INCREASING ACCESSIBILITY OF WEB CONTENT VIA SEMANTIC RENARRATION

by

Emrah Güder

B.S, Computer Engineering, Işık University, 2006

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2016

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Dr. Suzan Üsküdarlıfor the continuous support of my M.S. study and research. Her patience, enthusiasm, and immense knowledge guided me all the time of my research and writing of this thesis. I've learned many things since I became Dr. Suzan Üsküdarlı's student.

Besides my advisor, I would like to thank T.B. Dinesh for his insightful comments, and discussions. In addition to discussions, readings he suggested throughout my study and research were incredibly helpful.

Last but not least, I owe more than thanks to my family members which includes my parents, elder brother and my wife for their support and encouragement throughout my life. In addition to them, I would like to mention our dog, Bulut, who helped me a lot during stressful times.

ABSTRACT

INCREASING ACCESSIBILITY OF WEB CONTENT VIA SEMANTIC RENARRATION

Much of the content on the Web is not accessible to a large portion of the global population due to barriers such as language, literacy levels, physical impairments, expertise, etc. Attempts to make content more accessible are made by those who translate or otherwise transform material for wider consumption. However, such efforts are often limited to a particular collections (i.e.Wikipedia) or by people who make it a point to create alternatives via blogs and web sites. There is need for a framework that supports the specification of providing alternative narrations (renarrations) in a form that is processable so that they can be located and otherwise utilized.

In this thesis we propose a framework for a crowdsourced approach to renarrating Web documents. This framework aims to support the creation of relations between Web resource elements. A renarration ontology is created for supporting this framework. Furthermore, it utilizes renarrations with external resources establishing relations among existing resources. Establishing relations has the potential of increasing the utility for further accessibility. A prototype for proof of concept has been built.

ÖZET

İÇERİK ERİŞİLEBİLİRLİĞİNİN ANLAMSAL AÇIKLAMALAR İLE ARTIRILMASI

İnternet ortamında bulunan bilgilerin birçoğu, dil, fiziksel engeller, okuryazarlık düzeyi, uzmanlık ve buna benzer birçok engel sebebiyle küresel nüfusun büyük bir kısmı için erişilememekte. İçeriği daha erişilebilir hale getirmek için girişimlere, tercüme edilmesi ya da değiştirilerek daha geniş kitlelerce ulaşılabilmesini sağlamaya çalışmak örnek olarak gosterilebilir. Ancak, bu yöntemler çoğu kez özel derlemeler (Wikipedia) ya da kişilerin bloglar ya da web siteleri üzerinden oluşturduğu alternatif içeriklerle sınırlıdır. Alternatif içeriklerin(renarration) oluşturulabildiği ve bu içeriklerin bulunabilmesinin yanısıra, kullanılabildiği bir altyapıya ihtiyaç bulunmaktadır.

Bu tezde, Web üzerindeki veriler için alternatif oluşturulmasına olanak veren bir altyapı tasarlanmıştır. Bu altyapı alternatif verilerin oluşturulmasına ek olarak, veriler arasındaki ilişkilerin de tutulmasına olanak vermektedir. Önerilen altyapı, tez kapsamında geliştirilen bir ontolojiyi kullanmaktadır. Bunlara ek olarak, alternatif veriler oluşturulurken harici kaynakların kullanılmasının yanısıra, bu kaynaklar arasındaki ilişkiler de altyapı kapsamında saklanmaktadır. Bu ilişkilerin saklanması, içeriklerin daha erişilebilir hale gelmesi adına büyük bir potansiyele sahiptir. Önerilen ontoloji ve altyapının ispatı adına bir prototip geliştirilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Listings

LIST OF ACRONYMS/ABBREVIATIONS

DAWG	RDF Data Access Working Group
FOAF	Friend Of A Friend Vocabulary Specification
JSON	Javascript Object Notation
RDF	Resource Description Framework
RDFS	RDF Schema
OWL	Web Ontology Language
W3C	World Wide Web Consortium
TSI	Turkey Statistical Institue
XML	Extensible Markup Language
URI	Uniform Resource Identifier

1. INTRODUCTION

1.1. Motivation and Problem Statement

The way that we communicate has changed a lot with the World Wide Web. The World Wide Web has also changed the way we think of computers. It wouldn't be wrong to say that they were used for numerical calculations only. Nowadays, we use computers for information processing such as preparing monthly reports by using database applications, or text processing, and for lots of other purposes as well.

Most of the data on the Web is consumable by human. The way it is consumed is that people seek for the data and use the information gathered and seek/get in touch with people, order products, etc. The connections between Web documents is established by links. Another way is to use search engines such as AltaVista, Google [2] or Yahoo [3]. Even though search engines are great tools for finding information and people, there are some serious problems such as: Low precision, low recall, sensitivity to the vocabulary, resulting just a web page, etc. Although the result of search is successful, it is just a web page, so the resulting data are ready by human to consume. One may conclude that information retrieval is performed by human rather than search engines. The main problem with that is the Web content cannot be easily interpreted by machines.

In the context of knowledge management, retrieving, accessing and maintaining data is crucial. Data play very important role from which new values can be created, and productivity can be increased. Even though there is lots of data, searching it still depends on keyword-based search engines. After data are extracted by using search engines, human time is required to process the returned data. Extracted data can also have inconsistencies in terminology and be outdated as well. Data mining plays very important role in creating new knowledge implicitly. However, this task is very hard when unstructured data are used.

Semantic Web [4] can assist in solving the problems mentioned above. The word "semantic" itself means "meaning". So, the fundamental difference between Semantic Web technologies and other technologies is that the Semantic Web is concerned with the meaning of data. This fundamental difference brings along completely different outlook on how to store, query and display information. Semantic Web aims to allow more advanced knowledge management techniques. In Semantic Web world, knowledge is organized according to conceptual models. By the use of automated tools, inconsistencies can be checked and new knowledge can be extracted. Instead of keyword-based search, data will be queried, and retrieved in a human friendly format. Since knowledge is presented in a structure, it is also possible to view parts of documents instead retrieving them as a whole. According to the W3C, "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries" [5]. The basis for the Semantic Web are computer-understandable descriptions of resources. The Semantic Web is all about shared terminology which is achieved through consistent use of URIs. URIs can be seen as the atoms of the Semantic Web.

The data on the Web are generally hidden in HTML files and amount of data is massive and continually growing. Due to the characteristics of the content, it is not accessible by huge amount of people due to a variety of barriers. The barriers to accessibility range from limited Internet connectivity, to physical impairment, linguistic differences, and also social, cultural and economic factors [6]. Therefore, it is crucial to create meaningful relations between Web elements and to provide accessibility. Annotation is one of the methods to increase content accessibility on the Web. Generally speaking, annotation means to attach data to some piece of data. Annotations can be done manually, semi-automatically or full-automatically. Manual annotations are created by people whereas semi-automatic annotations are based on suggestions. A research by Digital Enterprise Research Institute, National University of Ireland, Galway [7] differentiated three types of annotations: informal, formal and ontological.

Renarration is another method to increase content accessibility on the Web. It can be defined as the process of rewriting a web page or elements within the web page.

For instance, a paragraph which is written by a lawyer can be hard to understand by someone else even if the text is in the same language. So, more accessible version of this paragraph can be created, and this process is called renarration. W3C Director and inventor of the World Wide Web, Tim Berners-Lee, said that the Web fundamentally designed for all people. Therefore, it is crucial that the content should be accessible to everyone.

1.2. Proposed Solution

This thesis focuses on making contents on the Web accessible to a wider audience. We provide a framework which enables renarrating contents by creating semantic relations between URIs and contents of web pages. We investigated the Web contents and detected the concepts and relations between them. In order to represent relations between Web elements, an ontology has been created and terms are used to assist the proposed accessibility which is beyond the traditional accessibility that focuses on limitations of vision. We have used Web technologies and Semantic Web in order to give context for the renarration process.

In an XML [8] document, XPath is used to navigate through elements and attributes. Since all HTML documents can be considered as XML documents, XPaths can be used to address individual elements as well. We have used XPaths for defining parts of HTML documents. In order to represent relations between Web elements, we have used Web Annotation Data Model [1,9,10] as a starting ontology. This ontology is used in semantic annotation(SemAnn) framework. A prototype has been implemented for semantic annotation process. The prototype allows users to select individual elements in Web pages, and create annotations for them. Created annotations are stored in a database, and can be queried through MongoDB collections.

Another framework has been designed for renarration of Web pages. The input to renarration framework(SemRen) is the output of SemAnn framework. The framework is not just bounded to stored annotations, but also users can create content during renarration process. Using the implemented prototype all annotations can be listed for

an element in a Web page, and the element can be renarrated by using annotations and/or other content on the Web.

In order to test the proposed ontologies and the prototype, different kinds of annotations and renarrations are created. The usability of the framework is evaluated in Chapter 5. Experiments show that the proposed ontology can be used to define relations between elements of Web pages in order to increase content accessibility. In addition to the ontologies proposed, it is also evaluated that the prototype implementation can be used to define for semantic annotation and renarration.

1.3. Structure of this thesis

The structure of this thesis is as follows. Chapter 2 gives background information about technologies used for semantic web. In addition to the technologies, related work will be discussed in detail. Different models and applications for both annotation and renarration will be researched. Chapter 3 presents a detailed model for semantic annotation. Chapter 4 provides implementation decisions in detail. Chapter 5 presents the evaluations of semantic annotation, and renarration models and implementation. Finally, Chapter 6 presents conclusions and discusses ideas for future work.

2. BACKGROUND AND RELATED WORK

In this section, fundamental knowledge about the subjects mentioned in our thesis is given. General information about Semantic Web technologies, the definition of an annotation, and annotation related studies are introduced. After that, the definition of renarration and the renarration approach for accessibility problem are provided. Finally, applications for increasing accessibility of content are provided.

2.1. Semantic Web Technologies

2.1.1. XML

HTML (hypertext markup language) is the standard language which is used to create Web pages. It was derived from SGML (standard generalized markup language) which is an international standard for representing information both human-readable and machine-readable. Since applications which conform to SGML are called SGML applications, HTML is such an application. The reason why HTML is used instead using SGML is that it was considered too complex for Internet-related purposes [11]. XML is also an SGML application which stands for extensible markup language. It is defined W3C's XML 1.0 Specification.

An XML document consist of a declaration, a number of elements, properties and comments.

2.1.1.1. XML declaration. XML documents may contain a declaration which is an information about the XML document itself.

<?xml version="1.0" encoding="UTF-8"?>

Above is an example of XML declaration which is a processing instruction that identifies the document as being XML. It specifies that the document is an XML document, the version and the character encoding used. The character encoding is not mandatory but it is a good practice. In XML declaration part, it can also be specified that the XML document is self-contained or not. If an XML document is a self-contained document then it can be said that the document doesn't refer to external documents.

2.1.1.2. Elements. XML elements are used to represent things in XML documents. An element consist of a start tag, content and end tag. For example, the following shows an element which is used to define a university name.

<university>Boğaziçi University</university>

2.1.1.3. Attributes. It is not mandatory that elements should have content; there can also be empty elements. An empty element is not meaningless, because it may have some attributes. An attribute is a markup construct which consists of name/value pairs. Attributes exist within a start tag or empty element tag. The following is an example of an element named as university with one attribute.

<university page="http://www.boun.edu.tr">Boğaziçi University</university>

An XML document can be considered as well-formed [12] when it respects certain rules. However, these rules say nothing about the structure of the document. It is very important to know the structure of documents when different applications try to communicate [11]. The structure of the documents is defined in documents which have all elements and attributes an XML document can use. The documents which define the structure are also XML documents. There are two ways to define the structure. DTDs [13], the older way and using XML Schema. XML Schema offers a richer language for defining the structure.

2.1.2. RDF

XML is a markup language which is used for interchange of data between applications. However, XML express the meaning of data. In Figure 2.1, even though the same information is represented, there are two different formalizations. This can also be expressed as there is no standard way of expressing information.

```
1 <person name="Karl Marx">
2  <wrote>Das Capital</wrote>
3 </person>
4
5 <book name="Das Capital">
6  <writtenBy>Karl Marx</writtenBy>
7 </book>
```

Figure 2.1. XML formalization for Das Capital written by Karl Marx

RDF(Resource Description Framework) is a standard model for data interchange on the Web [14]. It extends the linking structure of the Web to use URIs to name the relationship between things and two ends of the links. This is usually referred to as "triple".

The fundamental concepts of RDF are resources, properties and statements.

