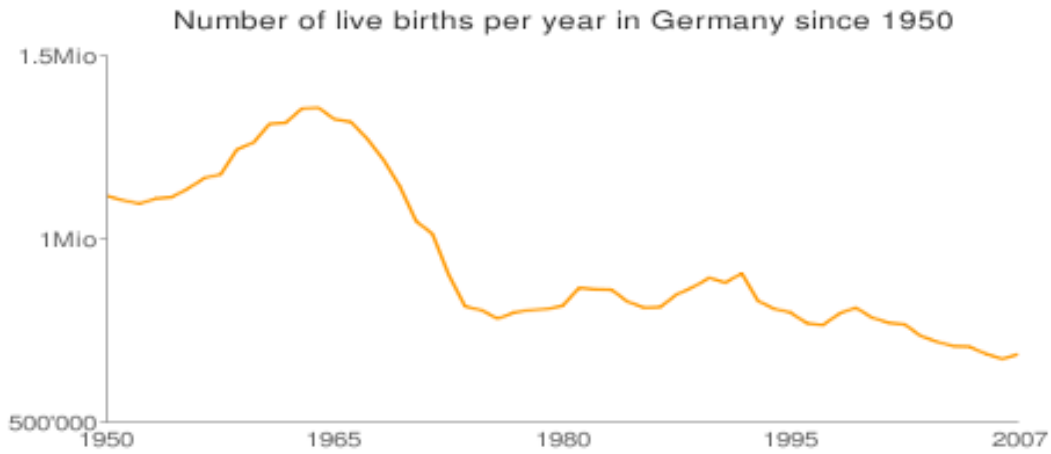


Bild 6.5: Mashup Google Maps API - Data.gov Erdbebendaten

6.4 Mashups

Durch die Möglichkeit der Verwendung von Javascript in Templates können sehr einfach Mashups mit bestehenden Javascript-Bibliotheken und APIs erstellt werden. Es ist möglich Eingabedaten für diese Anwendungen direkt aus dem Linked-Data-Web zu verwenden. Als Umsetzung wurde das Template „Eurostats Birth Data Germany“ erstellt. Es verwendet eine in RDF konvertierte Versionder Eurostats Daten zu Geburtenraten in Deutschland. Außerdem wird die Google Charts API verwendet um aus den Daten ein Diagramm zu erstellen. Der Templatecode sowie das gerenderte Template sind in Abbildung 6.6 zu sehen.

[3] http://data-gov.tw.rpi.edu/wiki/The_Data-gov_Wiki




```
<img src="http://chart.apis.google.com/chart?chs=450x250&chd=t:
{$data_de}&cht=lc&chds=500000,1500000&chtt=Number+of+live+births
+per+year+in+Germany+since+1950&chxt=x,y&chxl=0:|1950|1965|1980|
1995|2007|1:|500'000|1Mio|1.5Mio">
```

Bild 6.6: Mashup Google Charts API und Eurostatsdaten zur Geburtenstatistik

6.5 Datenaggregation aus unterschiedlichen Quellen

Während es in LESS nicht direkt möglich ist, mehrere Datenquellen pro Template zu verwenden, ist dies sehr wohl durch Verwendung mehrerer Templates zu erreichen. Bei komplexeren Aufgabenstellungen ist es im Allgemeinen notwendig mehrere Templates zu verbinden.

Im hier beschriebenen Fall soll die Mitarbeiterwebseite eines Projektes mithilfe der Daten aus den FOAF-Profilen der Projektmitglieder erstellt werden. Dazu ist es lediglich notwendig ein Dokument zu schaffen, welches durch Verwendung von `foaf:Group` die Mitglieder der Gruppe mit der URI zum jeweiligen FOAF-Profil auflistet. Dann kann mithilfe eines Templates eine Iteration über die einzelnen Mitarbeiter vorgenommen werden. Für jedes Mitglied wird dann ein Detailtemplate aufgerufen, welches sich um die Darstellung der Informationen kümmert. Diese Aufgabe wurde mit den Templates „FOAF Group Adressbook“ und „AKSW FOAF Card“ umgesetzt. Der Quellcode der beiden Templates sowie ein Bild des entstandenen Ergebnisses sind in Abbildung 6.7 zu sehen.

```
<h2>{{foaf:name}}</h2>
{{dc:description}} <br />
<a href="{{foaf:homepage}}">web</a>
<div id="members">
 {foreach {{foaf:member}} as $var}
  {template id="5" uri="{$var}"
    parameter_language="en"}
 {/foreach}
</div>
```

```
<div id="foaf-card-block">
 {if {{foaf:depiction}} !== ''}
  <img class='photo'
    src="{{foaf:depiction}}" />
 {/if}
 <div id="name">{{foaf:name}}</div>
 ...
 {if {{foaf:phone}} !== ''}
  <div id="tel">{{foaf:phone}}</div>
 {/if}
 <div id="email">
  <a href="{{foaf:mbox}}">
   <img src="email.png" />
   {if $language === 'en'}
     email {else} E-Mail
   {/if}
  </a>
 </div>
</div>
```

AKSW
Agile Knowledge Management and Semantic Web
homepage

Sören Auer
http://www.informatik.uni-
tel:+49(341)97-32323
email

Sebastian Dietzold
http://sebastian.dietzold.
tel:+49-341-97-32366
email

Raphael Doehring
http://www.raphas.net/
tel:+49-341-112321
email

Bild 6.7: Mitarbeiterseite generiert aus FOAF-Profilen der Mitglieder

Kapitel 7

Verwandte Arbeiten

Die Arbeiten zur Darstellung, Wieder- und Weiterverwendung von RDF-Daten lassen sich grundlegend in zwei Gruppen einteilen. Auf der einen Seite gibt es interaktive benutzeroberflächenbasierte Ansätze, welche es dem Benutzer ermöglichen durch eine interaktive Oberfläche Daten auszuwählen und u.U. aus unterschiedlichen Datenquellen zusammen zu stellen. Diese Ansätze zeichnen sich durch eine niedrige Einstiegshürde für den Benutzer aus, weil sich die Oberfläche relativ schnell intuitiv selbst erklärt. Auf der anderen Seite gibt es eher programmatisch orientierte Ansätze, welche eine Transformations- oder Templatesprache schaffen, um RDF-Daten weiterzuverarbeiten. Für diese Ansätze ist ein größerer Lernaufwand notwendig, allerdings übersteigen sie die Flexibilität der benutzeroberflächenorientierten bei weitem.

Zur ersten Gruppe zählt Sig.ma[1]. Sig.ma bietet eine fest tabellenorientierte Möglichkeit zur Darstellung und Weiterverwendung von semantischen Daten. Der Benutzer kann Datenquellen zu einem bestimmten Thema auswählen, ihre Reihenfolge anpassen und das so gestaltete Sigma in eine andere Webseite integrieren. Diese Applikation bietet somit die Möglichkeit semantische Daten zu aggregieren. Der Nutzer hat jedoch keinen großen Einfluss auf die Gestaltung der Informationen.

Auch marbles[2] ist der Gruppe der benutzeroberflächenzentrierten Ansätze zuzuordnen. Es setzt auf das Fresnel-Vokabular auf, welches später in diesem Kapitel genauer vorgestellt wird. Es handelt sich bei marbles um eine Webapplikation, die es erlaubt, mithilfe von Fresnel *lenses* und *formats* RDF-Daten aus unterschiedlichen Datenquellen zu aggregieren und in der Applikation darzustellen. Der Schwerpunkt liegt dabei auf der Entdeckung von neuen Quellen zu einer Ressource und der Auswahl von bestimmten Quellen oder Daten. Marbles ist für die Darstellung der Daten auf einen von drei vorgefertigten Views für

[1] http://sig.ma/
[2] http://marbles.sourceforge.net/

das XHTML-Format angewiesen. Damit bietet es nur sehr eingeschränkte Flexibilität bezüglich des Zieldatenformates.

