Schließen auf Usability-Probleme mit OWL 2 RL

Manuel Dudda

Hochschule RheinMain Informatik Master of Science
Fachbereich Design Informatik Medien
Campus Unter den Eichen 5 65195 Wiesbaden , Deutschland
manuel.b.dudda@student.hs-rm.de
https://github.com/manuel84/abrupt

Abstract Die Usability von Internetauftritten wird größtenteils subjektiv wahrgenommen. Es existiert zwar eine Normierung durch die ISO für die Anforderungen an die Gebrauchstauglichkeit einer Software (EN ISO 9241-11), diese trifft jedoch nur sehr wage Aussagen und orientiert sich stark am respektiven Nutzungskontext. Trotzdem gibt es Merkmale einer Website, die allgemein als negativ für die Eigenschaften der Usability eingestuft werden können, so zum Beispiel schlechte Kontrastverhältnisse, ein hoher Lesbarkeitsindex und Diskrepanzen zwischen erwartetem und tatsächlichem Nutzerverhalten. Das "AbRUPt"-System der Hochschule RheinMain ist in der Lage viele dieser Merkmale einer Website automatisiert zu erfassen. Durch die Konvertierung, Modellierung und Entwicklung eines durchdachtes Regelsystem in der semantischen Ontologiesprache OWL 2 RL ensteht ein System, welches automatisiert Folgerungen auf Usability-Probleme ermöglicht.

1 Einleitung

Die Gestaltungsvielfalt von Webseiten hat sich seit der Freigabe des Internets für die kommerzielle Nutzung (ca. 1990) rasant weiterentwickelt. Anfangs bot nur eine kleine Anzahl von Institutionen, Firmen, Regierungen spärliche Informationen meist im Form von einfachen Texten an. Mittlerweile gehört die Webseite zum Aushängeschild eines jeden Unternehmens. Das Aussehen muss dabei häufig zur Unternehmensphilosphie passen, das sogenannte Cooperate Design sorgt für eine visuelle Identität, um sich von anderen Unternehmen zu unterschieden. Dazu gehören Schriftbild, Farben, Logos, und vieles mehr. Des Weiteren wurden jede Menge Web-Dienste entwickelt, die den Webseiten mehr Interaktivität einräumen. Webanwendungen ähneln komplexer Software, die eine grafische Benutzeroberfläche im Browser zur Verfügung stellen. Für die Benutzerfreundlichkeit gibt es eine ISO-Norm, die folgende Anforderungen an die Software stellt:

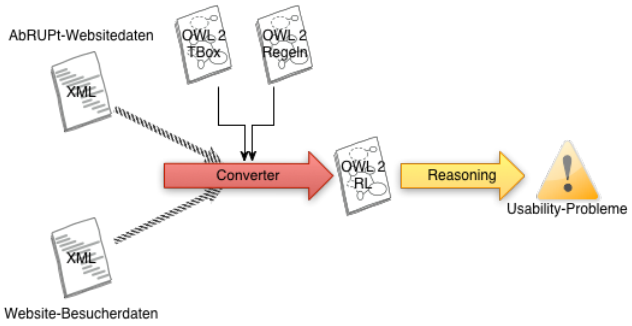
> Die Gebrauchstauglichkeit einer Software ist von ihrem Nutzungskontext (beinhalten den Benutzer, die Arbeitsaufgabe, die Arbeitsmittel wie z.B. Hardware oder Software sowie die physische und soziale Umgebung) abhängig, in dem sie eingesetzt wird. Im Teil 11 der ISO 9241 werden drei Leitkriterien für die Gebrauchstauglichkeit einer Software bestimmt:
> – Effektivität zur Lösung einer Aufgabe,
> – Effizienz der Handhabung des Systems,
> – Zufriedenheit der Nutzer einer Software.

Es wird deutlich, dass es keine objektive Bestimmung von Gebrauchstauglichkeit gibt, sondern diese stark vom Nutzungskontext abhängt. Insbesondere spielt die Zielgruppe dabei eine Rolle. Handelt es sich um ein jüngeres Publikum, die man mit einfacher Sprache zu erreichen versucht oder um Fachpublikum, für die komplexere Texte unabdingbar sind? Welche kulturellen Gegebenheiten sind zu beachten? Das Farbverständnis der Chinesen beispielweise weicht der westlichen Welt stark ab, so ist weiß die Farbe der Trauer. Ob eine Software dementsprechend gebrauchstauglich ist oder nicht, ist demnach nur subjektiv von Menschen bewertbar. Dennoch gibt es auch Merkmale, die allgemeingültig als mögliche Probleme eingestuft werden können:

– Schlechte Kontrastverhältnisse
– Seiten ohne/mit zu vielen Links
– Unausgeglichenes Bild/Text-Verhältnis
– Diskrepanzen zwischen erwarteter und tatsächliher Formular-Eingabezeiten
– Texte mit schlechtem Lesbarkeitsindex anhand [Sch03]
– Inkonsistenzen in der HTML-Struktur

Der Online-Service AbRUPt [Rhe14] von der Hochschule RheinMain bietet die Möglichkeit viele dieser Merkmale über eine Web-Schnittstelle zu erfassen. In [Bri14] wurde gezeigt, wie man diese Schnittstelle anwendet und daraus ein Konstrukt für eine mögliche Datenauswertung entwickelt. Diese Arbeit beschäftigt sich mit der Weiterentwicklung des darin beschrieben Converters und dessen Datenuswertung (Abb. 1).