2.1.2.1. Resources. A resource is an object, a "thing" for which it is desired to be talked about. For example, resources are person and book (Lines 1 and 5) in Figure 2.1. Every resource has a URI(Uniform Resource Identifier)

2.1.2.2. Properties. Properties are special kinds of resources which are used to describe relations between resources. Considering Figure 2.1, properties would be "wrote" and "writtenBy". (Lines 2 and 6)

2.1.2.3. Statements. A statement is used to define relations between subject and object. RDF statements are also known as RDF triples. An example of statement can be seen in Figure 2.2.

```
<?xml version="1.0" encoding="UTF-16"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xlmns:mydomain
    ="http://mydomain.com">
 <rdf:Description rdf:about="Das Capital">
  <mydomain:writtenBy rdf:resource="Karl Marx"/>
 <rdf:Description>
<rdf:RDF>
```

Figure 2.2. XML-based syntax of RDF for Das Capital

In Figure 2.2, the first line specifies that XML is used. *rdf:Description* element makes a statement about the book "Das Capital". The relation, *writtenBy* between object, "Karl Marx", is defined in *http://www.mydomain.org/my-rdf-ns* which is a namespace. The difference between plain XML formalization and this one is that the relation, writtenBy, has semantics. In other words, applications use the statements in Figure 2.2 will know that *writtenBy* means to write a book and the resource connected to this relation would be a person.

2.1.3. RDF Schema

Resources can be described using RDF; however, RDF does not make any assumptions about any particular domain. This can be done using RDFS (RDF Schema) [15]. RDFS provides the framework to describe application-specific classes and properties.

A *class* can be thought as a set of elements and objects which belong to a particular class are referred as *instances* of that class [11].

Using classes are not enough to describe particular domains, but also restrictions should be defined. For example, considering two statements below, both of them are using the same property, *"writtenBy"*. Although, the first statement makes sense, the second one isn't correct. Restrictions can be defined using *domain* and *range*.

Das Capital is written by Karl Marx.

Karl Marx is written by Das Capital.

In order to restrict above property, domain should be restricted to be a book, and range should be restricted to be only a writer. In Figure 2.3, we present a simple ontology for books and writers. In the example ontology, a property "*writtenBy*" is defined to model books written by writers, and it's restricted to *book* and *writer* classes.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

 <rdfs:Class rdf:ID="book">
  <rdfs:comment>
   The class of books. All books should be written by someone
  </rdfs:comment>
 </rdfs:Class>

 <rdfs:Class rdf:ID="writer">
  <rdfs:comment>
   The class of writers
  </rdfs:comment>
 </rdfs:Class>

 <rdf:Property rdf:ID="writtenBy">
  <rdfs:comment>
   It relates books to writers
  </rdfs:comment>
  <rdfs:domain rdf:resource="#book"/>
  <rdfs:range rdf:resource="#writer"/>
 </rdf:Property>
</rdf:RDF>
```

Figure 2.3. Simple ontology for books and writers using XML based RDF syntax

2.1.4. OWL

The W3C Web Ontology Language (OWL) is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things [16]. OWL is a logic-based language which can be used to express knowledge that computer programs can process.

The expressivity of RDF and RDF Schema is very limited. RDF is limited to binary ground predicates, and RDF Schema is limited to a subclass and property hierarchies.

OWL documents are called OWL ontologies and are RDF documents [11]. The root element of an OWL ontology is an *rdf:RDF* element, which can be used to specify namespaces used in the ontology. Using OWL language, classes can be defined with *owl:Class* element. It can also be expressed that the class is disjoint or equivalent with another class where this expressivity doesn't exist in RDF Schema. OWL language has two predefined classes which are *owl:Thing* and *owl:Nothing*. The former is the most general class and every class defined with OWL language is a subclass of the class. The latter is an empty class. Thus every class is a superclass of *owl:Nothing*.

In addition to property elements in RDF Schema, OWL language offers more capable properties. For example, inverse properties can be related. Also, with *rdfs:subClassOf* it can be specified that a class C to be subclass of C'. This means every instance of C is also an instance of C'. However, restrictions cannot be defined with RDF Schema. For instance, suppose that a course should be taken only after a course is taken. Such a restriction can be declared using *owl:Restriction*.

OWL Language also includes some additional properties like cardinalities on restrictions, transitive, symmetric, functional and inverse functional properties, and boolean combinations like *owl:unionOf*, *owl:intersectionOf*, *owl:complementOf*.

In Figure 2.4, the ontology presented in Figure 2.3 is declared using OWL. In addition to class declarations, new properties are also defined which OWL supports. Classes are distincly defined; in other words, intersection of *book* and *writer* classes is empty. This is done using *owl:disjointWith*. Also, another object property is declared just by using *owl:inverseOf* which means *writes* object property is inverse of *writtenBy* object property. This implies that for *writes* object property *rdfs:domain* can be an instance of *writer* class and *rdfs:range* can be an instance of *book* class.

2.1.5. SPARQL

SPARQL is an RDF query language which can be used to retrieve and manipulate data stored in Resource Description Framework(RDF) format. It was made a standard

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
                  xmlns:owl ="http://www.w3.org/2002/07/owl#">

 <owl:Class rdf:ID="book">
  <rdfs:comment>
   The class of books. All books should be written by at least one writer
  </rdfs:comment>
  <owl:disjointWith rdf:resource="#writer"/>
   <rdfs:subClassOf>
        <owl:Restriction>
         <owl:onProperty rdf:resource="#writtenBy"/>
         <owl:someValuesFrom rdf:resource="#writer"/>
        </owl:Restriction>
  </rdfs:subClassOf>
 </owl:Class>

 <owl:Class rdf:ID="writer">
  <rdfs:comment>
   The class of writers
  </rdfs:comment>
  <owl:disjointWith rdf:resource="#book"/>
 </owl:Class>

 <owl:ObjectProperty rdf:ID="writtenBy">
  <rdfs:comment>
   It relates books to writers
  </rdfs:comment>
  <rdfs:domain rdf:resource="#book"/>
  <rdfs:range rdf:resource="#writer"/>
 </owl:ObjectProperty>

 <owl:ObjectProperty rdf:ID="writes">
  <owl:inverseOf rdf:resource="#writtenBy"/>
 </owl:ObjectProperty>
</rdf:RDF>
```

Figure 2.4. Simple ontology for books and writers using XML based RDF syntax and OWL

by the DAWG(RDF Data Access Working Group) of World Wide Web Consortium, and is recognized as one of the key technologies of the semantic web [17].

SPARQL query language is based on matching graph patterns. Graph pattern used in SPARQL is very similar to an RDF triple but with the possibility of a variable instead of an RDF term. Variables can be used to point subject, predicate or object of the pattern.