Zur zweiten Gruppe, der programmatisch orientierten Lösungen, gehört Xenon [QK05]. Xenon ist eine Transformationssprache, welche sich stark an XSLT orientiert. RDF-Daten werden durch eine festgelegte Transformation in eine andere textorientierte Darstellung umgewandelt. Die Einstiegshürde ist wie für alle programmatisch orientieren Ansätze relativ hoch, besonders wenn noch keine Vorkenntnisse in Programmierung oder Webentwicklung vorliegen.

Einen sehr generischen Ansatz zur Lösung der Darstellung und Umwandlung von RDF-Daten bietet Fresnel [PBKL06]. Es handelt sich dabei um ein Vokabular zur Beschreibung von Darstellungkonzepten für RDF-Daten. Die Grundlagen dieses Ansatzes ähneln der Idee hinter Xenon. Fresnel bietet dazu zwei Hauptkomponenten: *lenses* und *formats*. *lenses* sind ein Filter mit dem RDF-Daten zur Darstellung ausgewählt werden können. *formats* haben die Aufgabe, die Art der Darstellung für ausgewählte Ressourcen festzulegen. Das Ergebnis nach der Transformation durch Fresnel ist noch keine endgültige Darstellung, sondern eine Baumstruktur mit darstellungsspezifischen Informationen. Die konkrete Interpretation dieser Struktur wird der implementierenden Software überlassen.

Neben marbles gibt es noch weiter RDF-Browser die Fresnel implementieren. Beispiele dafür sind IsaViz[3] (W3C/INRIA), Longwell[4] / Piggy Bank [HMK05] und Horus[5] (Freie Universität Berlin). Fresnel bietet eine Grundlage für die Implementierung einer Applikation zur Darstellung von RDF-Daten. Es bietet dem Nutzer noch nicht die Möglichkeit, Daten ohne weitere Hilfsmittel (XSLT) in ein konkretes Textformat seiner Wahl zu transformieren. Des Weiteren gibt es keine vorgesehene Plattform zum Austausch und zur Weiterverwendung bereits erstellter Arbeiten.

Ein weiterer Vertreter der Gruppe der programmatisch orientierten Ansätze ist Tal4RDF. Es ist eine in Python geschriebene Programmierbibliothek, die auf Zopes Template Attribute Language basiert [Cha09]. Ziel dieser Lösung ist es eine leichtgewichtige Templatesprache für RDF-Daten zu schaffen. Die Bibliothek erlaubt es RDF-Daten in ein beliebiges Textformat umzuwandeln. Ähnlich zu der in dieser Arbeit vorgeschlagenen Lösung wird ein Webservice bereitgestellt, der es ermöglicht Templates zu rendern. Tal4RDF legt im Unterschied zu der hier vorgestellten Lösung jedoch keinen Wert auf die Speicherung und Verwaltung der erstellten Templates. Darüber hinaus bietet die Plattform keinerlei Unterstützung bei der Entwicklung von Templates. SPARQL-Datenquellen werden momentan noch nicht unterstützt.

[3] http://www.w3.org/2001/11/IsaViz/

[4] http://simile.mit.edu/longwell/

[5] http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/tutorial/horus/index.htm

Die Visualization Providers for Ontology ElemenTs (VPOET) sind ein Hilfsmittel, das zur Darstellung von Ontologieelementen in Java implementiert wurde [RCOC09]. Basierend auf der Fortunata-Bibliothek [RCC09] kann der Benutzer ein individuelles Template zur Darstellung von semantischen Daten erstellen. Auch ein Webservice zum Rendern der Templates ist vorhanden. Insofern ist der Service dem in dieser Arbeit entwickelten ähnlich. Die Unterschiede sind jedoch ebenfalls sehr vielfältig. So erlaubt das VPOET-System ausschließlich die Darstellung von Attributen, die zu einer Ontologie gehören. Eine Darstellung von Attribute aus einer anderen Ontologien sind nicht möglich. Die hier vorgeschlagene Templatesprache übersteigt die in VPOET eingeführte, sowohl in Mächtigkeit als auch in Benutzerfreundlichkeit. VPOET unterstützt keine Schleifen zur Iteration über eine Menge von Ergebnissen. Die verwendeten bedingten Operatoren sind kompliziert und überprüfen nur das Vorhandensein eines Attributes. VPOET verwendet keine intuitiven Platzhalter zur Datenadressierung und der Benutzer wird bei der Auswahl der gewünschten Properties nicht unterstützt.

Ein weiteres sehr mächtiges Werkzeug zur Weiterverarbeitung und Verbindung von Daten aus dem (Semantic) Web sind die Semantic Web Pipes [LpPH+09]. Ähnlich wie LESS bieten diese dem Nutzer visuelle Unterstützung bei der Entwicklung seiner Applikation. In einem graphischen Editor stellt der Nutzer aus Komponenten den Ablauf des Informationsflusses zusammen. Die einzelnen Komponenten können entweder Datenquellen anfragen, oder Daten verarbeiten. Der Schwerpunkt bei den Semantic Web Pipes liegt auf der Vereininigung von Daten über die selbe Ressource, die aus unterschiedlichen Quellen stammen.

Kapitel 8

Schlussfolgerungen und Ausblick

Das Ziel der Arbeit eine Strategie zur Wieder- und Weiterverwendung von Daten im Semantic Web zu schaffen, welche nur geringe technische Spezialkenntnisse verlangt, wurde erreicht. Die Semantic Web Template Engine LESS ist eine der ersten Arbeiten, die dem Endanwender den Zugriff auf Linked Data erleichtert. Mit LESS können einfach Templates zur strukturierten Darstellung und Transformation von RDF-Daten erstellt werden. Da der Benutzer durch einen visuellen Editor und einen „Property Recommender“ bei der Entwicklung unterstützt wird, ist dies auch mit geringeren technischen Kenntnissen möglich.

Die Entwicklung an LESS ist jedoch mit dem Abschluss dieser Arbeit noch nicht beendet. Es wurden bereits während der Arbeit Probleme entdeckt, welche zur optimalen Verwendung der Applikation gelöst werden müssen. Besonders im Bereich der Benutzbarkeit der Applikation gibt es Verbesserungspotential. Die Entwicklung wird in Zukunft weitergehen und durch Feedback der Benutzer priorisiert werden.

Es gibt des Weiteren vielfältige Ansatzpunkte den hier vorgestellten Ansatz zu erweitern.

Der angesprochene, verteilte Einsatz der Komponenten könnte LESS für eine ganze Reihe anderer Einsatzszenarien interessant machen. Eine Integration der Template Engine in Ontowiki wäre eine Möglichkeit, da in einem großen Repository u. U. Templates für die Anzeige von unbekannten Typen von Daten enhalten sind. Somit könnte LESS als Lieferant für Vorschaudarstellungen von RDF-Daten fungieren.

Die Unterstützung der Integration von LESS-Templates in bestehende Applikationen wie in Kapitel 6.2 beschrieben, kann für weitere Plattformen fortgeführt werden. Besonderes Interesse gilt als Erstes den Plattformen Wordpress sowie iGoogle. Aufgrund ihrer großen Verbreitung und Bekanntheit wäre es möglich sehr viele Nutzer zu erreichen. Die Integration in Wordpress könnte auch zu einem Service erweitert werden, welcher anhand der im

Artikel verwendeten Worte Templates und passende Datenquellen zur Vervollständigung des Inhaltes vorschlägt.

Interessante Perspektiven ergeben sich auch aus der Idee LESS als Plattform für die Verwendung unterschiedlicher Templatesprachen zu öffnen. Die in Kapitel 7 erwähnte Sprache Tal4RDF zum Beispiel könnte neben LeTL unterstützt werden. Dies würde eine Konzentration der Benutzer auf eine Plattform ermöglichen, ohne sie auf eine Templatesprache einzuschränken.