Abb. 1. Funktionsweise des Converters



Die Daten liegen zunächst im XML-Format vor und werden in OWL konvertiert. Mittels OWL-basierter Regeln wird überprüft, ob eine aussagekräftige Datenauswertung möglich ist. Es wird dabei besonderer Wert auf das OWL-Profil "RL"gelegt, um Performance-Schwachstellen zu vermeiden. Das OWL 2-RL Profil basiert auf dem funiderten Grundlagen der Beschreibungslogiken (3.3).

2 Gliederung

Im Abschnitt 3 werden die theoretischen Grundlagen für das OWL 2 RL-Profil aufgearbeitet. Um zu verstehen, wie sich eine Beschreibungslogik (3.3) von der Aussagenlogik (3.1) abhebt werden kurze Beispiele gezeigt, wie sie auch in der Anwendung existieren könnten. Abschnitt 3.2 zeigt, dass mächtigere Aussagen mit der Prädikatenlogik möglich sind und welche Einschränkungen in der Entscheidbarkeit daraus entstehen. Insgesamt stellt die Implementierung (4) des Converters eine Erweiterung vorheriger Arbeiten dar. Welche Anpassungen dabei vorgenommen wurden wird in Abschnitt 4.1 beschrieben, welche Technologien in 4.2. Der wesentliche Bestandteil der Erweiterung besteht aus der Regelerstellung für das Schließen auf Probleme (5). Die Regeln sind in Inkonsistenz- (5.1), Produktions- (5.2), Listen- 5.3 und Datentyp-Regeln (5.4) unterteilt. Einige Herausforderungen können nur durch den Einsatz von Erweiterungen beschrieben in 5.5 gelöst werden. Verbundene Arbeiten werden in Abschnitt 6 erwähnt. Abschließend wird das Projekt in 7 zusammengefasst, ein Ausblick gewährt sowie ein persönliches Fazit gezogen.

3 Logik und Berechenbarkeit

Es gibt verschiedene Arten von formalen Logiken, die sich in ihrer Struktur und Mächtigkeit unterscheiden. Für triviale Anwendungsfälle reicht meistens die Verknüpfung von strukturlosen Elementaraussagen aus. Komplexere Aussagen lassen sich nur mit mächtigeren Logiken ausdrücken, für dessen automatiserten Schlussfolgerungen allerdings auch größere Aufwände entstehen.

3.1 Aussagenlogik

Mit der Aussagenlogik lassen sich in ihrer Mächtigkeit beschränkte Formeln aufstellen [HKR09, p. 6,373]. Sie entsprechen in der Programmierung dem Datentyp *Boolean*, also Elementarussagen, die verknüpft und negiert erscheinen und deren Wahrheitsgehalt *Wahr* oder *falsch* ist.

$$\begin{aligned} A &= \text{Die Checkbox hat den Namen } \textit{agb}. \\ B &= \text{Die Checkbox ist kein Pflichtfeld.} \\ A \to C &= \text{Die Checkbox hat den Namen } \textit{agb}\text{, also ist sie ein Pflichtfeld.} \\ (A \to C) \land B &= \lightning \text{ (Es liegt ein Problem vor.)} \end{aligned} \tag{1}$$

Abb. 2. Aussagenlogische Formeln

Aus den oben genannten Aussagen (1), folgt, dass die Checkbox mit dem Namen *agb* ein Pflichtfeld ist $(A \to C)$. Diese ist allerdings nicht als solches ausgezeichnet (B). Aus dieser Inkonsistenzeigenschaft ergibt sich ein Problem $(A \to C) \land B$. Die Aussage $(A \to C)$ könnte genauso als die Negation von B definiert werden. Es ergäbe sich daraus eine Kontradiktion $(\neg B \land B)$, einer Formel die stets zu *falsch* ausgewertet wird. Über Wahrheitstafeln können jegliche (zusammengesetzte) Aussagen auf deren Wahrheitsgehalt überprüft werden. Damit verdeutlicht sich einfach, dass das Schlussfolgern über die aussagenlogische Formeln ein entscheidbares Problem darstellt.

3.2 Prädikatenlogik

Eine Erweiterung der Aussagenlogik stellt die Prädikatenlogik dar [HKR09, p. 6,367–373]. In der Prädikatenlogik werden Elementaraussagen hinsichtlich ihrer Struktur untersucht. Mit Prädikaten können Eigenschaften über Individuen festgelegt werden. Die Anzahl der Stellen im Prädikat ist dabei nicht beschränkt, so können mit 1-stelligen Prädikaten Zugehörikeiten (2), mit 2-stelligen Rolleneigenschaften (2) modelliert werden Quantoren geben an, von wie vielen Individuen eine Aussage erfüllt wird, so kann die Aussage über die Existenz eines Individuums getroffen werden (4).

$$TextInput(x) \tag{2}$$

Abb. 3. Prädikatenlogik erster Stufe: 1-stelliges Prädikat als Klasse

$$hasName(x, \text{agb}) \tag{3}$$

Abb. 4. Prädikatenlogik erster Stufe: 2-stelliges Prädikat als Rolleneigenschaft

Somit lassen sich wesentlich mächtigere Aussagen treffen, die mit der Aussagenlogik allein nicht möglich sind. Das hat allerdings auch Asuwirkungen auf die Laufzeit des Schlussfolgerns. Das Entscheidungsverfahren für die Prädikatenlogik ist unentscheidbar.