In Figure 2.5, foaf prefix is used for "friend-of-a-friend" ontology. The query returns names of every person in the dataset. The query references the subject with the variable name *?person*. "*?person a foaf:Person*" states that the subject is a *foaf:Person*. Another triple, "*?person foaf:name ?name*" states that the same person, because the subject is referenced with the same variable name, has a name. The query returns just names of persons since *?name* variable is the only variable selected.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
}
```

Figure 2.5. Simple SPARQL query selecting foaf:name data property for instances of foaf:Person

SPARQL uses namespace mechanism to define prefixes and namespaces so that queries can be simpler and shorter. In Figure 2.5, *foaf* (Line 1) is used as a namespace for friend-of-a-friend ontology.

As in SQL, SPARQL uses select-from-where structure. SELECT specifies the order of retrieved data. FROM is used to specify the source being queried. Finally, WHERE is used to impose constraints in the form of graph pattern templates.

2.1.6. JSON-LD

JSON is a useful messaging and data serialization format. The data model used for JSON-LD [18] is a labeled, directed graph which contains nodes which are connected by edges. It is easy to parse and generate JSON data.

For example, one line of JSON document is obvious for a human to interpret that this is a name of a person. A machine wouldn't be able to understand the meaning of the document.

"name" : "Karl Marx"

Linked data and Web in general, uses IRIs for unambiguous identification. IRIs to assign unambiguous identifiers to data that may be of use to other developers [18]. For instance, instead of using just name, one can use schema.org specification [19] so that above could be unambiguously expressed.

"http://schema.org/name" : "Karl Marx"

JSON-LD is a lightweight Linked Data format. It is easy for humans to read and write. It is based on the already successful JSON format and provides a way to help JSON data interoperate at Web-scale.

2.2. Annotation

Semantic Web enables machines to interpret, combine and use data on the Web. Whereas the current Web is understandable mostly by humans, but Semantic Web can be used by computers as well. The basis for the Semantic Web are computer understandable description of resources. Such descriptions can be created by annotating resources with metadata [7].

Several tools, paradigms and models exist to create annotations of Web resources. In this section, we analyze the Web Annotation Data Model which is a interoperable framework for creating associations between related resources and uses a methodology that conforms to the architecture of the World Wide Web. After analyzing the model, we talk about some applications to create annotations. Lastly, we will talk about studies related to annotations.

2.2.1. Web Annotation Data Model

The primary problem about creating annotations on the Web, is that user-created annotations cannot be shared or reused due to lack of common approach to expressing them. The Web Annotation Data Model species an interoperable framework for creating associations between related resources, annotations, using a methodology that conforms to the Architecture of the World Wide Web. [1]

In the model, an annotation is considered to be a set of connected resources, including a body and a target where the body is somewhat related to the target. Other possible relationships include choosing a representation of a resource, enabling content to be embedded within annotation, selecting segments of resources and defining styling hints for clients. In Table 2.1 namespaces used in specification are listed.

2.2.1.1. Core Annotation Framework. An Annotation can be expressed as the relationship between two or more resources using an RDF graph. Three basic resources, the Annotation itself, which is used to identify the concept of this relationship, the Body and the Target, are described in the RDF graph. The Body and Target resources can be any media type.

The Data Model states that an Annotation is a web resource and should have an HTTP URI. All annotations must be instances of the class oa:Annotation. Annotations may also have provanence information such as who created them.

Table 2.1. Namespaces used in the Web Annotation Data Model [1]

Prefix	Namespace	Description
oa	http://www.w3.org/ns/oa#	The Web Annotation Data Model
dc	http://purl.org/dc/elements/1.1/	Dublin Core Elements [20]
dcterms	http://purl.org/dc/terms/	Dublin Core Terms [21]
dctypes	http://purl.org/dc/dcmitype/	Dublin Core Type Vocabulary [22]
foaf	http://xmlns.com/foaf/0.1/	Friend-of-a-Friend Vocabulary [23]
prov	http://www.w3.org/ns/prov#	Provenance Ontology [24]
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	RDF [25]
rdfs	http://www.w3.org/2000/01/rdf-schema#	RDF Schema [15]
skos	http://www.w3.org/2004/02/skos/core#	Simple Knowledge Organization System [26]

The model also allows tagging of resources either with a text or with a URI. The tag is represented as the Body of Annotation, and the resource being tagged is the Target. For example, "Istanbul" can be associated with an image of the most crowded city of Turkey to describe what is depicted. If a resource is being tagged, the Body should have the *oa:Tag* class assigned to it because of applications may render tags in different ways.

For semantic tags, the tag is expressed with a URI. The example above, can use *http://dbpedia.org/page/Istanbul* as the Body. The model doesn't recommend to use the URI of a document as a Semantic Tag. The reason is that it might also be used as a regular Body in other annotations. Instead, it recommends to create a new URI and link it to the document. Using *foaf:page* is recommended for semantic tagging. If the URI does not have a representation and the URI truly identifies the concept itself, the data model recommends using skos:related property.

An Annotation may not have a Body resource. An example to this would be bookmarking a particular resource or highlighting a section of a resource. On the contrary, it is also possible for an Annotation to have multiple Bodies and/or Targets. Having multiple bodies and/or targets is allowed as long as dropping any of them would not invalidate the Annotation's meaning. Examples may be comparing Targets, ordering the Bodies, etc.

It is important for clients to understand when the Annotation is created and what software is used to serialize the model, and the creator of the Annotation. Provenance information can be attached to the Annotation, Body, Target any other resource in the RDF graph.

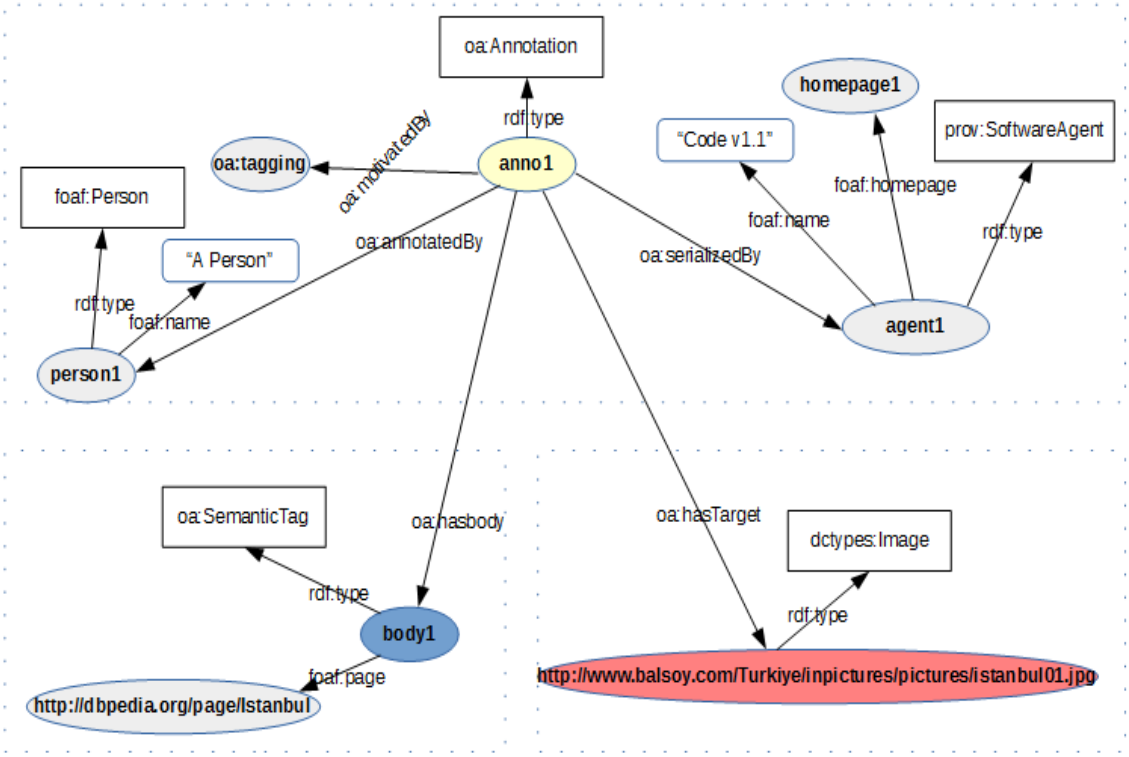

Figure 2.6. An annotation example of Semantic Tagging for an image of Istanbul

Figure 2.6 shows an annotation which has a target whose type is a *dctypes:Image* and has a URI *http://www.balsoy.com/Turkiye/inpictures/pictures/istanbul01.jpg*.The annotation has a body which is an instance of *oa:SemanticTag* and has *foaf:page* property as *http://dbpedia.org/page/Istanbul*. The annotation's motivation is tagging and it is annotated by a person whose name is "*A Person*" with an open id "*openid1*".

The model is serialized by a software agent named as "*agent1*".

2.2.1.2. Specifiers and Specific Resources. The core of the Web Annotation Data Model is adequate to handle majority of use cases; however considerable number of use cases remain not covered. Resource segment selection is one of these use cases.

The resource that identifies the segment is called the Specific Resource, and the resource which describes how to extract the targeted segment is called Selector. Within the Web Annotation Data Model, only one Selector can be associated with a Specific Resource.

Fragment Selectors : The Web Annotation Data Model defines a fragment-based selector called the FragmentSelector with which the segment of interest can be described through the use of the fragment identifier component of a URI. The description of the segment is included in the Annotation graph via the *rdf:value* property.

Range Selectors : The model defines several extractors which describe how to extract segments that have a start and end in linear data. There are two text selectors and one for bitstreams.

Text Quote Selector is a class which is used to describe a textual segment by quoting the target, plus text before and after it. For example, if the document were "*The Maiden's Tower (Turkish: Kız Kulesi), also known as Leander's Tower*", "*Kız Kulesi*" could be selected by a prefix of "*Turkish: *" and a suffix of "*), also*".

If the content is copyright, then this method can be dangerous as one might select entire document. For such cases, the use of the Text Position Selector is more appropriate. The Selector describes a range of text based on its start and end positions. The properties used to describe the positions are *oa:start* and *oa:end*. Previous example would be described as start would be 29, and end would be 39, then the selection again would be "*Kız Kulesi*".

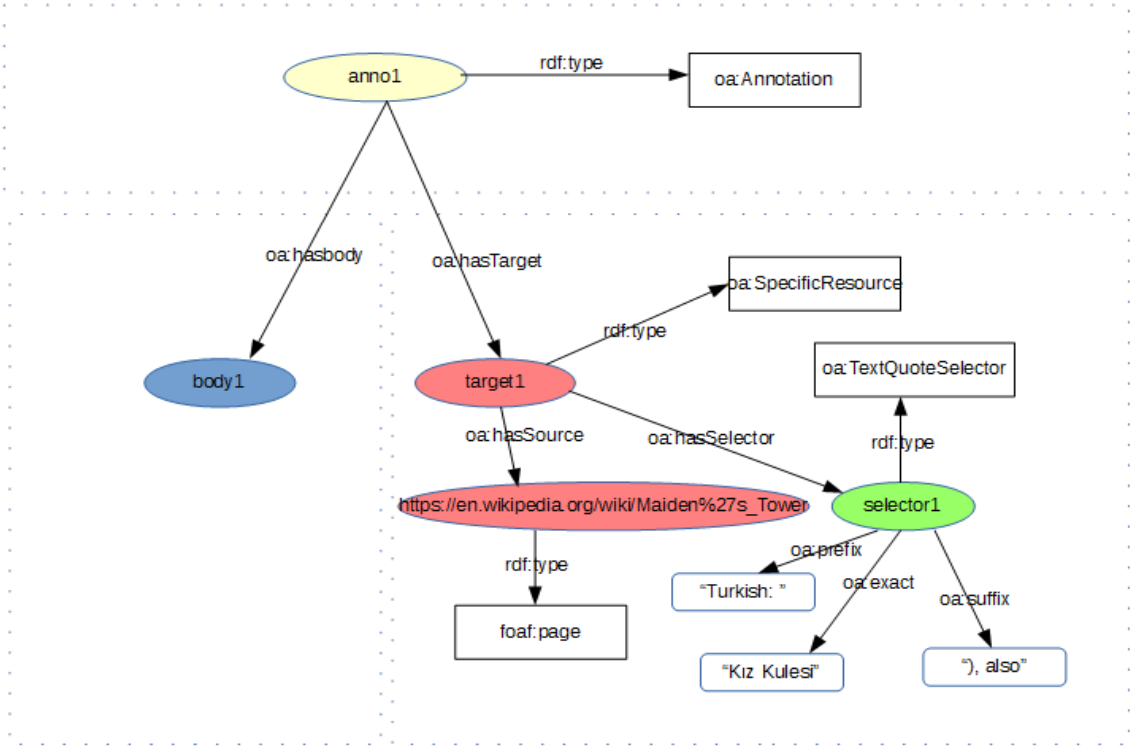

Figure 2.7. Annotation Target with a oa:TextQuoteSelector selecting textual content "Kız Kulesi"

Figure 2.7 shows an example of Text Quote Selector. The Target is a Specific Resource which has a source of a URI "https://en.wikipedia.org/wiki/Maiden'sTower". The target in the figure is not the whole page, but the text "*Kız Kulesi*". In order to select the content, Text Quote Selector is used with a prefix and a suffix.

SVG Selectors : Even though fragment selectors can be used to descrive rectangular areas, it is useful to describe circles, ellipses and polygons. SVGSelector is defined a subclass of **oa:Selector** and can be used to define shapes using the SVG standard.

2.2.1.3. States. Resources on the Web are increasingly dynamic or they can be available in multiple formats. The Web Annotation Data Model also addresses this problem. To resolve this issue, the data model introduces the oa:State class which is used to describe how to retrieve a representation of a resource.

A **Time State** specifier records the time that the Annotation was created. Applications consuming Annotation, can then use this information to figure out an appropriate representation of the resource.

HttpRequestState is a class which can be used to describe how to retrieve an appropriate representation of a resource based on the HTTP Request headers.

In Figure 2.8, time state information is added to the instance of the Target. **state1** is an instance of **oa:TimeState** class and has a property **oa:when** which is used to store the timestamp at which the Source resource should be interpreted.

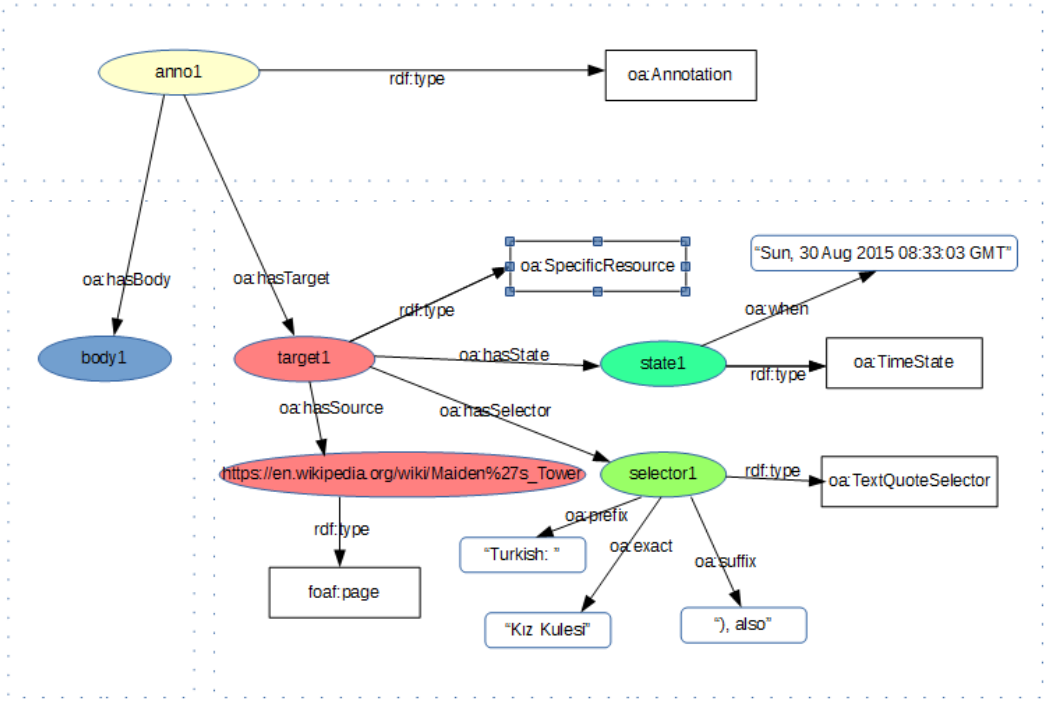

Figure 2.8. Annotation Target defining the state of target resource by using oa:TimeState

2.2.1.4. Highlighting and Styles. In the Web Annotation Data Model, the Style resource is associated with the Annotation itself. The content of the resource provides the rendering hints. **oa:Style** is used to define a resource which describes the style the selection or resource should be rendered. The model doesn't recommend to use the class directly in Annotations, but only its subclasses.

CSS is used in the model via the **oa:CssStyle** class which is a subclass of oa:Style. The class label is attached to a Specific Resource using the **oa:styleClass** property.

Figure 2.9 shows an example of CSS Style. In the annotation, Specific Resource is styled with green style class. The style class is defined as ".green {color : green}".

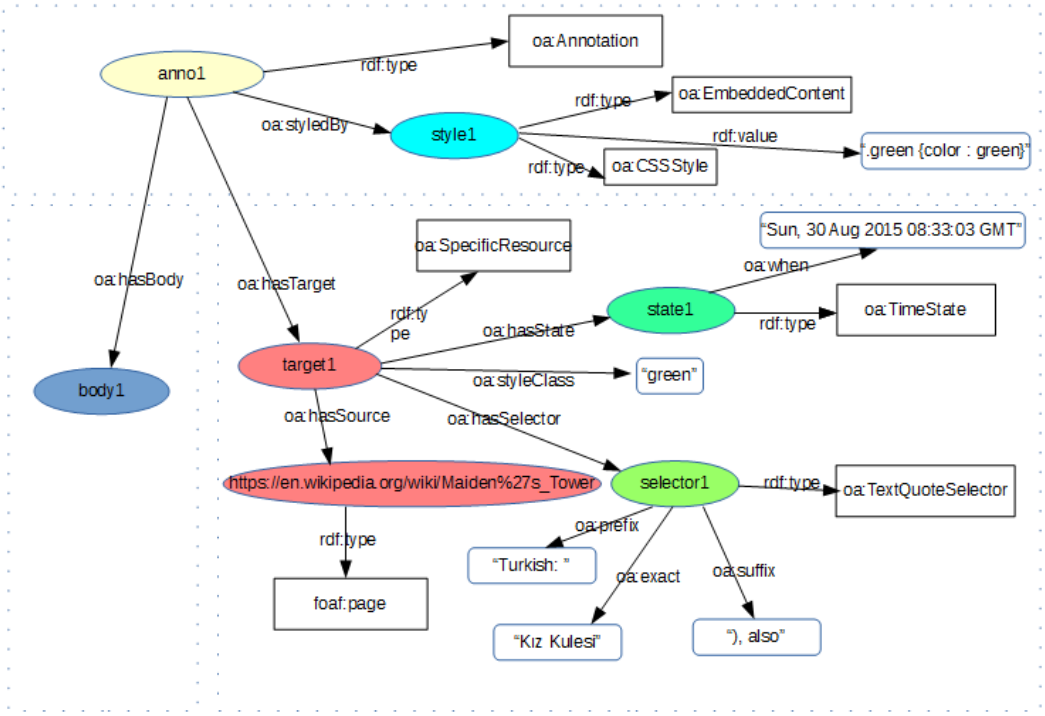

Figure 2.9. Annotation for which content is styled by using oa:EmbeddedContent and oa:CSSStyle

2.2.1.5. Multiplicity Constructs . Even though an Annotation may have multiple Bodies, Targets or both, the semantics are that each Body individually is about each Target, is not always the case. Using the Web Annotation Data Model, an Annotation can be associated with one resource from a group of alternatives. An example to this would be having a comment that is available in Turkish and English.

The Web Annotation Data Model includes three multiplicity constructs : **oa:Choice**, **oa:Composite** and **oa:List**.

oa:Choice : A Choice is a set of resources for which a consuming application

should select only one to process or display.

oa:Composite : A Composite is a set of resources which are all required for an Annotation. For instance, an Annotation which compares the differences between two resources.

oa:List : A List is a set of resources which should be in order in the context of the Annotation.

The following figure shows an example of a Choice construct. The annotation has a Body whose type is **oa:Choice** and it includes to images.

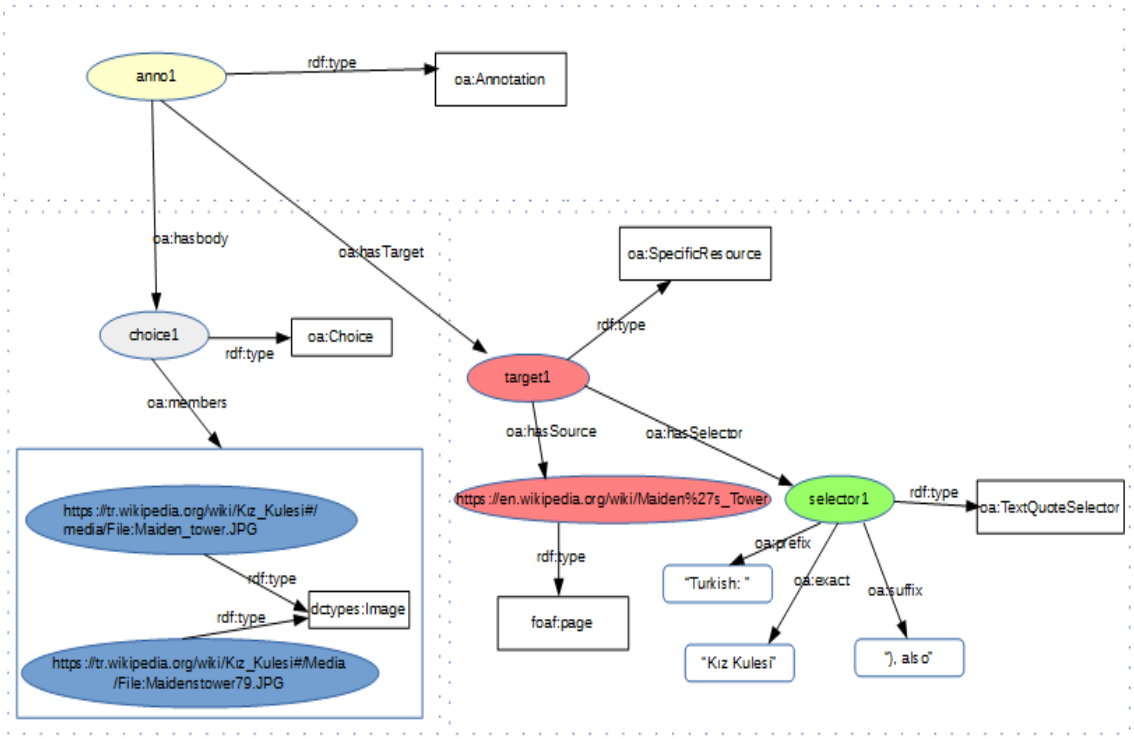

Figure 2.10. Annotation having a body of an instance of oa:Choice, consisting of two images of Kız Kulesi

2.2.2. Other Annotation Related Studies

In this section, we will summarize studies related to annotation topic. We will just give notion of the related studies.

- A Web application called SAPIENT [27] is introduced by Maria Liakata, Claire Q, and Larisa N. Soldatova for sentence based annotation of full papers with semantic information. The application enables annotation of scientific papers sentence by sentence and also linkin related sentences together.
- Another study related to semantic annotation of unstructured and ungrammatical text is carried out by Matthew Michelson and Craig A. Knoblock. The study [28] is about annotation of informal data. The method they used is to leverate collections of known entities and their common attributes, called "reference sets," so that these unstructured data can be annotated despite lack of grammar and structure.
- An open ontology in OWL presented by for scientific documents on the web. The Annotation Ontology supports both human and algorithmic contents [29]. The presented model supports versioning and a set model for specifying groups and containers of annotation.
- SAWSDL [30] defines how to add semantic annotations to various parts of a WSDL document such as input and output message structures, interfaces and operations. In addition to this, it also defines an annotation mechanism for specifying the structural mapping of XML Schema types to and from an ontology.
- Since tweets contain mentions of numerous entities, persons and events, and often additional information, like an opinion that can be viewed as an annotation of that entity [31]. In the study, a natural processing approach is discussed to extract information about entities and their annotations from tweets.
- Semi-automatic semantic moderation of web annotations [32] is a research that focuses on leveraging the context and content features of social web annotations for semi-automatic semantic moderation. In the study, an approach for semi-automatic semantic moderation by analyzing and exploiting both content and context dimensions of annotations.
- A study about annotation of images and videos for multimedia analysis is carried out to propose an approach different than older methods which mainly focus on the content dimension and corresponding annotations, such as a person or vehicle [33]. In this study, a software environment is presented to bridge between two directions by linking low level MPEG-7 visual descriptions to conventional

Semantic Web ontologies and annotations.

- Django [34] is a free and open source web application framework which is written in Python in order to create open sourced online tutorials and curate amazing first experiences with technology. The web framework is a set of components that helps people to develop websites faster and easier. Within this framework, there is translation as well as renarration. Tutorials are translated by sentence by sentence while keeping the links between translated version and the original version.

2.3. Renarration

According to W3C website, the Web is fundamentally designed to work for all people whatever the hardware, software, language, culture, location, etc. However, when websites, web technologies or web tools are badly designed, they can create barriers that exclude people from using the Web [35]. In the renarration model, a web page or even an element of a web page is rewritten, renarrated, to make it accessible for a target audience. [6]

2.3.1. Accessibility

As with the improvement in the Web technologies, the impact of disability is radically changed because of barriers being removed. However, there are accessibility issues on the Web due to social, cultural geographical and other factors.

Figure 2.11 shows an example of accessibility issue. In the figure, there is a web page P about fire safety which is originally written in English. A blind person B, might be able to read this page with help of a screen-reader software. This would be possible only if the author of this page, doesn't violate any guidelines.

Renarration architecture enables someone X, not necessarily the authors, to provide a renarration of this fire safety page which might be more accessible for blinds. X might create *Blind(P)* and put it on the Web. Whenever B visits the page P,

A fire department or fire brigade is a public or private organization that provides fire protection for a certain jurisdiction, which typically is a municipality, county, or fire protection district. A fire department usually contains one or more fire stations within its boundaries, and may be staffed by career firefighters, volunteers, or a combination thereof.

Figure 2.11. A Web Page about fire safety originally written in English [6]

alternative page *Blind(P)*, could be accessed.

2.3.2. Barriers

In this section, we'll talk about kinds of accessibility issues such as language, literacy, etc.

2.3.2.1. Language Barriers. Considering the example, in Figure 2.11, another barrier might lead the page inaccessible. For instance, if the page is visited by a person who cannot read English, it would mean nothing. When a person Y, who know how to read in Turkish, and if a Turkish version of P *Turkish(P)* exists, then Y can read that page instead.

In addition to this, fire engine in the image might not be familiar to Y. So, the renarrated page *Turkish(P)* might also substitute an image of the fire engine found in Turkey instead.

2.3.2.2. Literacy Barriers. According to statistics retrieved from TSI [36], almost 5 percent of the population are still not lettered in Turkey by 2014. It is obvious that most of the content on the Web is text dominated. This kind of statistics doesn't mean that the content will be consumable to them, if it was made available in different

format. For instance, audio, video might be more appropriate for such audience. The renarration approach provides a good way to handle this kind of accessibility issues.

2.3.2.3. Other Accessibility Issues. Even though there are many different ways of implementing web pages, many of them violate guidelines. An example to this would be that tables within the HTML are used mostly for formatting, not just for data in tabular format. Not only violating guidelines might make web pages inaccessible, but also some other accessibility issues might occur. For instance, a page might be inaccessible because of simply poor writing or because the writing is too technical. In those situations, simplifying the content, alternative narration, clearly might be helpful.

2.3.3. Addressing Accessibility on the Web

The section *Designing for Inclusion* of the W3C site states :

"Inclusive design, design for all, digital inclusion, universal usability, and similar efforts address a broad range of issues in making technology available to and usable by all people whatever their abilities, age, economic situation, education, geographic location, language, etc." [6]

The renarration approach is one of the ways of making the web more inclusive. The approach doesn't prescribe any specific implementation or any choices which any implementation should make. It just states the problem of accessibility more broadly and move the discussion towards inclusion.

2.4. Applications

In this section, we provide list of applications which have attempted to increase accessibility of contents on the Web. First, we'll list an annotation and a renarration application. After that, we'll list crowd-sourced applications which are : Django Girls [34], Duolingo [37] and Google Translate [38].

2.4.1. Annotator

Annotator is an open-source JavaScript library which can be used to add annotation functionality to any webpage. Annotations can have comments, tags, links, users and more [39]. Annotator library is designed for easy extensibility.

In order to initialize the library, the following javascript needs to be added.

var ann = Annotator(document.body);

In order to setup default plugins, annotator.min.css and annotator-full.min.js should be included and the following method should be called.

ann.setupPlugins();

Figure 2.12 shows an example of the Share Annotator Plugin [40], which enables sharing annotations in social networks.

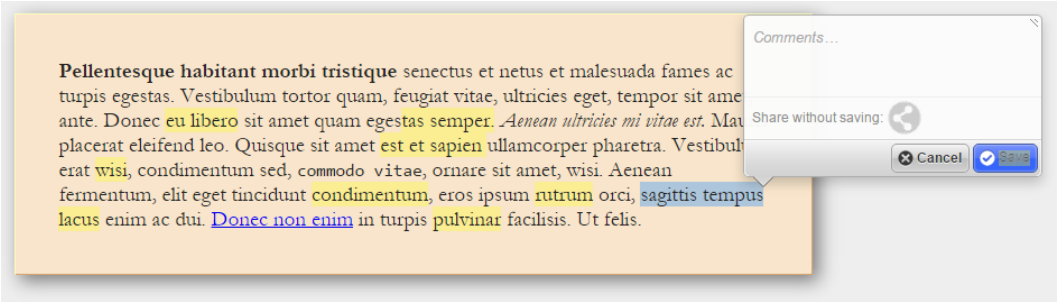

Figure 2.12. User interface for creation of an annotation using Share Annotator Plugin [40]

2.4.1.1. Storage API. The Annotator library defines storage API in terms of a prefix and a number of endpoints. The API follows the principles of REST. Each of the endpoints for the storage API is expected to be existing on the Web. For example, if the prefix were `http://example.storage/api`, then the index endpoint would be `http://example.storage/api/annotations`. There are six core endpoints, **root**, **index**, **create**, **read**, **update**, **delete**.

2.4.1.2. Plugins. The Annotator is a highly modular architecture and plugins provide a great deal of functionality.

The **Auth** plugin complements the Store plugin and provides authentication for requests. The plugin works by requesting a token from the local server and the token is used for all requests to the store.

The **Filter** plugin allows users to filter the displayed annotations. When the plugin is used, a toolbar is added to the top of the window. The toolbar contains available filters which can be applied to the annotations.

The **Markdown** plugin allows users to use Markdown [41] in annotation comments. These comments are then rendered in the viewer. Markdown is a text-to-HTML conversion tool for web writers. It allows writing using an easy-to-read and easy-to-write plain text format, then convert it to valid XHTML.

The **Permissions** plugin handles the user and permissions properties on annotations. The plugin adds Viewer section to a viewed annotation and this displays the name of the user who created it. It also adds two fields to the annotation editor which are showed only when current user has admin permissions on the annotation.

The **Store** plugin sends annotations to the server. The plugin sends the annotations serialised as JSON. The annotator can perform create, update, destroy and search actions.

The **Tags** plugin allows annotators to tag their annotations with keywords. Again when the plugin is used, viewer, which displays any tags, is automatically added. The plugin also adds an input field to the editor which is used to enter space separated list of tags.

2.4.2. Alipi

Alipi [6] is designed with the objective of enabling one set of web users, the renarrators to renarrate any web page or its element, and consumers who consume the web resources renarrated.

2.4.2.1. Approach. From a DOM perspective, renarration is a syntactic restructuring of the DOM structure of the document. [6] Traditional solutions for accessibility is usually done by the author specifying a rewriting rule. For instance, specifying alt tag for an image. However, the approach taken by the Alipi that these rewriting rules will not be a fixed; there might me multiple versions of these rewrites e.g., by a user, the author of the page, or even a service. Alipi accomodates multiple strategies for accessibility : fetching renarrations of a page from a store anywhere on the web, or restructuring a page on the place, or combination of two.

2.4.2.2. Architecture. The architecture of Alipi relies on three main subsystems: (a) a subsystem for renarrators to create renarrations, (b) a subsystem for indexing web pages or its elements to renarrations, and (c) another subsystem for renarrating web pages.

Figure 2.13 shows creation and indexing of renarrations of a page P consisting of elements (E and E'). Renarrations E1 and E2 are being created for the element E. The renarrations exist on the Web each with its own url U1 and U2, etc. The Alipi architecture requires all renarrations accessible on the Web. That also ensures a decentralised renarration model. Each renarration contain an attribute called **foruri** which is used to refer to the original object.

Crawler and indexer fetches the original page and existing renarrations created for the page P's elements U/E and generates an index.

When a user requests the page at url U, semantic attributes for the user's profile

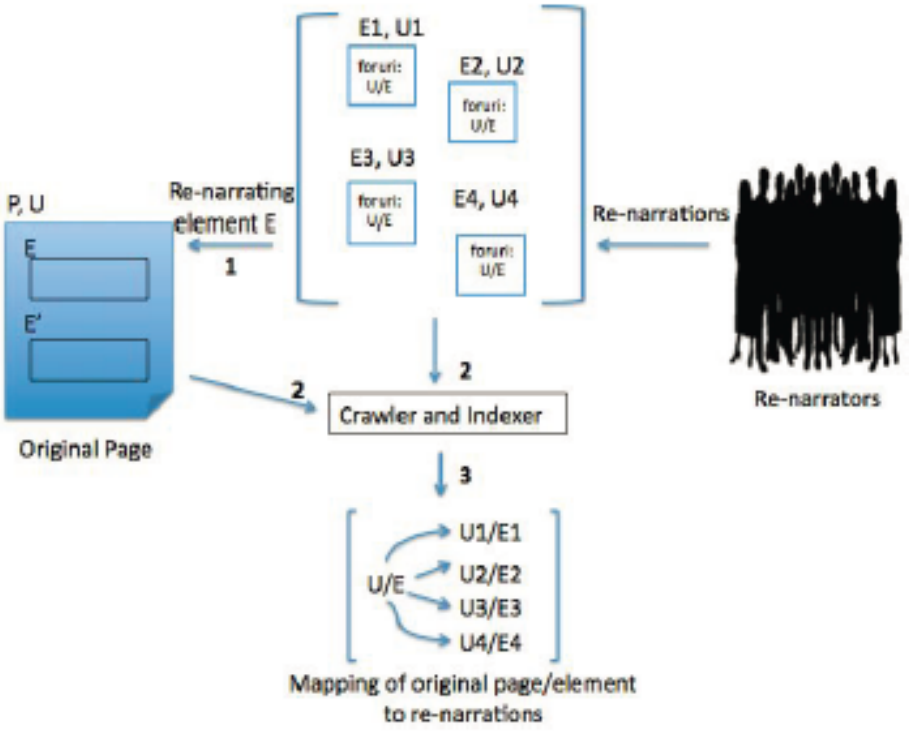

Figure 2.13. The creation and indexing of renarrations of a page P with url U using Alipi prototype [6]

are being used and to match appropriate renarrations of the requested page. The appropriate renarration then rendered in the user's browser as renarration P' of P at the same url.

2.4.2.3. Alipi Prototype. Renarration is implemented as a service in the Alipi prototype. Using the service, a user can choose a web page for renarration and specify target groups and publish the renarration.

Any number of renarrations may be created for any given web page. The renarrator can define translations, simplifications or provide alternative media to the target audience. In addition, the renarrator can provide the language, geographical region, and tags.

Figure 2.14 shows a page on fire safety in English. In the figure, a popup is shown to the renarrator with the image of fire truck can be replaced.

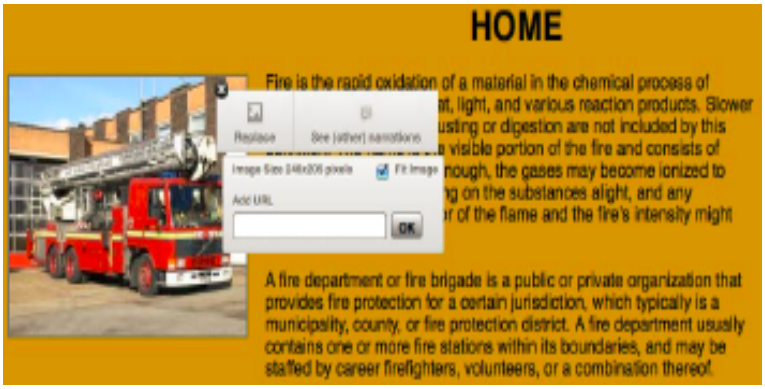

Figure 2.14. The page on fire safety being edited for renarration. Note the popup on the image inviting its replacement [6]

Figure 2.15 shows renarrated version of the fire safety page in Figure 2.14. The second paragraph has been replaced by text in Hindu and the picture has been replaced with an image of Indian fire truck.

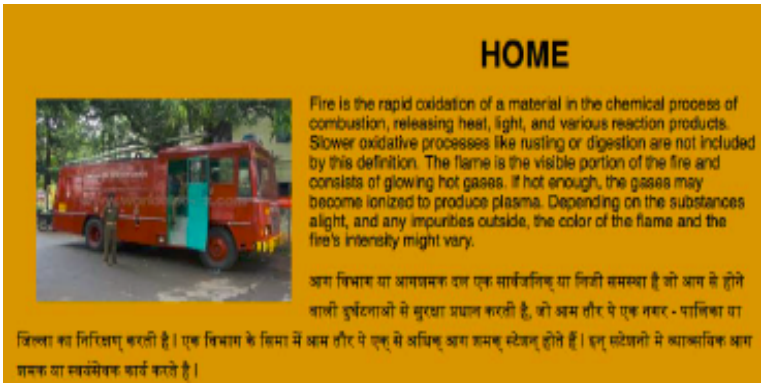

Figure 2.15. Renarrated version of the fire safety page in Figure 2.14. [6]

2.4.3. Other Crowd-sourced Applications

There are several systems that have attempted to increase the accessibility of content on the Web. In this subsection, we'll talk about three of them : Django Girls [34], Duolingo [37] and Google Translate [38].

Django Girls is a non-profit organization that empowers and helps women to organize free one-day programming workshops by providing tools, resources and support. It is a crowdin translation application for collaborative translations. The mission

of Django Girls is to bring more women into the world of technology and increase the diversity. The barrier that they are dealing with is language. Other barriers are trying to be resolved with mentors. Figure 2.16 shows the translation of a tutorial about how to create a Django views. In the figure, it can be seen that each sentence is indexed and relation between original content and its translation is kept.

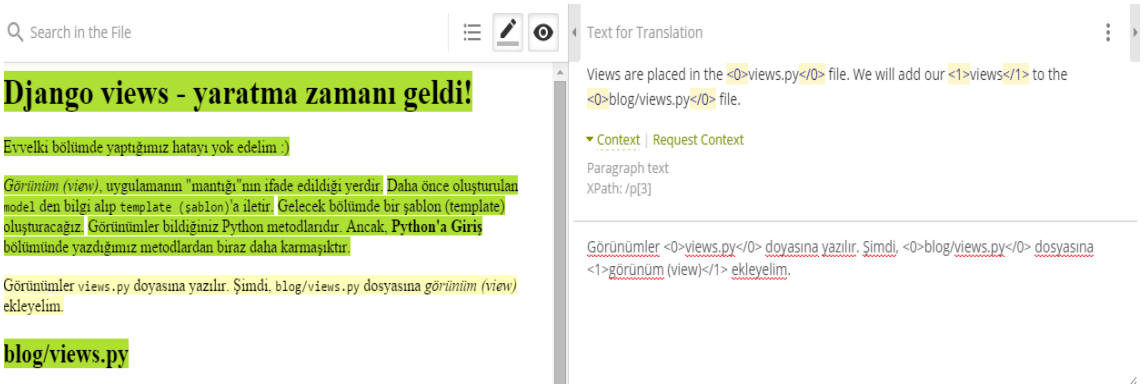

Figure 2.16. Translation of a tutorial about how to create a Django views.

Duolingo is a free language-learning platform that includes a language-learning website and app along with a crowdsourced text translation platform and a language proficiency assessment center. The website offers over 50 different language courses across 23 languages. It employs a crowd sourced business model, where members of the public are invited to translate content and vote on translations. The content comes from organizations that pay Duolingo to translate it. Figure 2.17 shows a discussion of a particular question. User wants to discuss the answer with others. People who are native or know language well often answer.

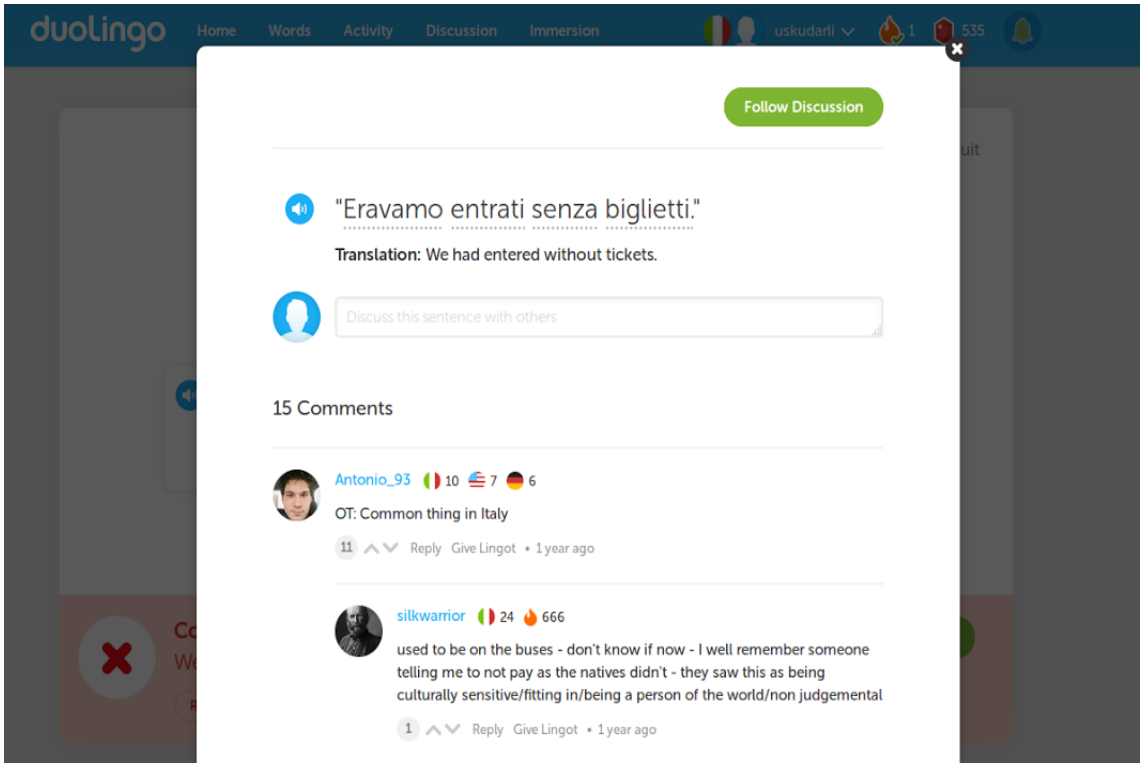

Figure 2.17. Duolingo screen showing a discussion of a particular question where user wants to discuss the answer with others.

Google Translate is a free multilingual statistical machine translation service provided by Google to translate text, speech, images, or real-time video from one language into another. It does not apply grammatical rules, since its algorithms are based on statistical analysis rather than traditional rule-based analysis. It also has gamification and crowd wisdom. Figure 2.18 shows an example translation for a course equivalence for an exchange student. The aim of the student is to figure out whether the content of courses matches or not.

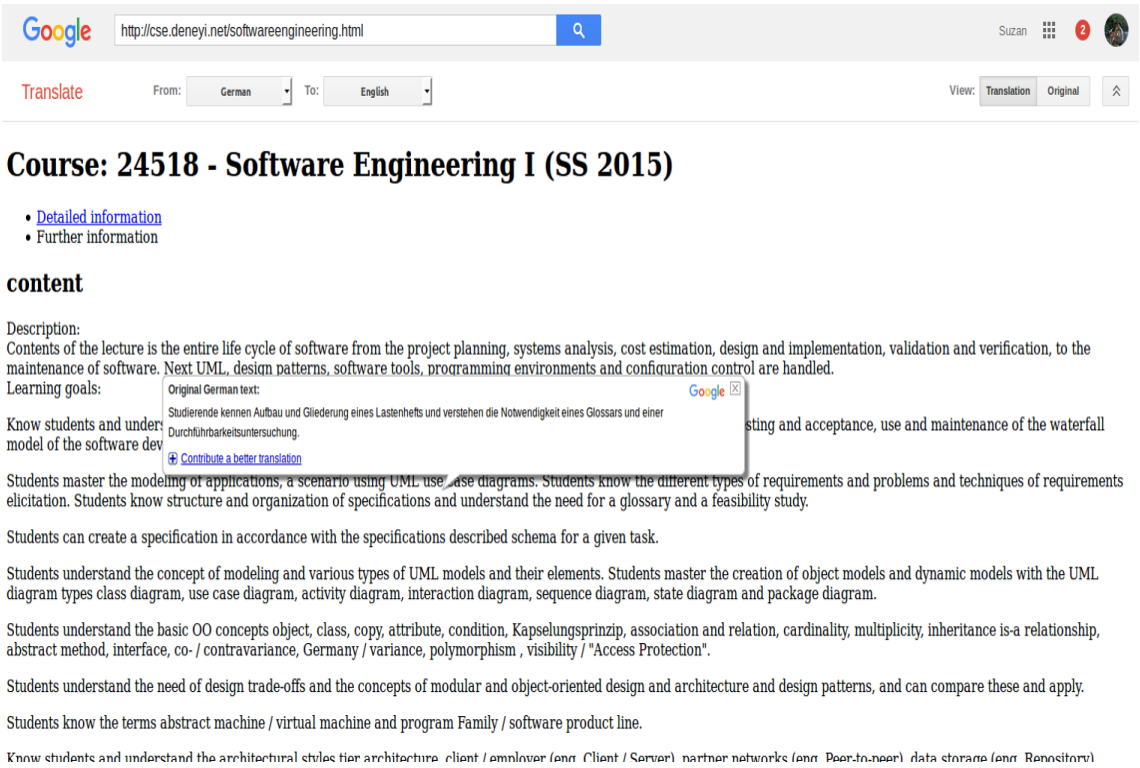

Figure 2.18. An example Google translation for a course equivalence for an exchange student.

3. MODEL

This chapter proposes a model for creating renarrations of resources. The model has been realized through the design of five major sub-systems: *Renarration Transformation* for describing different type of actions on HTML elements, *Selectors* for describing parts of resources, *Actions* to describe the activity done on the element in source documents, *Motivations* for describing the reason of renarration transformations and/or whole renarrations, and lastly *Lists* for defining ordered constructs for selecting parts of source documents in an order or creating ordered narrations of source documents.

3.1. Renarration Data Model

Renarration Data Model specifies a framework for creating alternative narrations of source documents by transformation of their elements such as translation of a paragraph or replacing an element with an image.

Figure 3.1 shows high level pseudo code of renarration process on a source document.

A **Renarration** is considered to be an alternative narration described by a given user at a given time consisting of a set of transformations. Figure 3.2 shows renarration of a source document. The source document is renarrated by a set of transformations on the elements of the document, as a result target document is composed. The whole process is called a *Renarration*.

The Model defines a namespace for its classes, and properties. It also uses other namespaces. Table 3.1 shows namespaces used in the data model.

The specification for the data model is divided into distinct modules. Figure 3.1 shows the relationship between **rn**:*Renarration* and **rn**:*Renarrator* classes. The