Auch die Templatsprache LeTL selbst wird natürlich von einer Weiterentwicklung profitieren. Eine Erweiterung um absolute Pfade oder eine Möglichkeit zum Umgang mit mehreren Datenquellen in einem Template könnten Vorteile für die Anwender bringen.

Kapitel 9

Kurzzusammenfassung

Die Veröffentlichung von strukturierten Daten im Linked Data Web hat stark zugenommen. Für viele Internetnutzer sind diese Daten jedoch nicht nutzbar, da der Zugriff ohne Kenntnis einer Programmiersprache nicht möglich ist.

Mit der Webapplikation LESS wurde eine Templateengine für Linked Data-Datenquellen und SPARQL-Ergebnisse entwickelt. Auf der Plattform können Templates erstellt, veröffentlicht und von anderen Nutzern weiterverwendet werden. Der Nutzer wird bei der Entwicklung von Templates unterstützt, so dass es auch mit geringen technischen Kenntnissen möglich ist, mit Semantic Web-Daten zu arbeiten.

LESS ermöglicht die Integration von Daten aus unterschiedlichen Quellen, sowie die Erzeugung textbasierter Ausgabeformate wie RSS, XML und HTML mit Javascript. Templates können für unterschiedliche Ressourcen erstellt und anschließend einfach in bestehende Webapplikationen und Webseiten integriert werden.

Um die Zuverlässigkeit und Geschwindigkeit des Linked Data Web zu verbessern, erfolgt eine Zwischenspeicherung der verwendete Daten in LESS für eine bestimmte Zeit oder für den Fall des Ausfalls der Datenquelle.

Unter `http://less.aksw.org` ist eine erste öffentliche Installation von LESS erreichbar.

Anhang A

Paper

Auf Grundlage dieser Diplomarbeit ist ein Paper entstanden, welches zur Konferenz *European Semantic Web Conference* 2010 (ESWC) für den Linked Data Track eingereicht wurde.

LESS - Template-Based Syndication and Presentation of Linked Data for End Users

Sören Auer[1], Raphael Doehring[2], and Sebastian Dietzold[1]

[1] Universität Leipzig, Institut für Informatik,
Postfach 100920, D-04009 Leipzig, Germany,
{auer|dietzold}@informatik.uni-leipzig.de
http://aksw.org
[2] Netresearch GmbH & Co. KG,
Nonnenstrasse 11d, D-04229 Leipzig,
raphael.doehring@netresearch.de
http://netresearch.de

Abstract. Recently, the publishing of structured, semantic information as linked data has gained quite some momentum. For ordinary users on the internet, however, this information is not yet very visible and (re-) usable. With LESS we present an end-to-end approach for the syndication and use of linked data based on the definition of templates for linked data resources and SPARQL query results. Such syndication templates are edited, published and shared by using a collaborative Web platform. Templates for common types of entities can then be combined with specific, linked data resources or SPARQL query results and integrated into a wide range of applications, such as personal homepages, blogs/wikis, mobile widgets etc. In order to improve reliability and performance of linked data, LESS caches versions either for a certain time span or for the case of inaccessibility of the original source. LESS supports the integration of information from various sources as well as any text-based output formats. This allows not only to generate HTML, but also diagrams, RSS feeds or even complete data mashups without any programming involved.

1 Introduction

Recently, the publishing of structured, semantic information as linked data has gained much momentum. A large number of linked data providers meanwhile publishes more than 200 interlinked datasets amounting to 13 billion facts[1]. Despite this initial success, there are a number of significant obstacles, which hinder the large-scale deployment and use of the linked data web. These obstacles are primarily related to the quality and coherence of linked data as well as to providing direct benefits to end users. In particular for ordinary users of the Internet, linked data is not yet sufficiently visible and (re-) usable.

[1] http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/DataSets/Statistics

With the template-based syndication and presentation approach LESS presented in this article, we target primarily the *usability* aspect of linked data for end users. Another important problem of the linked data web tackled by LESS are quality issues, in particular with regard to *performance* and *reliability* of linked data endpoints, the importance of which has, for example, been noted earlier in [4].

LESS represents an end-to-end approach for the syndication and use of linked data based on the definition of templates for the visual presentation of linked data resources and SPARQL query results. LESS allows to edit and publish syndication templates and to share them by using a collaborative Web platform. Templates for common types of entities can then be combined with specific, linked data resources or SPARQL query results and integrated into a wide range of applications, such as personal homepages, blogs/wikis, mobile widgets etc.

As a result, a blogger writing about a recent trip to Berlin can easily integrate a nicely formatted fact box with important information about Berlin obtained from Wikipedia into her blog post. A community of science fiction fans can integrate lists on a recent BBC programming matching their preferences into their community portal. Wikipedia authors can use LESS to generate pages with lists from DBpedia [6] content. Citizen scientists interested in earthquakes can display recent earth crust activity in their region on a map without having to do any programming.

LESS supports the integration of information from various sources as well as any text-based output formats. This allows not only to generate HTML, but also diagrams, RSS feeds or even complete data mashups without any programming involved. In order to improve reliability and performance of linked data, LESS caches versions of the retrieved linked data resource descriptions or SPARQL query results either for a certain time span (thus improving the performance) or for the case of inaccessibility of the original source (thus improving reliability).

LESS represents an end-to-end solution by not only focusing on a single aspect, but tackling template definition, processing, authoring and sharing in a coherent way. Concrete usage scenarios tackled by LESS include for example:

- the flexible visualization of linked data resources,
- the creation of views on the linked data web,
- the integration of linked data into existing applications such as Content Management Systems, Wikis, Weblogs etc.
- the integration and compilation of information from various sources,
- the end-user-driven generation of linked data mashups.

The paper is structured as follows: We present the high-level LESS concept and architecture in Section 2. We report about our LESS implementation in Section 3. We present a number of complementary use cases for LESS in Section 4. We review related work in Section 5 and conclude with an outlook on future work in Section 6.

2 Concept and Architecture

From the usage scenarios mentioned in the introduction (and described in more detail in Section 4) we derived the following requirements:

- support for various data access paradigms, in particular direct linked data URI requests and SPARQL queries,
- simple template language, which is, on the one hand, easy to learn and, on the other hand, flexible enough to accommodate a wide range of usage scenarios (in particular the ones described in the introduction),
- mitigation of the current reliability and performance problems of linked data endpoints,
- facile integration of the processed templates into various Web applications, e.g. via a REST API,
- enable the collaborative authoring, sharing and re-use of templates.

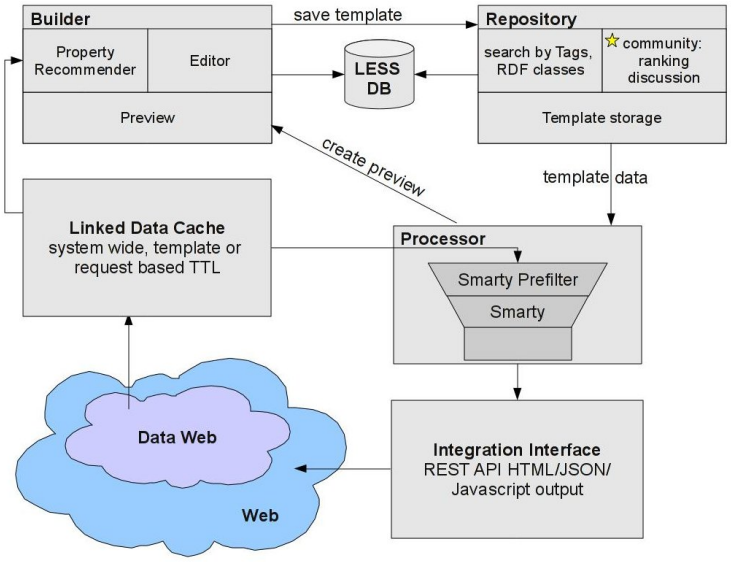


Fig. 1. LESS system architecture.