3.3 Beschreibungslogiken

Eine entscheidbare Untermenge der Prädikatenlogik erster Stufe bieten die Beschreibungslogiken [HKR09]. Im Wesentlichen enthält eine Beschreibungslogik die Operatoren der Aussagenlogik ($\wedge, \vee, \neg$), ein- und zweistellige Prädikate sowie eine eingeschränkte Quantifizierung. Sie umfassen Individuen (Objekte), Rollen (2-stellige Prädikate) und Klassen (1-stellige Prädikate), wobei keine Metamodellierung (Klassen von Klassen) möglich ist. Normalerweise wird die Wissensbasis dabei in 2 Teile separiert, *TBox* und *ABox*. Das Terminoligische Wissen (*TBox*) beschreibt die Struktur möglicher Welten. Es stellt Begriffe zur Beschreibung von Sachverhalten bereit. So definiert es das Vokabular in einem Anwendungsbereich. Das Grundvokabular besteht aus den Klassen (einstellige Prädikate) und Rollendefinitionen (zweistellige Prädikate). Das Assertionale Wissen beschreibt die Sachverhalte, die in einer bestimmten Welt gelten. Es existieren Annahmen (assertions) über einzelne Objekte ausgedrückt mit Hilfe des Grundvokabulars.

Die Syntax ist stark an die Prädikatenlogik angelehnt. Es gibt jedoch wesentliche Unterschiede. Hervorzuheben ist der Wegfall der Implikation ($\rightarrow$). Während in der Prädikatenlogik auch Rollenaxiome rechterhand der Implikation stehen können (2), sind bei Beschreibungslogiken nur Klassen in Subsumptionen möglich $C_1 \sqsubseteq C_2$. Nur ein Teil der Prädikatenlogik ist demnach adäquat in eine Beschreibungslogik zu übersetzen, man spricht in diesem Fall von DL-safe Regeln.

Eine Auflistung der Syntax und der Entsprechung in Prädikatenlogik sowie OWL finden Sie in 5.

Mit dem RL-Profil aus OWL 2 [W3C12] gibt es eine Sprache, die die glei-

$$\forall x \exists y. TextInput(x) \rightarrow hasName(x, y) \tag{4}$$

Abb. 5. Prädikatenlogik erster Stufe: Quantoren als Existenz-Aussagen

che Mächtigkeit wie die Beschreibungslogik ($\mathcal{SHOIN}(D)$) besitzt und demnach für das Reasoning in entscheidbarer Komplexität bestens geeignet ist. Sie hat ein gute Balance zwischen Ausdrucksfähigkeit und Berechenbarkeits-/Verarbeitungseigenschaften. OWL 2 ist zu dem Empfehlung des W3C und es existieren bereits einige Reasoner-Implementierungen. Die bekanntesten derzeit sind Fact++, Hermit, KAON2 und Pellet.

4 Implementierung

[Bri14] untersuchte inwieweit das Analyse-Tool [Rhe14] verwendet werden kann um eine objektive Beurteilung über die Benutzerfreundlichkeit einer Webseite geben zu können. Die Arbeit zeigt gute Ansätze und setzt auf die Konvertierung der erhobenen Daten in ein OWL-Dokument. Mit einem anschließenden Projekt können mit einem durchdachten Regelsystem Schlussfolgerungen gezogenwerden. Das Programm ist als Dienst mit HTML-Oberfläche gestaltet und kann lediglich die Webseiten-Daten aus *AbRUPt* umwandeln, die Integration der Webseiten-Besucherdaten ist noch nicht berücksichtigt. Auch die Integration in nachfolgende Projekte erweist sich durch den wenig modularen Aufbau als eher schwierig. Mit Einbezug der Ergebnisse aus [**?**] konnte ein Konzept für eine sinnvolle Aufbereitung der Daten gezeigt werden. Es ist OWL-Converter entstanden, der erhobene *AbRUPt*-Daten von einem XML-Format in ein OWL-Dokument transformiert.