```
let SD be a web resource to be renarrated
let TD be a target document (renarrated document)
list ← list of elements inSD
for each element E in list do
   if E is renarrated then
      action ← insert, replace, remove
      E′ ← apply action on E
      create semantic relation between E and E'
      TD ← E'
   end if
end for
return TD
```

Figure 3.1. Pseudo code for creation of renarration of a resource

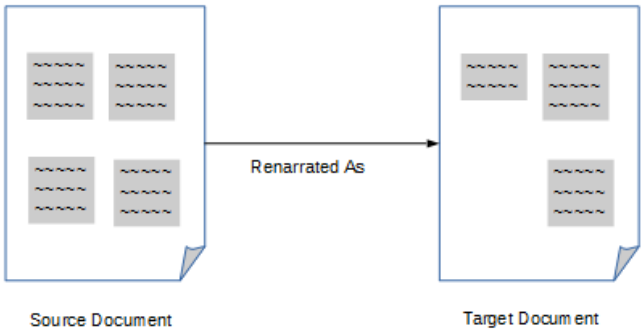

Figure 3.2. Renarration Of A Document

Table 3.1. Namespaces used in the Renarration Data Model

Prefix	Namespace	Description
oa	`http://www.w3.org/ns/oa#`	Web Annotation Data Model [1]
foaf	`http://xmlns.com/foaf/0.1/`	Friend-of-a-Friend Vocabulary [23]
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	RDF [25]
rdfs	`http://www.w3.org/2000/01/rdf-schema#`	RDF Schema [15]
skos	`http://www.w3.org/2004/02/skos/core#`	Simple Knowledge Organization System [26]
owl	`http://www.w3.org/2002/07/owl#`	OWL Web Ontology Language [16]
xsd	`http://www.w3.org/2001/XMLSchema#`	XML Schema [42]
cnt	`http://www.w3.org/2011/content#`	Content in RDF [43]

data model proposes that each renarration must be created by a renarrator and each renarrator must be a person not a software agent. The equivalent class of **rn**:*Renarrator* is defined as **foaf**:*person*. Table 3.2 shows explanation of terms used in Figure 3.1.

Table 3.2. Definitions of classes and properties related directly to renarration

Item	Type	Description
rn:*Renarration*	Class	The class for Renarrations. The **rn**:*Renarration* class must be associated with a renarrator and a time stamp.
rn:*Renarrator*	Class	The class for Renarrators. Each renarrator must be equivalent with **foaf**:*Person*.
rn:*Document*	Class	Any type of resource on the Web. If a more specific type is known, than it should be associated with source or renarrated documents.
rn:*renarratedBy*	Object Property	The relationship between a Renarration and a Renarrator. There must be at least one **rn**:*renarratedBy* relationship associated with a Renarration. The object of the relationship must be an instance of **rn**:*Renarrator* class, which is the person who is responsible for the creation of the Renarration.
rn:*onSourceDocument*	Object Property	The relationship between a Renarration and a source document. There must be at least one **rn**:*onSourceDocument* relationship associated with a Renarration. The object of the relationship must be a document for which the renarration is being done.
rn:*toTargetDocument*	Object Property	The relationship between a Renarration and a target document (renarrated document). There must be at least one **rn**:*toTargetDocument* relationship associated with a Renarration. The object of the relationship must be an instance of **rn**:*Document* class.
rn:*renarratedAt*	Data Property	The time at which the Renarration is created. There must be exactly one **rn**:*renarratedAt* property associated with a Renarration.

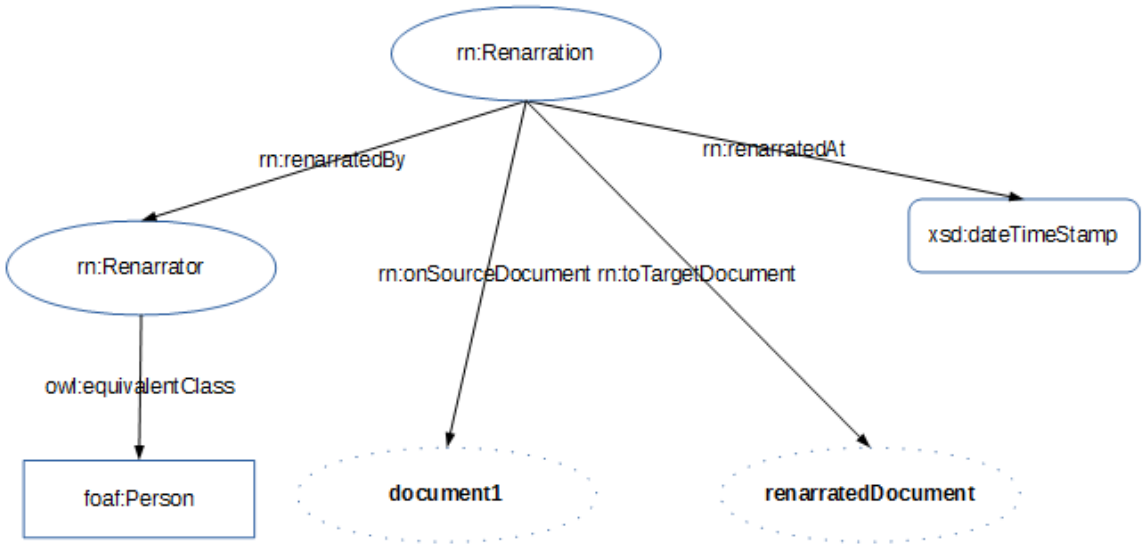

Figure 3.3. The Renarration Model consisting of a Renarrator, a source document, and a renarrated document, and the time of renarration.

Figure 3.4 shows serialization of a basic renarration in JSON-LD format. In the serialization, it is assumed that there is a source document 'document1' (Line 6) and target document (Line 15).

3.1.1. Renarration Transformation

Renarration of a source document is done by transformation of elements in the document. **Renarration Transformation** can be defined as an atomic modification to part of a source document to create an alternative version of it.

In Figure 3.5, source document SD includes 4 elements: E1, E2, E3, and E4. The source document is renarrated and the only transformation done on the element E1. As it can be seen in the figure, the text in E1 is updated. This atomic operation itself is called a Renarration Transformation, and whole process is called a Renarration.

$$Renarration\ Transformation(E1) = E1'$$
$$Renarration(SD) = E1'\ U\ E2'\ U\ \ldots\ EN'$$