In order to fulfill these requirements, we developed the LESS architecture as presented in Figure 1. The main LESS components are:

- *template language:* defines a declarative language for creating textual output form RDF resources and SPARQL results,
- *template builder:* allows users to define and edit LESS templates in a collaborative manner,

– *repository:* stores template revisions and metadata, allows to retrieve templates based on various annotations,
– *processor:* combines a template with concrete data retrieved from the data web,
– *linked data cache:* stores retrieved linked data resources and SPARQL queries for reuse.

In the following subsections we describe these components in more depth.

2.1 Template Language

A core part of LESS is the LESS Template Language (LeTL), which allows to define arbitrary text-based output representations. In general, the use of various template languages (such as Fresnel [7] or Tal4RDF [2]) is possible with LESS. However, in order to lower the entrance barrier for end users, we decided to specify a template language tailored for simplicity, convenience and versatility. LeTL is based on the popular Smarty[2] template language, but uses some custom extensions. From Smarty LeTL inherits custom functions, variable modifiers, loops, conditional parts based on the evaluation of complex expressions, cache handling, a plugin architecture and many other features. However, LeTL additionally comprises functionality for direct interaction with RDF data and SPARQL query results.

A LeTL template is applied to each individual resource described in an RDF document or every row in a SPARQL result set (unless the template application is restricted to resources of a certain class). The properties (or columns) of the resource (or SPARQL result row) are made available for reference within the LeTL template by using the syntax showcased in Table 1. The LeTL syntax allows to iterate through multiple property values and to reference recursively (sub-)templates, which are applied to dereferenced object property values. An example of such a recursive template, which iterates through a list of `foaf:person` resources, dereferences these resources and calls another template for each one of them, is shown in Figure 2.

2.2 Template Builder

The template builder (depicted in Figure 3) enables end users to create LESS templates. The template builder comprises the template editor, a property recommender and a template rendering preview. The template editor supports syntax highlighting for HTML. The property recommender suggests properties based on a given, example linked data resource or SPARQL query. Once a certain property has been selected, the corresponding Smarty code to display the value of the property is added to the current cursor position in the template editor. Once a stable version of a certain template has been created by using the template builder, additional metadata (such as a template name and tags) can be attached to the template and a revision can be stored in the template repository.

[2] `http://www.smarty.net/`

Syntax	Description	Example
`{{property}}`	Refers to the value of the property 'property'	`{{foaf:name}}`
`{{p@lang}}`	Refers to the value with language tag 'lang' of the property 'p'	`{{rdfs:comment@en}}`
`{{p^^dtype}}`	Refers to the value with datatype 'dtype' of the property 'p'	`{{dbp:birth^^xsd:date}}`
`{{p1->p2}}`	Refers to the value of the property 'p2' of the resources referred to by the property 'p1'	`{{foaf:currentProject ->rdfs:label}}`
`{template id="id" uri="uri" instances="" sortBy=""}`	Processes the template with id 'id' and resource 'uri' (using the optional attributes 'instances' iteration can be restricted and sorted with 'sortBy')	`{template id="5" uri="{$var}"}`
`{template id="id" sparql="query" endpoint="srv"}`	Processes the template with id 'id' and SPARQL query 'query' from endpoint 'srv'	`{template id="3" sparql="SELECT * WHERE {}" endpoint="http://dbpedia.org"}`

Table 1. LeTL syntax extensions to the Smarty template language.

2.3 Template Repository

The template repository allows to publish templates, to browse existing templates based on their supported RDF classes and user-defined tags as well as to rate, comment and reuse existing templates. The template repository is implemented on top of a relational database, but made available as RDF by using Triplify [1] at `http://less.aksw.org/triplify`. For each template, a unique template id is assigned. As soon as a template is changed, its revision id is incremented. Each registered LESS user can only change her own templates in order to prevent conflicts. However, a user can create her own copy of any of the templates stored in the template repository. The public availability of templates in the repository has a number of advantages: templates serve as examples for new users, they can be used by other third-party applications and the reuse of templates facilitates a natural modularization.

2.4 Template Processor

The template processor is the actual LESS execution environment. It takes a LESS template and a linked data resource or SPARQL query and renders the respective output. By default the LESS template processor iterates through all the resource descriptions or SPARQL query result items and applies the defined template to each of them. This allows to apply LESS also to RDF documents, which contain more than one resource description. However, in certain cases, the template application should be restricted to resources of a certain type. For these cases, the LESS template can be associated with a certain RDF class. This can be, for example, the class `foaf:person` for a FOAF profile, thus preventing the

```
<h2>{{foaf:name}}</h2>
{{dc:description}} <br />
<a href="{{foaf:homepage}}">web</a>
<div id="members">
 {foreach {{foaf:member}} as $var}
  {template id="5" uri="{$var}"
    parameter_language="en"}
 {/foreach}
</div>
```

```
<div id="foaf-card-block">
 {if {{foaf:depiction}} !== ''}
  <img class='photo'
    src="{{foaf:depiction}}" />
 {/if}
 <div id="name">{{foaf:name}}</div>
 ...
 {if {{foaf:phone}} !== ''}
  <div id="tel">{{foaf:phone}}</div>
 {/if}
 <div id="email">
  <a href="{{foaf:mbox}}">
   <img src="email.png" />
   {if $language === 'en'}
     email {else} E-Mail
   {/if}
  </a>
 </div>
</div>
```

AKSW
Agile Knowledge Management and Semantic Web
homepage

Sören Auer
http://www.informatik.uni-
tel:+49(341)97-32323
email

Sebastian Dietzold
http://sebastian.dietzold.d
tel:+49-341-97-32366
email

Raphael Doehring
http://www.raphas.net/
tel:+49-341-112321
email

Fig. 2. Left: LESS templates for displaying a group profile including information about group members, obtained from separate FOAF profiles; Right: rendered output

application of the template for a person's projects described in the same FOAF file. When there is more than one resource of the type selected, the `sortBy` parameter can be used to order the resources based on the values of a certain property. The template processing can be additionally influenced by user-defined parameters, which can be accessed from within the template definition. Based on a system, template or request configuration, the template processor can be instructed to use a locally cached version of the linked data resource or SPARQL query. This particularly facilitates situations, where linked data endpoints are temporarily not available or respond slowly.

2.5 Integration Interface

In order to integrate the output of the template processor into external applications, LESS provides an REST API via the URL `http://less.aksw.org/build`. Required URL parameters are the `id` of the template, the `revision` of the template (can be omitted for accessing the last revision) and either `uri` or `sparql` for defining the linked data resource or the SPARQL query to be used. By default this REST function call returns the rendered output as text (in most cases HTML snippets). By using the optional URL parameter `output`, Javascript or JSON can be alternatively selected as output format. A further optional URL