4.1 Anpassungen und Erweiterung des OWL-Converters

Einige wichtige Aspekte wurden gegenüber dieser Arbeit geändert. Die Ontologie für den *AbRUPt-Service* wurde von Grund auf intuitiver gestaltet und um die Konzepte der Webseiten-Besucherdaten erweitert (12). Dabei wurde ebenso auf sinnvolle Konsistenzeigenschaften sowie Inferenz-Regeln zurückgegriffen, die auf Usabilty-Probleme schlussfolgern. (5) Ein Problem ergab sich für die Eindeutigkeit von Webseiten-Elementen, die nicht kontextbasiert benannt wurden. Mit dem an *REST* [FT00] angelehnten URI-Aufbau ergeben sich global einheitlich URIs. Dabei wird der Kontext in die URI-Struktur jeweils eingebettet (Listing 1). Ein Formular-Element kann somit auf mehreren Webseiten mit gleicher Id, Name und sonstigen Attributen koexistieren und wird nicht zu dem selben OWL-Element konvertiert. Beispielsweise liegt dann das Input-Textelement des Namens für ein Kontaktformular innerhalb von Formular (Form = *kontaktformular*), Seitenstatus (State = *state5*), Seiten-URL (Page = *http://www.rikscha−mainz.de/Kontakt*) und der zugehörigen Website (Website = *http://www.rikscha−mainz.de*). Das sieht zunächst nicht sehr lesbar aus, ist aber unvermeidbar aufgrund möglicher großer Datenbestände.

```
1 @prefix abrupt: <http://wba.cs.hs-rm.de/AbRUPt/> .
2
3 abrupt:Website/http://www.rikscha-mainz.de/Page/http://www.rikscha-mainz.de/
      Kontakt/State/state5/Form/kontaktformular/Text/name a abrupt:Text;
4    abrupt:class "text";
```

```
5    abrupt:id "name";
6    abrupt:maxlength 60;
7    abrupt:name "name";
8    abrupt:placeholder "Vorname Nachname";
9    abrupt:size 40;
10   abrupt:type "text" .
```

Listing 1. Nested URIs

Für die Einstufung von Problemen gibt es die Klasse *Problem*. Durch geeignete Abfragen, bspw. mit SPARQL **??**, kann nach dem Reasoning die Liste aller möglichen Probleme abgerufen werden. Wie bereits angesprochen können dabei Elemente dazugehören, die aus einem subjektiven Erscheinungsbild annehmbar als Problem ignoriert werden können.

4.2 Eingesetzte Technologien

Für die Neugestaltung des Konvertierung-Tools fiel die Wahl der Technolgie auf *RubyGems*, dem offiziellen Paketsystem von Ruby . Zum einen ist die Unterstützung durch ruby-rdf [BLK15] für RDF/OWL ausgezeichnet und zum Anderen erweisen sich noch weitere wesentliche Vorteile:

- Benutzung durch Kommandozeilentool
- Modularer Aufbau
- Ausgezeichnete Komponententest-Unterstützung
- Wiederverwendbarkeit durch Einbindung als RubyGem Bibliothek oder des Kommandozeilentools
- Flexibilität der Eingabe und Ausgabe (bestimmte Services, nur Abrupt/ohne Benutzerdaten, tutrle, rdfxml, n3

Das terminologische Wissen (TBox) ist in einem OWL-Dokument gesichert, sodass es separat mit Protege entwickelt und in die Software einfach eingebunden wird. Da es sich ausschließlich um statische Inhalte handelt, erweist sich dies als sinnvoll. Zur Hervorhebung und Veranschaulichung von nicht-trivialen Regeln existieren weitere OWL-Dokumente, die als Ordnerinhalte der Software hinzugefügt werden. Mit dem Befehl in Listing 9 lässt sich Konvertierung starten. Zunächst werden die statischen Inhalten in ein RDF-Repsoitory geladen. Das Assertionale Wissen (ABox) entsteht aus der Transformation der Eingabedaten. Die Eingabedaten liegen in der Regel in 2 XML-Dokumenten vor - den Website- und den Besucher-Daten. Für jeden Kontentyp existiert eine Schema-Datei, die für die nötige Datentypkonvertierung sorgt. Dadurch werden aus den XML-Rohdaten die nötigen Werte umgewandelt, bspw. Integer, Date, Decimal, ... Anschließend erstellt eine zugewiesene Transformater-Klasse aus den XML-Daten die RDF-Statements zu dem RDF-Repository hinzu. Der Aufbau der Software ist somit sehr modular, individuell anpassbar, durch Oberklassen generisch und übersichtlich.

5 Schließen auf Probleme mit Inferenz-Regeln

Die formale Beschreibungssprache OWL 2 ist eine Empfehlung des W3C [W3C04a] vom 11. Dezember 2012 für die Beschreibung von Ontologien. Primär geht es darum Modellierungen einer Weltanschauung (Domäne) vorzunehmen, sodass Mensch und Maschine die Bedeutung verarbeiten können. OWL 2 basiert technisch auf RDF und RDFS und geht dabei über die Ausdrucksmächtigkeit weit hinaus. Es ist dokumentenbasiert, was die Erstellung und Verteilung gegenüber einem Binärformat enorm vereinfacht. Um die richtige Balance zwischen Ausdrucksstärke und Entscheidbarkeit bzw. Performance zu gewährleisten gibt es 3 verschiedene Sprachebenen - Lite, DL und Full. In dieser Arbeit wird der Fokus auf die 2. Sprachebene OWL 2 DL gelegt, für welche es 3 Profile existiert. Das OWL 2 RL Profil ist auf Anwendungen, die ein skalierbares Reasoning verlangen, ohne dabei zu viel Ausdruckskraft zu verlieren, ausgerichtet. Die Ausdrucksmächtigkeit entspricht dabei der Beschreibungslogiken $\mathcal{SHOIN}(D)$. Je nach Anwendung der Axiome ergibt sich das jeweilige Profil, es wird nicht für das Dokument definiert. Man ist daher darauf bedacht auf gewisse OWL-Axiome zu verzichten um nicht in ein ungewünschtes Profil zu gelangen.