```
{
 "@id" : "renarration1",
 "@type": "rn:Renarration",
 "rn:renarratedAt": "2015-09-14T00:00:00Z",
 "rn:onSourceDocument": {
  "@id": "document1",
  "@type": "rn:Document"
  },
 "rn:renarratedBy": {
  "@id": "http://ex.org/person1",
  "@type": "foaf:Person",
  "foaf:name": "Person One"
 },
 "rn:toTargetDocument": {
  "@id": "renarratedDocument",
  "@type": "rn:Document"
  }
}
```

Figure 3.4. Serialization of a basic Renarration renarrated by a person in JSON-LD

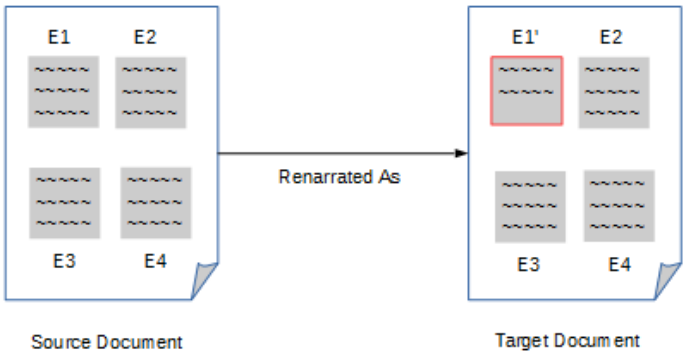

Figure 3.5. Renarration Transformation on an element of a source document

In the Renarration Data Model, each renarration transformation must be done on the same source document for a renarration. Each renarration transformation should have **rn**:*onSourceDocument*, and **rn**:*hasNarration* relationships. **rn**:*onSourceDocument* is the relationship between renarration transformation and selected part of source document. **rn**:*hasNarration* is used to define relationship between the renarration transformation and narration created by renarrator. **rn**:*toTargetDocument* defines part of the renarrated document which is related to selected part in the source document. If the source and target selections are the same, it's not mandatory to define target selection with **rn**:*toTargetDocument*. Table 3.3 shows explanation of terms used in Figure 3.6.

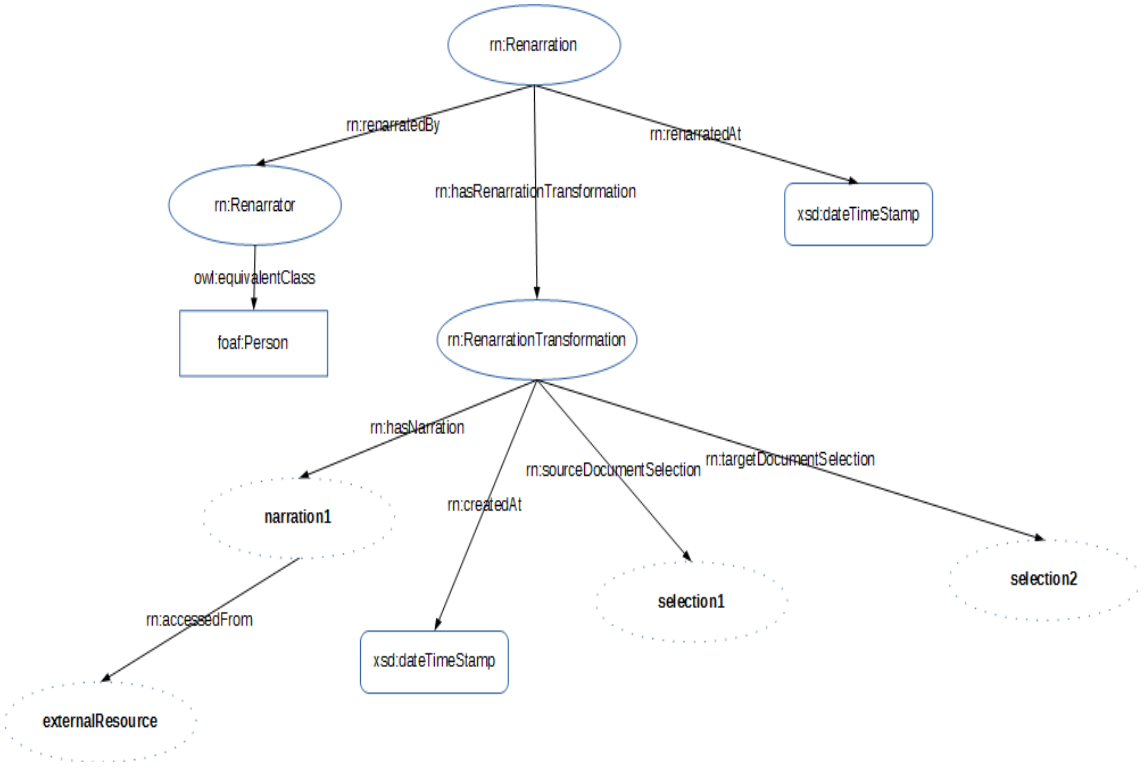

Figure 3.6. Diagram for relations between a renarration and renarration transformation with a narration, source document and renarrated document

Table 3.3. Description of items for a Renarration Transformation on source documents

Item	Type	Description
rn:*RenarrationTransformation*	Class	The class for Renarration Transformations defining specific actions on parts of source documents.
rn:*hasRenarrationTransformation*	Object Property	The relationship between a Renarration and Renarration Transformation. The object of the relationship can also be an instance of structure(**rdf**:*List*).
rn:*hasNarration*	Object Property	The relationship between a Renarration Transformation and a narration defining content being created for source content.
rn:*sourceDocumentSelection*	Object Property	The relationship between a Renarration Transformation and part of a source document. The object of the relationship must be an instance of a selector class.
rn:*targetDocumentSelection*	Object Property	The relationship between a Renarration Transformation and part of the renarrated document. The object of the relationship must be an instance of a selector class.
rn:*createdAt*	Data Property	The time at which the Renarration Transformation is created. There must be exactly one **rn**:*createdAt* property associated with a Renarration Transformation.
rn:*accessedFrom*	Object Property	The relation between a narration and external resource from which the content may be referenced.

Normally, it would be very rare that there will be only one transformation for a renarration. The data model recommends using **rdf**:*List* [15] where there are more than one transformation on the elements of a source document. The list-like structures are covered in following sections.

Figure 3.7 shows serialization of a basic renarration transform in JSON-LD format. Note that in the example, the narration type is defined as **cnt**:*ContentAsText*. The **cnt**:*ContentAsText* that RDF [43] is for any type of textual content.