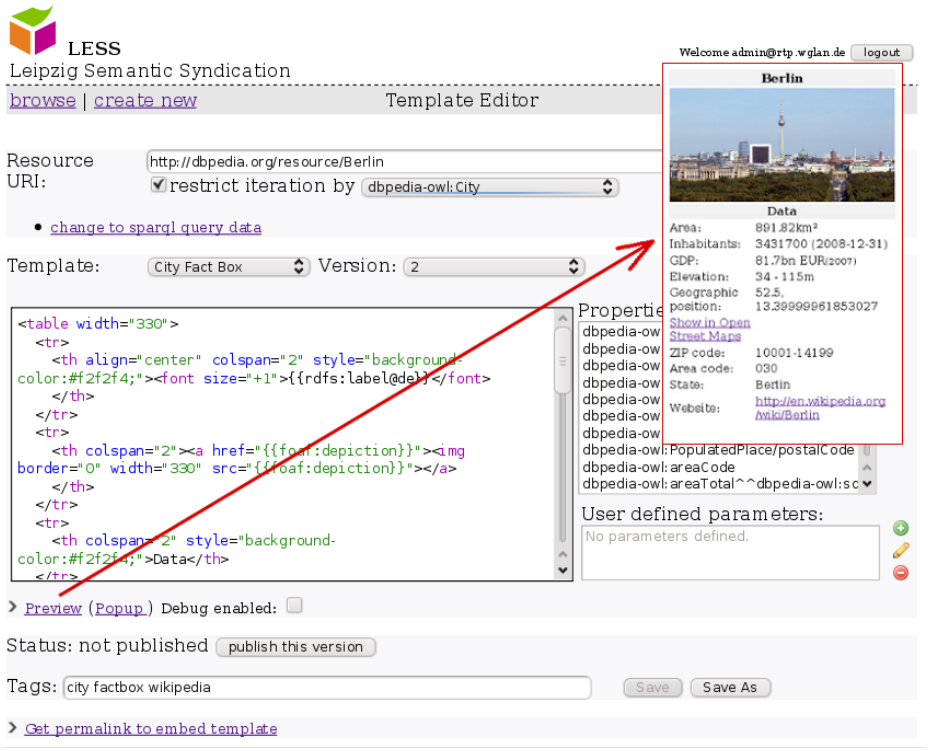


Fig. 3. Template Builder comprising editor, property recommender and rendering preview.

parameter is `ttl`, which specifies the time-to-live of a previously cached version of the linked data resource or SPARQL query. In case this parameter is missing, template- or system-based defaults are used. An example of a REST request (whose result is depicted in Figure 5) would look as follows:

```
http://less.aksw.org/build?id=2&
                    uri=http%3A%2F%2Fdbpedia.org%2Fresource%2FBerlin
```

3 Implementation

The LESS implementation was performed in PHP by using the Zend Framework[3] and the Erfurt framework for the development of semantic web applications [3]. The implementation follows the generic MVC architecture model and the convention-over-configuration paradigm, which assumes reasonable defaults

[3] `http://framework.zend.com`

for all possible configuration parameters. A demonstration platform with the LESS implementation is available at: `http://less.aksw.org`

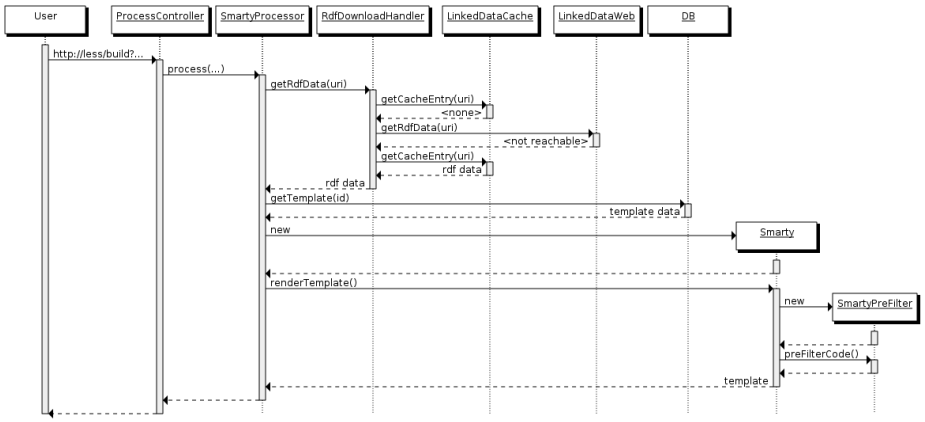


Fig. 4. Template processing UML sequence diagram.

A UML sequence diagram illustrating the template processing is displayed in Figure 4. After a template instance is requested by a user via the integration interface, the template processor checks the local cache or retrieves the data. Based on the template id, the template data and metadata are obtained from the relational database. The LeTL template is then compiled by the `SmartyPreFilter` into a valid Smarty template and compiled by the Smarty processor into PHP code. The Smarty processor also takes care of caching-compiled templates in order to increase the performance.

4 Usage Scenarios

In this section we present examples for employing LESS for the various usage scenarios identified in the introduction. The examples presented are available in the LESS template repository with tag ESWC[4].

4.1 Flexible Resource Visualization

Although there are some approaches for rendering RDF (e.g. [7, 2]), it is currently quite cumbersome to visualize RDF and linked data in user-defined ways. LESS provides a very flexible and easy-to-learn mechanism for visualizing linked data resources and SPARQL query results, since end users can directly create output fragments with placeholders for RDF data. Figure 5 demonstrates how a LESS

[4] `http://less.aksw.org/browse?tags=\%2Beswc`

template can be used to visualize a linked data resource and to integrate the resulting output into a Wordpress blog. By annotating the generated visual representations with RDFa, the resource descriptions can be easily obtained and reused not only from the original data source, but also from the syndicated content representations.

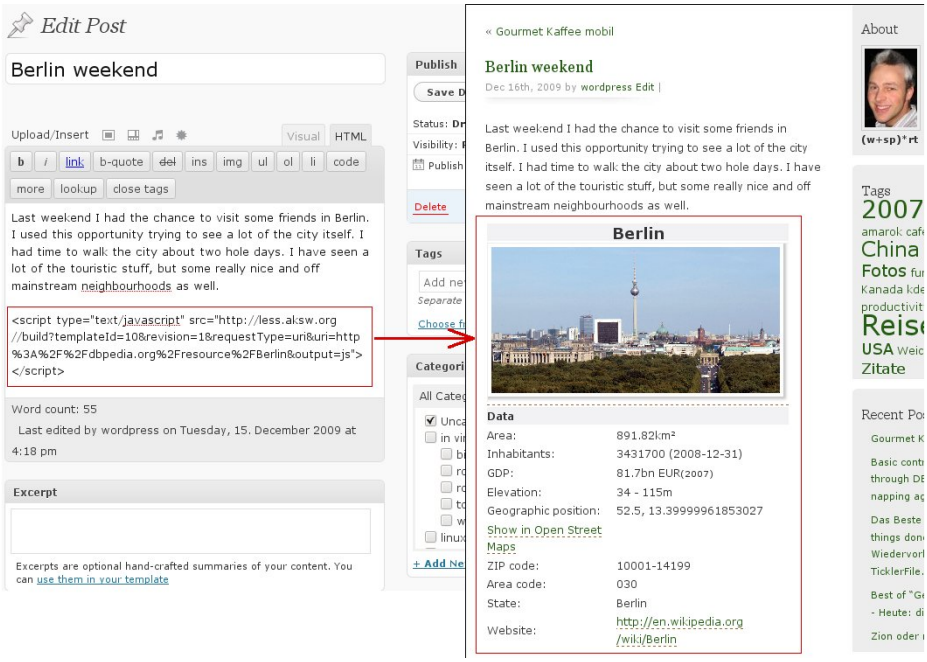


Fig. 5. Integration of a LESS template visualizing data obtained from DBpedia into a Wordpress blog post by using a Javascript snippet.

4.2 Linked Data View Creation and Visualization

In many cases not only a single linked data resource, but information from various interlinked linked data resources or SPARQL queries should be displayed. Figure 6 showcases a rendered template, which obtains information about recent BBC programmings related to the series 'Mountain'. The information is obtained by querying a SPARQL endpoint containing the BBC program information and rendering the query result by employing a LESS template.

4.3 Integration of Information from Various Sources

LESS is not limited to create visual representations of single RDF or SPARQL sources. The possibility to dereference linked data resources or call (sub-)templates

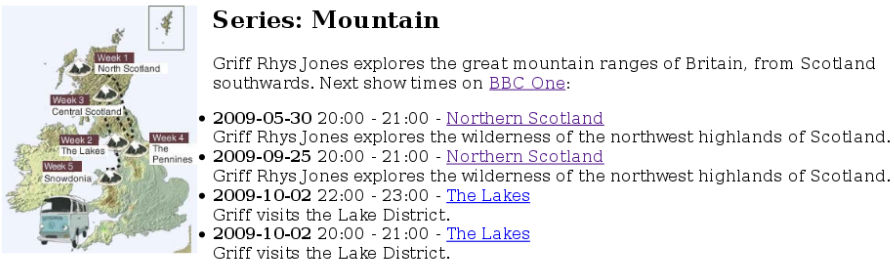


Fig. 6. Rendered LESS template representing a view on recent BBC programmings.

from within a LESS template (as described in Table 1) allows to integrate information from various sources into a coherent visual representation. In Figure 2 we illustrate this usage scenario with a template, which iterates through members of a group obtained from a group FOAF file and processes an individual FOAF template for each member with data from their personal FOAF profiles.