Prädikatenlogik erster Stufe	Beschreibungslogik ACL	OWL 2 RL
$C_1 \sqcap ... \sqcap C_n$	$C_1 \sqcap ... \sqcap C_n$	*intersectionOf*
$C_1 \sqcup ... \sqcup C_n$	$C_1 \sqcup ... \sqcup C_n$	*unionOf*
$\neg C$	$\neg C$	*complementOf*
$\{a_1\} \sqcup ... \sqcup \{a_n\}$	$\{a_1\} \sqcup ... \sqcup \{a_n\}$	*oneOf*
$\forall P.C$	$\forall P.C$	*allValuesFrom*
$\exists P.C$	$\exists P.C$	*someValuesFrom*
$\geq nP$	$\geq nP$	*minCardinality*
$\leq nP$	$\leq nP$	*maxCardinality*
$C_1 \sqsubseteq C_2$	$C_1 \sqsubseteq C_2$	*subClassOf*
$C_1 \equiv C_2$	$C_1 \equiv C_2$	*equivalentClass*
$C_1 \sqsubseteq \neg C_2$	$C_1 \sqsubseteq \neg C_2$	*disjointWith*
$\{a_1\} \sqsubseteq \{a_2\}$	$\{a_1\} \sqsubseteq \{a_2\}$	*sameIndividualAs*

Tabelle 1. Syntax der Beschreibungslogik und Entsprechung in der Prädikatenlogik sowie in OWL 2 RL

Eine vernünftige Unterteilung der Axiome ist in [AGH12] gegeben. Dort werden OWL 2 RL-Regeln in RIF [W3C13] beschrieben.

5.1 Inkonsistenz-Regeln

Für Rollenaxiome können mittels einfacher Regeln Definitions- und Wertebereich festgelegt werden. Als Definitionsbereich wird die Klassenzugehörigkeit des 1. Parameters von Rollen bestimmt. Im Beispiel des Rollenaxioms *hasPage* wird definiert, dass Seiten immer zu einer Website geören (Abb. 5, Listing 2).

$$\exists\ hasPage\ Thing \sqsubseteq\ Website \tag{5}$$

Abb. 6. Beschreibungslogik: Definitionsbereich-Regel für Rollen

```
abrupt:hasPage rdfs:range abrupt:Website .
```

Listing 2. Definitionsbereich-Regel für Rollen in OWL 2 RL

Für den Wertebereich, also der Festlegung der Klassenzugehörigkeit des 2. Parameters von Rollen ergibt sich Formel 6 sowie dessen Darstellung in OWL 2 RL (Listing 3).

$$\top \sqsubseteq \forall\ name\ Dataytpe(xsd{:}string) \tag{6}$$

Abb. 7. Beschreibungslogik: Wertebereich-Regel für Rollen

```
abrupt:name rdfs:domain abrupt:FormElement .
```

Listing 3. Wertebereich-Regel für Rollen in OWL 2 RL

5.2 Produktions-Regeln

Durch Inferenzen entstehen neue Klassenzugehörigkeiten. Aus Einbahnstraßenseiten (Seiten ohne Links) entstehen Problemfälle (Listing 4).

$$State \sqcap \leq 0\ hasLink.Link \sqsubseteq Problem \tag{7}$$

Abb. 8. Beschreibungslogik: Produktionsregel

```
[
    rdfs:subClassOf :Problem;
    owl:intersectionOf ( :State
    [   a owl:Restriction ;
        owl:onProperty :hasLink ;
        owl:onClass :Link;
        owl:maxQualifiedCardinality 0;
    ]
    )
] .
```

Listing 4. Produktionsregel in OWL 2 RL

$$hasPage \circ hasState \sqsubseteq hasStateThorugh \tag{8}$$

Abb. 9. Beschreibungslogik: Kettenregel

5.3 Listen-Regeln

Durch Listenregeln können Sammlungen genutzt werden um Restriktionen aufzubauen oder durch Kettenregeln Hilfsrollen zu aktivieren (Abb. 8, Listing 5).

```
:hasStateThrough a owl:ObjectProperty;
    owl:propertyChainAxiom ( :hasPage :hasState ) .
```

Listing 5. Kettenregl in OWL 2 RL

5.4 Datentyp-Regeln

Als mächtigstes Werkzeug erweisen sich die Datentypregeln, die es erlauben, aus Zahlenbereichseinschränkungen und Datentypdefinitionen Probleme abzuleiten. Der Lesbarkeitsindex einer Seite sollte einen definierten Wert nicht überschreiten um nicht als zu komplex zu gelten (Abb. 9, Listing 6).