```
{
 "@id": "renarration1",
 "@type": "rn:Renarration",
 "rn:renarratedAt": "2015-09-14T00:00:00Z",
 "rn:renarratedBy": {
  "@id": "http://ex.org/person1",
  "@type": "foaf:Person",
  "foaf:name": "Person One"
 },
 "rn:onSourceDocument": {
  "@id": "document1",
  "@type": "rn:Document"
  },
 "rn:hasRenarrationTransformation":{
  "@id": "renarrationTransformation1",
  "@type": "rn:RenarrationTransformation",
  "rn:hasNarration": {
   "@id": "narration1",
   "@type": "cnt:ContentAsText",
   "cnt:chars": "content of narration"
  },
  "rn:createdAt": "2015-09-14T22:20:00Z"
 },
 "rn:toTargetDocument": {
  "@id": "renarratedDocument",
  "@type": "rn:Document"
  }
}
```

Figure 3.7. Serialization of a basic renarration transform in JSON-LD

3.1.2. Actions on Documents

In many cases, a renarration transformation on a source document or on part of the source document has an activity on the element itself. In other words, a renarrator

might want to remove, replace or insert some content in the source document when creating a narration.

Action can be defined as type of atomic activity done on part(elements) of source document, where the result of the activity is the narration created by the renarrator. Each instance of **rn**:*RenarrationTransformation* must have at least one **rn**:*actionOnDocument* relationship to an instance of **rn**:*Action* class which is a sub-Class of **skos**:*Concept*.

Figure 3.8 shows an example of two actions on source document. Element E1 in the source document is replaced with new content and this is shown as E1' in target document. Another action on the source document is removal of E3; the target document doesn't have that element.

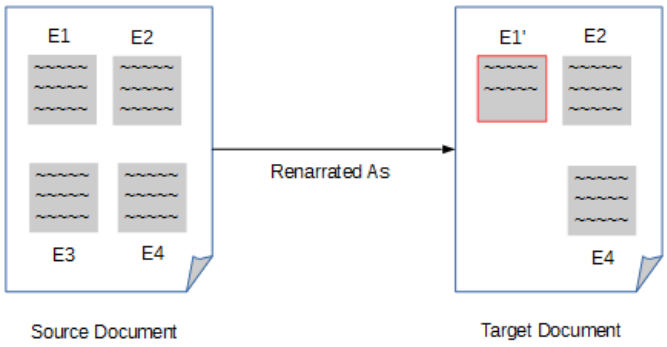

Figure 3.8. Diagram for renarration having replacement and removal actions

In Figure 3.9 the relationship between **rn**:*RenarrationTransformation* and **rn**:*Action* classes is shown. The data model uses SKOS Concepts and SKOS Concept Schemes for defining actions. **rn**:*Action* is defined as a subClass of **skos**:*Concept* and each instance of **rn**:*Action* is defined under **rn**:*actionScheme* which is an instance of **skos**:*ConceptScheme*. Table 3.4 shows explanation of terms used in Figure 3.9.

Table 3.4. Class and Property Definitions of Actions

Type	Name	Description
Class	**rn**:*Action*	subClass of **skos**:*Concept*. The **rn**:*Action* class must be associated with a Renarration Transformation.
Object Property	qnamernactionOnDocument	The relationship between **rn**:*RenarrationTransformation* and **rn**:*Action* for defining type of action on targeted source element.
Named Individual	**rn**:*actionScheme*	Instance of **skos**:*ConceptScheme*. The concept scheme for defining a set of instances of **rn**:*Action*.
	rn:*replace*	Instance of **rn**:*Action*. The action that represents replacing content in a source document with new content which will reside in a target document.
	rn:*remove*	Instance of **rn**:*Action*. The action that represents replacing content in a source document with nothing in a target document.
	rn:*insert*	Instance of **rn**:*Action*. The action that represents insertion of content to a specified location.

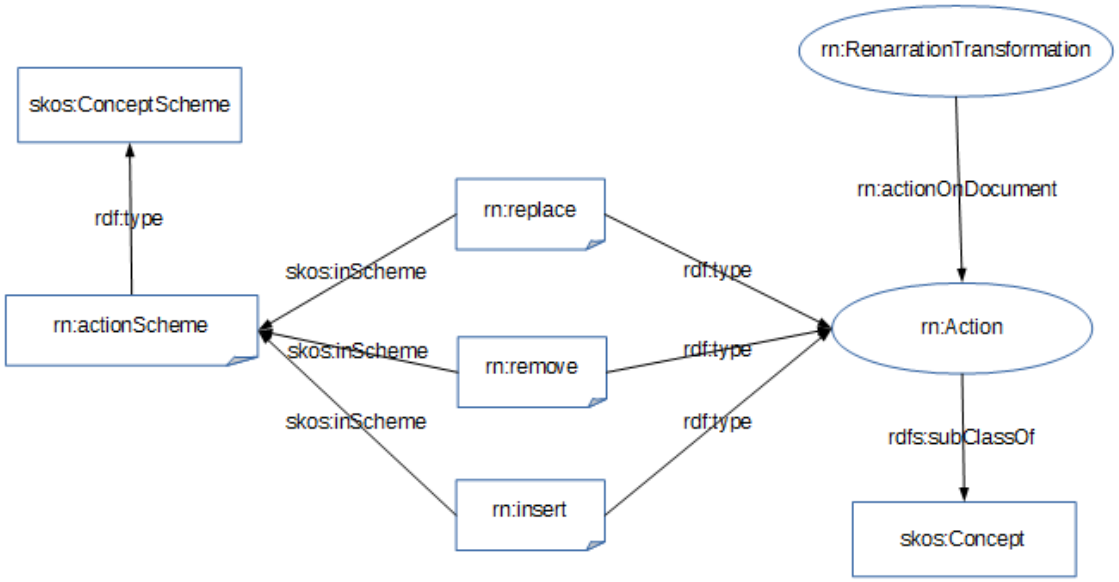

Figure 3.9. Diagram for list of actions can be defined for renarration transform

In Figure 3.10 which is a continuation of Figure 3.7, content in source document is replaced with new content which is defined by using **cnt**:*ContentAsText*.

3.1.3. Selectors

Even though a renarration can have more than one renarration transformation, all transformations on a document are thought to be only one renarration. However, renarration transformations usually refer to part of a source document instead all of it.

A Document can be described any web resource which can be identified by a URI. In order to determine parts of source documents, the Renarration Data Model uses selectors. A Document Selector is a superclass for all selectors to describe how to determine parts of source documents. There must be at least one selector class associated with a class **rn**:*Document*.

In Figure 3.11, an example of selection is shown. Source document has an element