4.4 Template Integration into Existing Applications

With LESS' REST style integration interface it is very easy to integrate rendered templates into existing Web applications, be it Weblogs, CMS, Wikis, traditional Web pages or any other Web application. LESS offers the template integration into existing application via employing HTML snippets, Javascripts, JSON or IFRAMEs. The integration of linked data content can be even further simplified, if Web applications offer to integrate LESS templates directly, i.e. without the detour to the LESS homepage. In order to showcase how such a LESS integration can be performed, we developed a Typo3 extension, which allows to define Typo3 page content objects based on LESS templates directly from within the Typo3 user interface (cf. Figure 7).

4.5 Mashups

LESS is not limited to employing RDF and solely creating HTML output. Due to the flexible template language, arbitrary text-based output can also be generated, including Javascript, various XML formats (such as RSS), CSV etc. In particular, LESS enables the combination of RDF data with WebAPIs in order to create data mashups. Figure 8 shows a LESS template processing data from Eurostats and combining it with the Google Charts API in order to create a chart with births per year over time. The Google Charts API is accessed by creating a special URL, which returns the generated chart as an image. In the example the URL is constructed by means of a LESS template. Using LESS, users do not have to perform any programing in order to construct the URL for accessing the API. Likewise, the LESS output (i.e. the diagram in this example) dynamically changes as the underlying data changes.

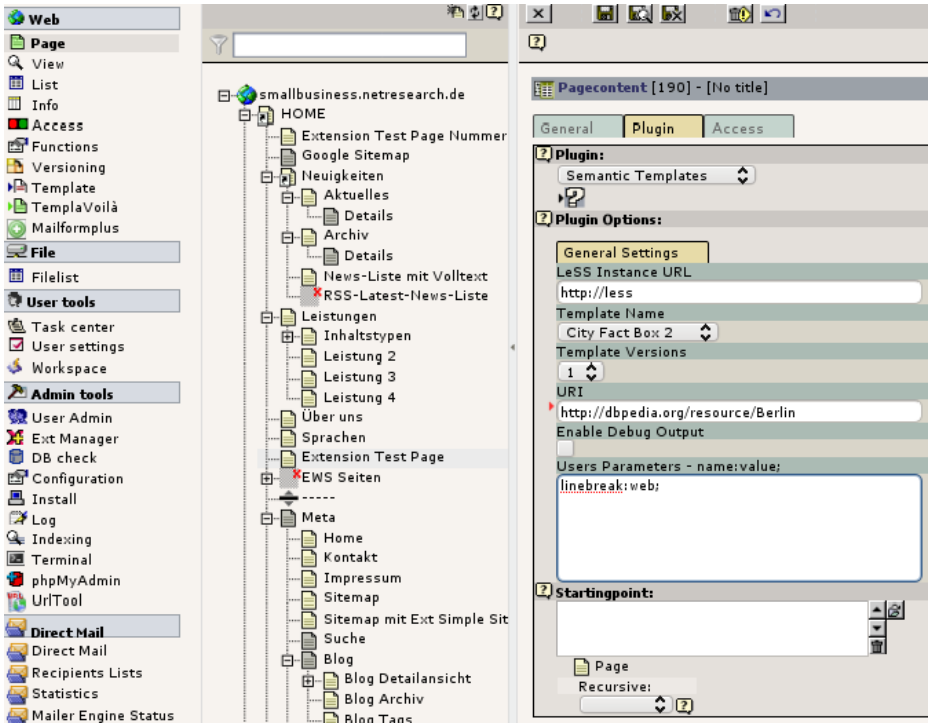


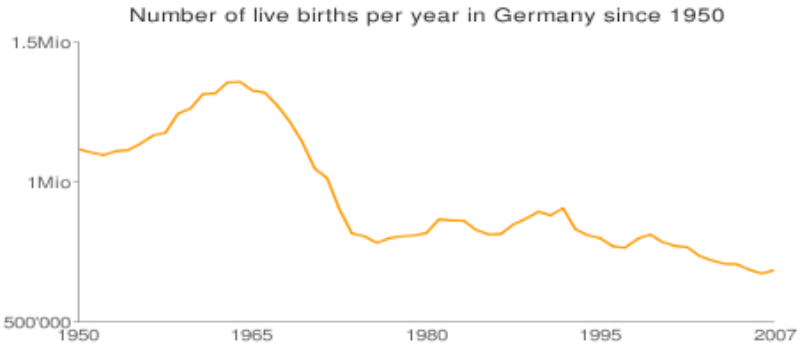
Fig. 7. LESS integration into Typo3: After specifying the LESS instance to use, the user is provided with a list of possible templates. The user is then able to specify the necessary template parameters through a user interface rather than by manually creating the REST URI. This simplifies integrating a LESS template into Typo3 even further.

5 Related Work

Related work dealing with the visual presentation and (re-)use of RDF data can be divided into two main groups. On the one hand, there are interactive, user-interface-centric approaches, which enable the user to select and combine data interactively. The entrance barrier of these approaches is relatively low, since user interfaces are intuitive and easy to understand. On the other hand, there are programmatic-oriented approaches that specify a transformation or template language for RDF data. Although more effort is necessary to get acquainted with them, the latter provide more versatility and flexibility.

Sig.ma[5] is part of the first group and offers a fixed, table-oriented presentation of linked data. The user is enabled to select a certain data source, to edit the ordering of data and to integrate the result into an existing web page. Apart from the order the user does not have any influence on the layout of the information.

[5] http://sig.ma/




```
<img src="http://chart.apis.google.com/chart?chs=450x250&chd=t:
{$data_de}&cht=lc&chds=500000,1500000&chtt=Number+of+live+births
+per+year+in+Germany+since+1950&chxt=x,y&chxl=0:|1950|1965|1980|
1995|2007|1:|500'000|1Mio|1.5Mio">
```

Fig. 8. Mashup template combining the Google Charts API with Eurostats data.

Marbles[6] is also part of the user-interface-centric group of approaches. Marbles is a web application that allows to aggregate and present RDF data from different sources by using Fresnel *lenses* and *formats* (see also below). Its focus lies on the aggregation and selection of new sources for an RDF resource as well as the selection of data. Marbles is limited to the presentation of data for XHTML clients through Fresnel-based views. There are only three views currently in use. The produced format is limited to the views.

Xenon [8] belongs to the second group of programmatic-oriented approaches. It defines a transformation language in the style of XSLT and allows a transformation from existing RDF data into any text-based format. Again, having to learn a new transformation language means a higher learning effort to be able to use the technology.

A very generic way to solve the problem of presentation and transformation of RDF data is offered by Fresnel [7]. The basic idea is similar to Xenon. Fresnel is a vocabulary for describing the basic concepts of RDF data presentation. It offers two main components: *lenses* and *formats*. While *lenses* are a filter to select the data to use, *formats* describe the way a set of data is presented. The result of a Fresnel transformation is not a final presentation of the data but a tree structure of data annotated with presentation-related information. The actual interpretation of this structure is delegated to the browser software using Fresnel. In addition to Marbles, other RDF browsers implementing Fresnel are IsaViz[7] (W3C/INRIA) and Longwell[8] / Piggy Bank [5]. Although very related, Fresnel is quite different from LESS, since it defines its own mechanisms for data

[6] http://marbles.sourceforge.net/
[7] http://www.w3.org/2001/11/IsaViz/
[8] http://simile.mit.edu/longwell/

selection (LESS uses simple projections and SPARQL), its formatting approach is more difficult to learn (due to the RDF representation of formatters) and it is less aligned with the linked data paradigm.