$$readability \geq 8.0\ Datatype(xsd{:}decimal) \sqsubseteq Problem \tag{9}$$

Abb. 10. Beschreibungslogik: Datentypregel für Lesbarkeitsindex

```
[
    a owl:Restriction;
    rdfs:subClassOf :Problem;
    owl:onProperty :readability;
    owl:someValuesFrom [
        a rdfs:Datatype;
        owl:onDatatype xsd:decimal;
        owl:withRestrictions ([ xsd:minInclusive 8.0])
    ]
] .
```

Listing 6. Datentypregel in OWL 2 RL

Für die obere Schranke von HTML-Ankern mit niedriegen Kontrastverhältnissen empfiehlt es sich am besten keine dieser Form darszustellen. Dementsprechend ist die Anzahl auf kleiner als 1 beschränkt. Alle Werte darüber stellen ein Probelm dar (Abb. 10, Listing 7). Gleiches gilt für Textknoten mit niedrigen Kontrast und Paragraphen mit zu vielen Zeilen.

```
[
    a owl:Restriction;
    rdfs:subClassOf :Problem;
    owl:onProperty :aTagWithLowContrast;
    owl:someValuesFrom [
        a rdfs:Datatype;
        owl:onDatatype xsd:integer;
```

$$\begin{aligned} aTagWithLowContrast \geq 1\ Datatype(xsd{:}integer) &\sqsubseteq Problem \\ textNodesWithLowContrast \geq 1\ Datatype(xsd{:}integer) &\sqsubseteq Problem \\ paragrahpsWithTooLongLines \geq 1\ Datatype(xsd{:}integer) &\sqsubseteq Problem \end{aligned} \tag{10}$$

Abb. 11. Beschreibungslogik: Datentypregel für Kontrastverhältnisse

```
        owl:withRestrictions ([ xsd:minInclusive 1])
    ]
] .

[
    a owl:Restriction;
    rdfs:subClassOf :Problem;
    owl:onProperty :textNodesWithLowContrast;
    owl:someValuesFrom [
        a rdfs:Datatype;
        owl:onDatatype xsd:integer;
        owl:withRestrictions ([ xsd:minInclusive 1])
    ]
] .

[
    a owl:Restriction;
    rdfs:subClassOf :Problem;
    owl:onProperty :paragrahpsWithTooLongLines;
    owl:someValuesFrom [
        a rdfs:Datatype;
        owl:onDatatype xsd:integer;
        owl:withRestrictions ([ xsd:minInclusive 1])
    ]
] .
```

Listing 7. Datentypregeln in OWL 2 RL

5.5 Erweiterungen

Die Implementierung durch einen Regelsatz mit OWL 2 RL [W3C12] ist eine ansehnliche Vorgehensweise für das Auffinden von Problemen. Für viele Anwendungsfälle muss man dennoch auf Erweiterungen zurückgreifen. Die Beschreibungslogik $\mathcal{SHOIN}(D)$ weist zwar eine vernünftige Mächtigkeit auf, lässt dabei gegenüber konventioneller Programmiersprachen die Arithmetik völligst außer Acht. So ist es nicht möglich Summen über Elemente, Durchschnittswerte, Minima und Maxima zu ermitteln. In konventionellen Datenbanken gilt die Annahme der Weltabgeschlossenheit (Closed World Assumption), man kann aus den vorhandenen Daten ableiten, welche Daten nicht vorhanden sind. Das größte Element ist also direkt ablesbar, weil kein anderes größeres Element dieses übersteigt. In der Wissensdatenbank beschrieben durch OWL 2 gilt diese Annahme nicht. Durch die fundierte Basis der Prädikatenlogik arbeitet OWL 2 RL auf der Annahme einer offenen Welt (Open World Assumption, [HKR09, p. 194]). Dies bedeutet, dass alles was nicht explizit als wahr beweisen werden kann, nicht automatisch als falsch bezeichnet wird. Das größte Element ist also nicht direkt ablesbar, weil ein anderes größeres (abgeleitetes) Element dieses übersteigen könnte. Eine Summe über alle Besucherzeiten kann nicht erfasst werden,

wenn davon ausgegangen werden kann, dass neue Besuche hinzukommen können. Das Überwinden dieses Problems stellt eine große Herausforderung dar, für die zur Zeit lediglich Lösungsansätze existieren.

In vielen Punkten geradlinig und elegant ist die Kombination aus OWL und RuleML – SWRL [W3C04b]. SWRL ist eine vom W3C entwickelte Spezifikation für eine Regelsprache in OWL 2 [W3C04a] mit einer großen Einschränkung, sie ist unentscheidbar. Mittels sogenannter DL-safe Regeln, also Regeln, welche die Einrschänkungen der Beschreibungslogiken einhalten, bietet SWRL die Entscheidbarkeit von OWL 2 RL (PTime) und die Mächtigkeit der Beschreibungslogiken inkl. arithmetischer Funktionen.

SWRL ist kein direkter Bestandteil von OWL 2 RL und funktioniert als Sprach-Erweiterung. Sie wird dennoch von vielen Reasonern unterstützt.