```
{
 "@id": "renarration1",
 "@type": "rn:Renarration",
 "rn:renarratedAt": "2015-09-14T00:00:00Z",
 "rn:onSourceDocument": {
  "@id": "document1",
  "@type": "rn:Document"
  },
 "rn:renarratedBy": {
  "@id": "http://ex.org/person1",
  "@type": "foaf:Person",
  "foaf:name": "Person One"
 },
 "rn:hasRenarrationTransformation":{
  "@id": "renarrationTransformation1",
  "@type": "rn:RenarrationTransformation",
  "rn:hasNarration": {
   "@id": "narration1",
   "@type": "cnt:ContentAsText",
   "cnt:chars": "content of narration"
  },
  "rn:createdAt": "2015-09-14T22:20:00Z",
  "rn:actionOnDocument" : "rn:replace"
 },
 "rn:toTargetDocument": {
  "@id": "renarratedDocument",
  "@type": "rn:Document"
  }
}
```

Figure 3.10. Serialization in JSON-LD format for a basic action for which content is replaced with a textual content

E1 which includes elements E11 and E12. In the example, renarrator renarrates E11 and creates E11' in the target document. In order to select an element which is inside another element, a selector can be used.

TextSelector is a superclass for **PrefixSuffixSelector** and **rn**:*ByteSelector*. This selector is used to select text based resources or parts of resources which are in text format. The main reason to have two different selectors is that **rn**:*ByteSelector* is aimed to select large chunks of texts. For applications implementing the Renarration Data Model, it would be a lot easier to use offsets when larger text is targeted. **PrefixSuffixSelector** has three data properties such as prefix, suffix and selected text. It is

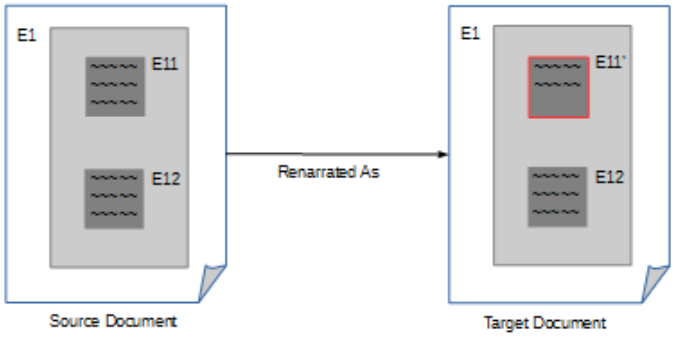

Figure 3.11. Diagram for renarration of a child element

not necessary to define prefix or suffix for a selected text which would mean all instances of selected text within the context. For instance, assuming a renarrator renarrating a sentence S in a paragraph P which is in a web page WP. PrefixSuffixSelector(S in P, prefix="", suffix="") would mean all S in P.

Even though most of the content on the Web is in text format, there are many other data formats. XML is just one of these formats, and HTML documents can also be treated like XML documents. XPath is used to navigate through elements and attributes of XML documents and it is a major element in W3C's XSLT standard. It can be described as a syntax for defining parts of an XML document. XPath models an XML document as a tree of nodes and it defines a way to compute a string value for each type of node. **XPathSelector** is used to select elements of an XML document. The reason that the model includes such a selector is that it is easy to target elements of HTML documents. In addition to this, consuming applications can use JavaScript to find targeted elements easily. **XPathSelector** uses `http://tools.ietf.org/rfc/rfc3023` specification to define syntax of the selection.

In order to select part of media on the Web, the model proposes **MediaSelector**. This selector uses `http://www.w3.org/TR/media-frags/` specification which describes the Media Fragments basic specification. The specification specifies the syntax for constructing media fragment URIs. It also explains how to handle them over the HTTP protocol. This selector can be used to select fragments of images, videos, and/or audio files.

While renarrating web resources, selection of part of documents between specified positions can also be defined by using **BetweenSelector**. This selector uses an object property **between** for which the object should be an instance of **rn**:*List* class. For selections like 'insertion of content before selection', we define an instance of **BetweenSelector** class for which the object of **between** object property is a list that has first element an instance **rn**:*Empty* class and next element being an instance of any selector.

Table 3.5. Class Definitions of Selectors

Name	Description
rdf:*DocumentSelector*	The superclass for all selectors.
rn:*TextSelector*	subClass of **rn**:*DocumentSelector* and superclass of **rn**:*PrefixSuffixSelector* and **rn**:*ByteSelector*. The class describes a range of text either by the use of start and end positions or by using prefix and suffixes.
rn:*XPathSelector*	subClass of **rn**:*DocumentSelector*. The class describes a targeted segment by using a fragment identifier using xpath.
rn:*MediaSelector*	subClass of **rn**:*DocumentSelector*. The class describes a targeted segment of any media on the Web by use of a fragment identifier.
rn:*PrefixSuffixSelector*	subClass of **rn**:*TextSelector*. The class describes a range of text by the use of prefix and suffix.
rn:*ByteSelector*	subClass of **rn**:*TextSelector*. The class describes a range of text by the use of start and end bytes.
rn:*BetweenSelector*	subClass of **rn**:*DocumentSelector*. The class describes a selection between two selectors.
rn:*EmptySelector*	subClass of **rn**:*DocumentSelector*. The class describes no selection. Special class designed to be used with **rn**:*BetweenSelector*.

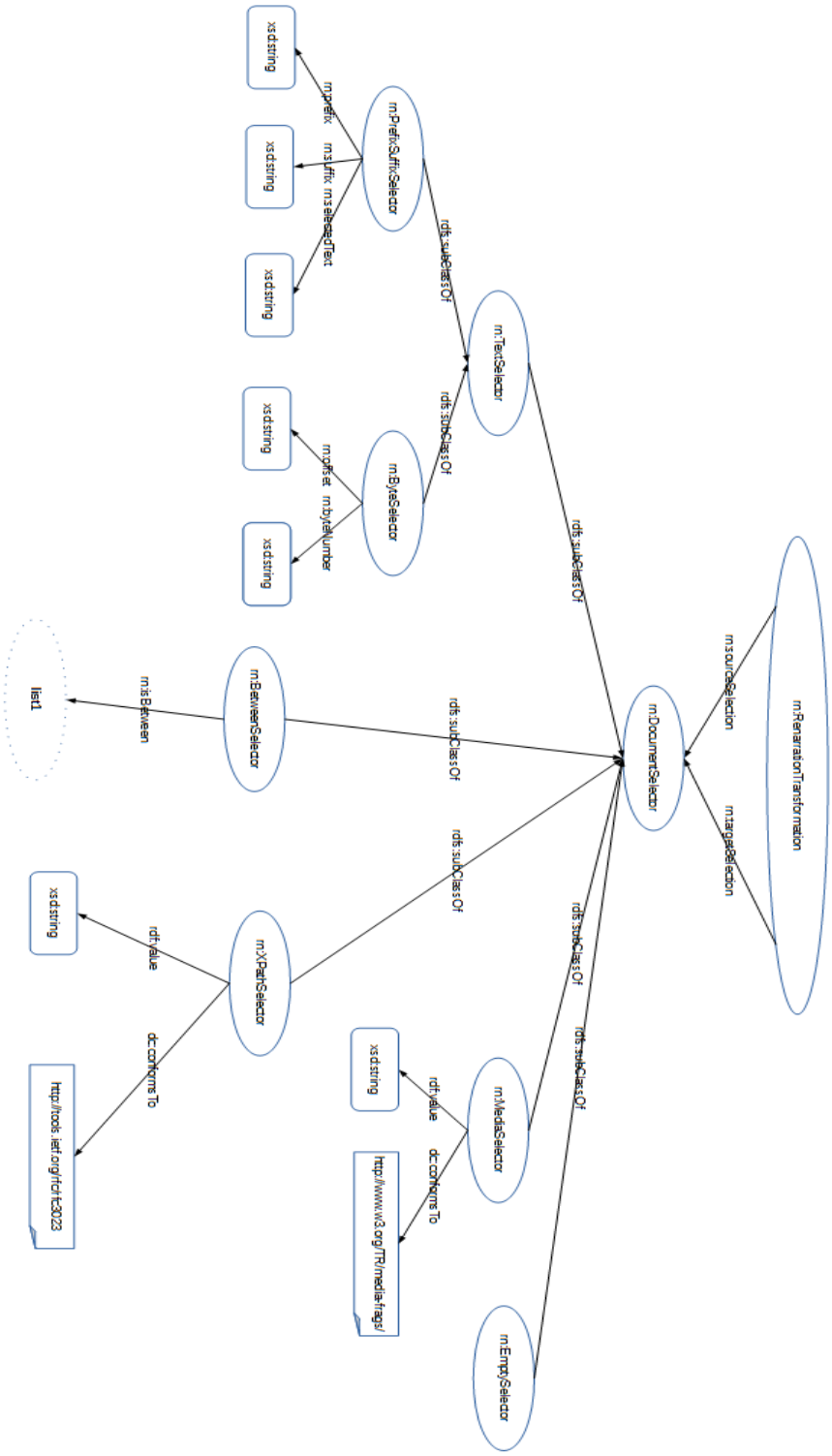

Figure 3.12. Document Selectors in the Renarration Data Model

Table 3.6. Description of the properties related to selectors

Type	Name	Description
Object	**rn**:*sourceSelection*	associates the source document of a rennaration with a selector
	rn:*targetSelection*	associates the target document of a rennaration with a selector
	rn:*isBetween*	specifies a position with a **rn**:*BetweenSelector* in a **rn**:*List*.
Data	**rn**:*prefix*	subsstring prior the target text
	rn:*suffix*	The snippet of text occurs after the targeted text in a resource.
	rn:*selectedText*	The snippet of text which is targeted.
	rn:*offset*	The starting position of the content to be selected.
	rn:*byteNumber*	The number of bytes should be selected after the value of **rn**:*offset*
	rdf:*value*	The value of fragment identifier.
	dc:*conformsTo*	Each **rn**:*XPathSelector* and **rn**:*MediaSelector* classes must have **dc**:*conformsTo* relationship. Defines an established standard to which the described resource conforms.

Figure 3.13 is a continuation of Figure 3.10, where **rn**:*ByteSelector* is used to select 20 bytes starting from 10 bytes within the source document.