Tal4RDF [2] is a programming library written in Python based on Zope's Template Attribute Language. The goal of this approach is to create a light-weight template language for RDF data. The library allows to transform RDF data into any text-based format. Similar to LESS, a web service to render the templates is provided. However, Tal4RDF does not offer any means to publish and share templates, there is no support for SPARQL and dynamic linked data dereferencing.

The Visualization Providers for Ontology ElemenTs (VPOET) is a tool developed for the presentation of ontology elements [10]. It is based on the Fortunata library [9]. The user is enabled to create a template presenting RDF data, and a web service for rendering the templates is provided. Different to LESS, VPOET is limited to the presentation of resources belonging to a single ontology. VPOET's template language appears to be slightly cumbersome (e.g. with regard to the data referencing), as it does not support loops for iterating through resources, and the conditional evaluation of template parts cannot be based on complex conditions such as the comparisons.

6 Conclusions and Future Work

With LESS we aimed at contributing to mitigate the largest obstacles for a large-scale deployment of the linked data web: the demonstration of direct benefits to end users and improving the reliability and performance of linked data endpoints. LESS represents an end-to-end solution by not only focusing on a single aspect, but tackling template definition, processing, authoring and sharing in a coherent and pragmatic way. As a result, end users are empowered to make use of linked data without the need to program or gain deep understanding of the technologies involved.

We see LESS as a first step towards bringing the data web closer to potential end users. As a continuing effort, we aim, in particular, to tackle the following enhancements and improvements in future work:

- usability improvements, such as a visual template designer supporting drag-and-drop of template elements and the integration of OntoWiki's SPARQL query builder for a more user-friendly construction of complex queries,
- development of plugins for standard Web applications, which enable users to define and integrate LESS templates from a single environment,
- better integration with data web services and search indexes such as Sindice,
- template language extensions, such as a template-in-template mechanism, support for property groups representing common information and support for different template languages such as Fresnel and Tal4RDF.

Acknowledgments

We would like to thank our colleague Christian Weiske for the helpful comments and inspiring discussions during the development of LESS. This work was supported by a grant from the German Federal Ministry of Education and Research (BMBF), provided for the project LE4SW.

References

1. Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify: light-weight linked data publication from relational databases. In *18th International Conference on World Wide Web, WWW 2009*, pages 621–630. ACM, 2009.
2. Pierre-Antoine Champin. T4R: Lightweight presentation for the Semantic Web. In Gunnar Aastrand Grimnes Chris Bizer, Sören Auer, editor, *Scripting for the Semantic Web, workshop at ESWC 2009*, June 2009.
3. Norman Heino, Sebastian Dietzold, Michael Martin, and Sören Auer. Developing semantic web applications with the ontowiki framework. In Tassilo Pellegrini, Sören Auer, Klaus Tochtermann, and Sebastian Schaffert, editors, *Networked Knowledge - Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 61–77. Springer, Berlin / Heidelberg, 2009.
4. Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of OWL class descriptions on very large knowledge bases. *International Journal on Semantic Web and Information Systems*, 2009.
5. David Huynh, Stefano Mazzocchi, and David Karger. Piggy Bank: Experience the Semantic Web inside your web browser. In *The Semantic Web ISWC 2005*, pages 413–430. Springer Berlin / Heidelberg, March 2005.
6. Jens Lehmann, Chris Bizer, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 2009.
7. Emmanuel Pietriga, Christian Bizer, David Karger, and Ryan Lee. *Fresnel: A browser-independent presentation vocabulary for RDF*, volume 4273, pages 158–171. Springer Berlin / Heidelberg, 2006.
8. Dennis Quan and David R. Karger. Xenon: An rdf stylesheet ontology, 2005. Available online at `http://simile.mit.edu/mail/GetAttachment?listId=9&msgId=2687&attachId=1` visited on December 22nd 2009.
9. Mariano Rico, David Camacho, and Óscar Corcho. A contribution-based framework for the creation of semantically-enabled web applications. *Information Sciences*, 2009.
10. Mariano Rico1, David Camacho1, and Óscar Corcho. VPOET Templates to Handle the Presentation of Semantic Data Sources in Wikis. In Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *Proceedings of the Fourth Workshop on Semantic Wikis The Semantic Wiki Web*, pages 186–190, Hersonissos, Crete, Greece, 2009.

Anhang B

Hilfeseite LESS

Dieser Hilfetext wurde für LESS entworfen und ist Teil der Applikation.

Important Security Warning - Disclaimer

There is no way yet to automatically filter template content before delivering it. This means that anything including (malicious) script code can be written into a template. **Reusing templates happens at your own risk**. Please check every template (view the code) before you use it to make sure it does only what you want.

General information

LESS is a web application to create templates for (re-)using RDF data. It consists of three main parts:

- Template Builder - the editor for developing the templates
- Template Repository - to browse existing templates
- Template Processor - to render created templates

It allows you to use RDF data from a Linked Data URI or from a SPARQL query result and format it in any text based format you want. The template language is called LeTL and is describe here.

Questions, Bugs, Feedback

Please use our google group for questions, bugreports and feedback. Feedback is welcome!

Template Repository

A good start to get to know the application is the repository. It allows you to browse templates that were created by other users. You can select templates by tags or RDF classes. You can publish your own templates to be visible in the repository as well. Every template is presented in a preview box. It shows the tags, the rating of the template as well as a small life preview of the rendered template. Using the "View"link you can view every template in the Template Builder.

Template Builder

The Builder is the editor to create templates. If features a property recommender showing possible properties found in the configured datasource. You can simply click on the property and it will be pasted in to the template code. The code editor if configured to highlight HTML code. While editing you can check the state of your work using the life preview under the code editor. Furthermore you can edit template name or tags associated.

Permalinks

At the very bottom of the Builder you will find three permalinks allowing access to the current template. Using these links it is possible to integrate the template in an existing web application or web page. Right now there are three ways to integrate the template: Javascript, IFrame and pure HTML.

Publishing Templates

Templates created are private by default. That means, that no one will see them in the template repository, except for you. If you think your template is ready to be shown to the rest of the world, you can publish it using the button next to the save buttons. Publishing a templates means that this revision of the template is read only as of this moment. You will be able to edit the template, but you will than work on a newer

revision. Visible in the repository will only be the older published version of the template until you publish the newer one.

Template Processor

The Processor is a webservice for rendering the created templates. It will retrieve data from the given datasource and apply it to the template, delivering the content in one of three formats. The Processor is reachable using the "/buildÜRL.

An example URL is:

```
http://less.aksw.org/build?id=6&revision=1
        &uri=http%3A%2F%2Fdbpedia.org%2Fresource%2FBerlin.
```

The processor expects a number of parameters. The minimum you have to specify are the templates id (*id*) and the data source information. In case of a Linked Data RDF document that is just the URI (*uri*). In case of a SPARQL query it is the query itselft (*query*) and the SPARQL endpoint to query (*endpoint*).

Mandatory parameters (on of uri or query/endpoint)

- id - the id of the template to use
- uri - the uri of the Linked Data resource
- query - a SPARQL query
- endpoint - the SPARQL endpoint to query

If you use the URL just like that the result will be pure HTML output. You can change the output to be embeded into Javascrit write code, so that it can be integrated in any webpage by simply adding the output parameter (*output=js*). Another optional parameter is *debug*. This one comes in handy, when you don't get any result and want to know if there was an error. Add it to the url and errors that occured will be shown as well.