Will man ohne Erweiterung für OWL 2 RL auskommen, müssen Vorkehrungen bereits in der Vorverarbeitung getroffen werden, die die Einschränkungen der offenen Weltannahme aushebeln. Auf gängige Arithmetik $+, -, *, /AVG(), SUM()$ kann in der unterliegenden Programmiersprache zurückgegriffen werden. Dabei werden die eingelesen Daten in der Vorverarbeitung somit als abgeschlossen angenommen. Die Besuchsdauer einer Seite wird durch einfache Differenz (8) ermittelt und als weiteres Wissen der ABox hinzugefügt.

```ruby
# /lib/abrupt/transformation/client/visit.rb

def add_visit_duration
  leavetime = Abrupt.parse_time(@values[:leavetime])
  enterteime = Abrupt.parse_time(@values[:entertime])
  return unless leavetime || enterteime # maybe no recognized
    leavetime
  visit_duration = (leavetime - enterteime).to_f # in seconds
  add_data_property('visitDuration', visit_duration)
end
```

Listing 8. Erweiterung der TBox in der Vorverarbeitung

6 Related Work

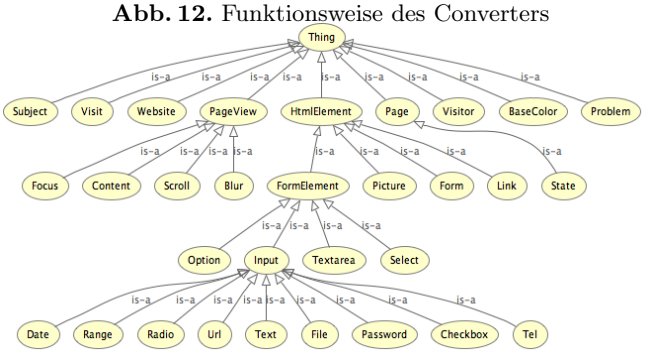
Diese Arbeit baut zu großen Teilen auf [Bri14] auf. Dort wird bereits ein OWL-Converter für die Daten aus dem AbRUPt-Service bereitgestellt. Eine Beschreibung für die Daten aus Readability, Subject, Input, Complexity, Picture und Link ist dort ebenso gegeben sowie der Funktionsweise des Crawlers und Analyzers. Die Readability-Berechnung basiert auf [Mar12b]. Dort wird anhand beliebter Webseiten eine Formel für den Lesbarkeitsindex von Texten erforscht. Eine übersichtliche Gestaltung für relevante Faktoren dieses Indexes bietet [Sch03]. In [Mar12a] ist die Funktionsweise zur Klassifizierung von Themengebieten eines Textes erläutert. Durch geeignete Filter und Selektion von Wörtern und deren Häufigkeit kann mittels dem Datenbestand von Wikipedia eine aussagenkräftige Auswahl an Schlagwörtern vorhergesagt werden. Für die Feststellung von Diskrepanzen zwischen erwareteter und tatsächlicher Bearbeitungsdauer von

Web-Formularen liefert [Mar14] zeitliche Richtwerte. Durch Analysen ergeben sich erwartete Bearbeitungszeiten kategorisiert nach Eingabetyp und Eingabegerät, bspw. Input-Elemente auf mobilen Endgeräten oder Desktop-PCs. Für das Schlussfolgern ist die Wahl eines geeigneten Reasoners nich nicht getroffen. Eine Übersicht sowie Ansätze für Hybrid-Lösungen aus Tableau und Resolution gibt [SSD11].

7 Zusammenfassung

Die Struktur des Converters wurde im Wesentlichen überarbeitet. Es wurde viel Wert auf die intuitive Verwirklichung des Grundvokabulars der AbRUPt-Ontologie gelegt (Abb. 12).

Abb. 12. Funktionsweise des Converters



Außerdem wurde der Graph um die Konzepte der Besucherdaten erweitert. Der vorherige Converte war wenig modular und umfasste ca. 2000 Zeilen in nur 4 Dateien. Die Konfiguration war im Code eingebettet, sodass der Code insgesamt nciht sehr gut wartbar ist. Die Neuimplementierung sieht einen wesentlichen modulareren Aufbau vor, sodass Anpassungen in der Vorverarbeitung effizient einsetzbar ist. Sie berücksichtigt außerdem die Transformation der Daten in die korrekten Datentypen (Integer, Decimal, Boolean, ...). Das Grundvokabular ist komplett separat entwickelt und wird in die Konvertierung miteinbezogen. Dadurch ist der Zugriff vereinfacht und mittels eigener Entwicklungstools die Ontologie als Graph darstellbar. Trotz der vielen Anpassungen und Erweiterungen umfasst der Quellcode nun gerade mal ca. 900 Zeilen und 15 Klassen inkl. rudimentären Testcode. Man kann das Tool mittels Kommandozeiletool (9) ausführen oder in ein Ruby-Projekt als Bibliothek einbinden.

```
1 abrupt convert website_data.xml visitor_data.xml --output out.
    ttl --format turtle
```