Another set of optional parameters deals with special situation with Linked Data URIs. Sometimes Linked Data documents contain data about more than one resource. To be able to access this kind of data we created the *instances* parameter. As value it expects

the URI of an RDF class. The template will than be applied to resources only, that are of the specified class. Additionally a *sortBy* parameter can be added. The value is expected to be a valid property of the beforementioned resource. If found, the resources are applied to the template sorted by the values of this property.

User defined parameters (see here) offer a much higher flexibilty in using templates. If you want to pass such a parameter to the templates through the url you have to simply add another GET parameter called *parameter_<your-parameters-name>*. If you want to set a language the parameter would look like that: *parameter_language*.

Optional parameters

- output - left blank for HTML, 'js' for Javascript embedding ready code
- debug - show debugging messages
- instances - RDF class of the resources to use from data sources
- sortBy - property of a resource to sort results by
- parameter_<your-parameters-name> - pass a user defined parameter

User Defined Parameters

A very flexible way to reuse template for different purposes is through the application of users defined parameters. They allow the access to additional information from within the template.

Depending on how a template is called, the parameters are called in a different way. You can pass the parameters through the URL (see chapter above) or throught the *template* function (see below) as well.

Inside the template you can access your parameter simply by using a native Smarty variable with the same name. So in our example from above we could find out about the language like that:

```
{if $language == 'de'}
    Willkommen!
{else}
    Welcome!
{/if}
```

Template Language

A core part of LESS is the LESS Template Language (LeTL), which allows to define arbitrary text-based output representations. LeTL is tailored for simplicity, convenience and versatility. LeTL is based on the popular Smarty template language, but uses some custom extensions. From Smarty LeTL inherits custom functions, variable modifiers, loops, conditional parts based on the evaluation of complex expressions, cache handling, a plugin architecture and many other features. However, LeTL additionally comprises functionality for direct interaction with RDF data and SPARQL query results.

The usage of the template language will be described by datasource used, since there are slight differences.

Linked Data Sources

For the us with Linked Data datasources the following special template variables can be used. These expressions offer access to properties found in the datasource belonging to the main resource.

Code	Description	Example
{{property}}	Refers to the value of the property‘property’	{{foaf:name}}
{{p@lang}}	Refers to the value with language tag‘lang’ of the property ‘p’	{{rdfs:comment@en}}
{{p}}	Refers to the value with datatype‘dtype’ of the property ‘p’	{{dbp:birthxsd:date}}
{{p1->p2}}	Refers to the value of the property‘p2’ of the resources referred to by theproperty ‘p1’	{{foaf:currentProject->rdfs:label}}

Some advanced expressions can be used with the special placeholders as well.

The conditional expression (*if*) can be used as follows.

```
{if {{foaf:name}} != ''}
    <tr>
        <td>
            Name:
        </td>
        <td>
            {{foaf:name}}
```

```
        </td>
    </tr>
{/if}
```

Another important example is the foreach loop. If offers a way to iterate through a collection of properties. It can be used as follows:

```
<ul>
    {foreach {{rdf:type}} as $var}
        <li>{$var}</li>
    {/foreach}
</ul>
```

SPARQL Datasource

Using SPARQL datasources simpler since we can use native Smarty variables. The variables used in the SPARQL Query are registered to the template and can be accessed as native Smaery template variables. Further information can be found on the Smarty homepage. Especially expressions like *if* (see here) and *foreach* (see here) can be used exactly like in Smarty.

Template in Template

For more complex scenarios it might be necessary to call another template from within a template. You can certainly simply use full permalinks to do so. But we also created a shortcut to do this. There is a Smarty function called *template* that allows for calling templates directly.

```
 {template
    id="3"
    uri="http://dbpedia.org/resource/Smarty"
 }
```

You can as well pass user defined parameters using this shortcut. Here is an example from one of our templates:

```
{foreach {{foaf:member}} as $var}
   {template id="5" uri="{$var}" parameter_language="en"}
{/foreach}
```

Further Smarty Functions

There are many handy Smarty function. We want to explain just one more, that comes in very handy, if dealing with a string that has to be used several times. You can define your own variables in Smarty by using the *assign* function.

```
{assign var="url" value="http://blog.aksw.org/2009/eswc10-in-use-track/"}.
```

A lot more can be found in the Smarty manual.

Cache TTL

LESS caches RDF data sources as well as templates themselves. The user can influence the cache timeout in two ways. The first one is a template setting, that can be changed in the Template Builder. The second one is to add a url parameter when calling the template processor. The parameter is simply called *ttl* and accepts a number of seconds. If no TTL is specified the system default of one day is used.

Debugging

When you try to get a template using the Template Processor and you don't get any result, there might be several reasons why that is happening. One could be a problem compiling the Smarty code, another could be reachability of the datasource. Whenever calling a template using the processor, you can simply add the URL or *template* function parameter *debug*. If there are error messages they will be shown to you.

Literaturverzeichnis

[ADL+09] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify: light-weight linked data publication from relational databases. In *Proceedings of the 18th international conference on World wide web*, page 621–630. ACM, 2009.

[BHAR07] Christian Bizer, Tom Heath, D. Ayers, and Y. Raimond. Interlinking open data on the web. *4th European Semantic Web Conference. http://www.eswc2007.org/pdf/demo-pdf/LinkingOpenData.pdf*, 2007.

[BL06] Tim Berners-Lee. Linked data. Website, 2006. URL: `http://www.w3.org/DesignIssues/LinkedData.html`; (25. November 2009).

[Cha09] Pierre-Antoine Champin. T4R: Lightweight presentation for the Semantic Web. In Gunnar Aastrand Grimnes Chris Bizer, Sören Auer, editor, *Scripting for the Semantic Web, workshop at ESWC 2009*, June 2009.

[HDMA09] Norman Heino, Sebastian Dietzold, Michael Martin, and Sören Auer. Developing semantic web applications with the ontowiki framework. In Tassilo Pellegrini, Sören Auer, Klaus Tochtermann, and Sebastian Schaffert, editors, *Networked Knowledge - Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 61–77. Springer, Berlin / Heidelberg, 2009.

[HMK05] David Huynh, Stefano Mazzocchi, and David Karger. Piggy Bank: Experience the Semantic Web inside your web browser. In *The Semantic Web – ISWC 2005*, pages 413–430. Springer Berlin / Heidelberg, March 2005.

[LBK+09] Jens Lehmann, Chris Bizer, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 2009.

[LpPH+09] Danh Le-phuoc, Axel Polleres, Manfred Hauswirth, Giovanni Tummarello, and Christian Morbidoni. Rapid prototyping of semantic mash-ups through

semantic web pipes. In *Proceedings of the 18th international conference on World wide web*, page 581–590. ACM, 2009.

[PBKL06] Emmanuel Pietriga, Christian Bizer, David Karger, and Ryan Lee. *Fresnel: A browser-independent presentation vocabulary for RDF*, volume 4273, pages 158–171. Springer Berlin / Heidelberg, 2006.

[QK05] Dennis Quan and David R. Karger. Xenon: An rdf stylesheet ontology, 2005. Available online at `http://simile.mit.edu/mail/GetAttachment?listId=9&msgId=2687&attachId=1` visited on December 22nd 2009.

[RCC09] Mariano Rico, David Camacho, and Óscar Corcho. A contribution-based framework for the creation of semantically-enabled web applications. *Information Sciences*, 2009.

[RCOC09] Mariano Rico, David Camacho, and Óscar Corcho. VPOET Templates to Handle the Presentation of Semantic Data Sources in Wikis. In Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *Proceedings of the Fourth Workshop on Semantic Wikis – The Semantic Wiki Web*, pages 186–190, Hersonissos, Crete, Greece, 2009.

[Ree79] Trygve Reenskaug. Mvc xerox parc 1978-79. Website, 1979. URL: `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`; (5. November 2009).

Abbildungsverzeichnis

Alle in dieser Arbeit verwendeten Abbildungen wurden vom Autor erstellt.

Leipzig, den 01. Februar 2010

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Ort Datum Unterschrift