Listing 9. Beispielnutzung des Kommandozeilentools

Die Implementierung durch einen Regelsatz ist eine anschauliche Vorgehensweise für das Auffinden von Problemen. Ohne die Vorverarbeitung des Converters sind der Arbeit allerdings aufgrund der Open World Assumption Grenzen gesetzt. Für das Überwinden der Grenzen, die in einer offenen Weltannahme entstehen, müssen die Eingabedaten vor der Konvertierung als geschlossen angenommen werden um vernünftiges Hilfswissen zu generieren. Nur so ist man mit OWL 2 RL in der Lage mit einem Regelsatz auf Probleme zu schließen. Die Generierung von Regeln ist nicht immer ganz einfach, da mit der Beschreibungslogik eine abgespeckte Mächtigkeit zur Verfügung steht. Diese zwingen den Entwickler oft zu einer bestimmte Vorgehensweise. Anders als in konventionellen Programmierparadigmen gibt es nur wenige Lösungswege. Dafür ist das Regelwerk Bestandteil des OWL-Dokuments und ist als solches für Mensch und Maschine lesbar und einfach austauschbar. Man spart und verteilt Rechenzeit, da das Schlussfolgern, die eigentliche Programmausführung, auf dem jeweiligen Rechner stattfinden kann. Theortisch ist es sogar eine Parallelisierung möglich. Der Converter muss in der Regel mit großen Dateien umgehen können. In der Entwicklung wurden Webseitendaten mit über 300 KB und Besucherdaten aus mit über 2 MB verwendet. Durch das Umwandeln und Ergänzen der Daten in das OWL-Dokument entsteht darus eine turtle-Datei mit über 30 MB. Dies entspricht einer Vergrößerung der Daten um den Faktor 15. Dementsprechend ist die momentane Verarbeitungszeit auch sehr lange. Der Flaschenhals liegt bei der Serialisierung des RDF-Grpahen durch die Ruby-Bibliothek "rdf-raptor"[BF14]. Das Regelwerk ist noch relativ bescheiden und ausbaufähig. Für die Datentyp-Regeln fehlen noch Schwellwerte. Diese zu definieren ist nicht trivial, da Texte kontextbezogen unterschiedliche Ansprüche hervorbringen. Die Inkonsitenz-Regeln dagegen bieten bereits eine weitgehend Abdeckung. In [Bri14] wurde bereits eine Web-Benutzeroberfläche (GUI) gestaltet, die den (alten) Converter anwendet. Für die neuen Erweiterungen, insbesondere der Einfluss der Besucherdaten ist diese GUI noch nicht vorbereitet. Eine Konfigurationsmaske für die kontextbezogenen Schwellwerte wäre denkbar. Das Reasoning über das Dokument ist im Projekt nicht berücksichtigt und wird derzeit über properitäre Software vollzogen. Eine Hybrid-Lösung könnte eine geeignete Wahl sein um die jeweiligen Vorteile der Klassifizierungs-Strategien zu nutzen. Um die Software zu einem Gesamtprodukt zu komplettieren, besteht die Herausforderung der Vereinigung von Crawler, Converter, Reasoner und einer Eingabe- und Ausgabemaske.

Literaturverzeichnis

[AGH12] Grigoris Antoniou, Paul Groth, and Frank Van Harmelen. *A Semantic Web Primer*. MIT Press, Cambridge, 3 edition, 2012.

[BF14] Arto Bendiken and John Fieber. rdf-raptor. 2014.

[BLK15] Arto Bendiken, Ben Lavender, and Gregg Kellogg. ruby-rdf. 2015.

[Bri14] Silas Markus Brieger. Aufbereitung und Umstrukturierung Webseiten spezifischer Daten in OWL, 2014.

[FT00] Roy T. Fielding and Richard N. Taylor. Principled Design of the Modern Web Architecture. In *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, pages 407–416, New York, NY, USA, 2000. ACM.

[HKR09] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. CRC Press, Boca Raton, Fla, 2009.

[KSH12] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. A description logic primer. *CoRR*, abs/1201.4089, 2012.

[Mar12a] Ludger Martin. Subject Classification of Web Pages. In *International Conference WWW/Internet 2012*, pages 298 – 306, 2012.

[Mar12b] Martin, Ludger and Gottron, Thomas. Readability and the Web. In *Future Internet*, pages 238 – 252, 2012.

[Mar14] Ludger Martin. A restful web service to estimating time requirements for web forms. In *International Journal of Knowledge and Web Intelligence*, volume 5, pages 62 – 75, 2014.

[Rhe14] Hochschule RheinMain. *Automated Reasoning for Web Usability Problems*. http://wba.cs.hs-rm.de/AbRUPt/service/, 2014.

[Sch03] Jürgen F. Schopp. Faktoren der Lesbarkeit. 2003.

[SSD11] Weihong Song, Bruce Spencer, and Weichang Du. Hybrid Reasoning for Ontology Classification. In *Proceedings of the 24th Canadian Conference on Advances in Artificial Intelligence*, Canadian AI'11, pages 372–376, Berlin, Heidelberg, 2011. Springer-Verlag.

[W3C04a] W3C. OWL Web Ontology Language Reference. 2004.

[W3C04b] W3C. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. 2004.

[W3C12] W3C. OWL 2 Web Ontology Language Profiles. 2012.

[W3C13] W3C. OWL 2 RL in RIF. 2013.