```
{
 "@id": "renarration1",
 "@type": "rn:Renarration",
 "rn:renarratedAt": "2015-09-14T00:00:00Z",
 "rn:renarratedBy": {
  "@id": "http://ex.org/person1",
  "@type": "foaf:Person",
  "foaf:name": "Person One"
 },
 "rn:onSourceDocument": {
  "@id": "document1",
  "@type": "rn:Document"
  },
 "rn:hasRenarrationTransformation":{
  "@id": "renarrationTransformation1",
  "@type": "rn:RenarrationTransformation",
  "rn:sourceDocumentSelection": {
    "@id": "selector1",
    "@type": "rn:ByteSelector",
    "rn:offset": "10",
    "rn:byteNumber": "20"
  },
  "rn:hasNarration": {
   "@id": "narration1",
   "@type": "cnt:ContentAsText",
   "cnt:chars": "content of narration"
  },
  "rn:createdAt": "2015-09-14T22:20:00Z",
  "rn:actionOnDocument": "rn:replace"
 },
 "rn:toTargetDocument": {
  "@id": "renarratedDocument",
  "@type": "rn:Document"
  }
}
```

.

Figure 3.13. Serialization in JSON-LD format for selection of 10 bytes

3.1.4. Motivations

It is very important to know the motivation behind a renarration in order to make content on the Web more accessible. The Motivation can be defined as the reason why renarration or renarration transformation within a renarration is created.

The Renarration Data Model proposes 4 types of motivations for creating renarrations. These named individuals are created as instances of **rn**:*Motivation* class. **rn**:*Motivation* class is defined as subClass of **skos**:*Concept*. Each renarration must have at least one **rn**:*hasMotivation* relationship to an instance of **rn**:*Motivation* class which is a subClass of **skos**:*Concept*.

Figure 3.14 shows the relationship between **rn**:*RenarrationTransformation* and **rn**:*Motivation* classes.

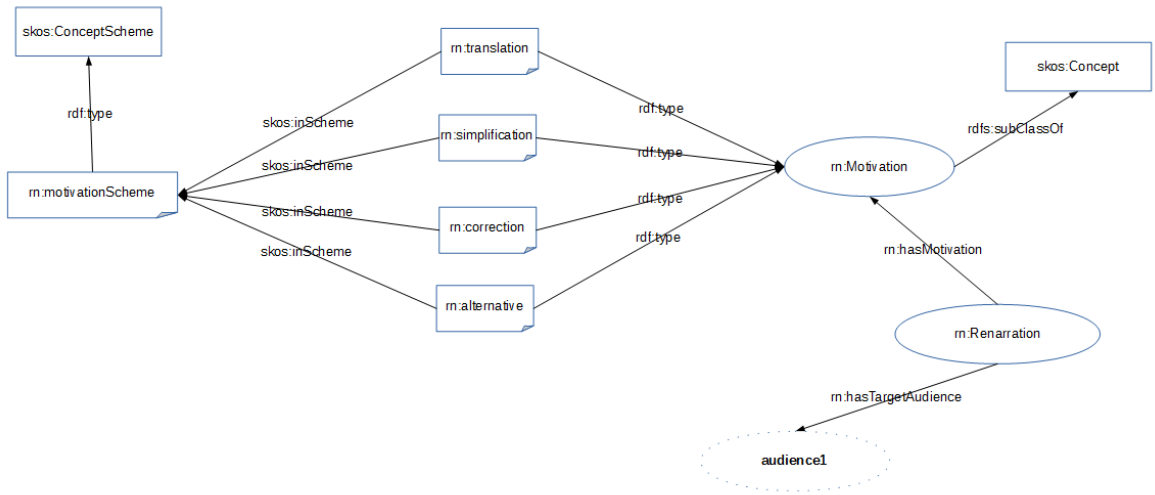

Figure 3.14. The relationship between **rn**:*Renarration* and **rn**:*Motivation* classes.

The Renarration Data Model allows usage of motivations for both renarration transformation and the renarration itself. For instance, the subject of the relationship can be an instance of **rn**:*Renarration* or an instance of **rn**:*RenarrationTransformation*. Table 3.7 shows explanation of terms used in Figure 3.14.

Table 3.7. Class and Property Definitions for Motivations

Type	Name	Description
Class	**rn**:*Motivation*	subClass of **skos**:*Concept*. The reason behind the creation of a renarration or a renarration transformation. Motivations can be things like translation, simplification, correction, or alternative.
	skos:*conceptScheme*	A SKOS concept scheme can be defined as an aggregation of one or more SKOS concepts.
Object Property	**rn**:*hasTargetAudience*	The relationship between a renarration and a group of people renarration is aimed for.
	rn:*hasMotivation*	The relationship between a renarration and motivation.
	skos:*inScheme*	The relationship between a Thing and an instance of **skos**:*ConceptScheme*.
Named Individual	**rn**:*motivationScheme*	The concept scheme for motivations.
	rn:*translation*	The motivation that represents interpretation of source content in another language.
	rn:*simplification*	The motivation that represents simplifying content in a source document so that it will be easier to consume.
	rn:*correction*	The motivation that represents updating content either for verification or adjustment.
	rn:*alternative*	The motivation that represents providing a new content which alternates content in a source document.

Figure 3.15 shows an example usage for a motivation. In the example, whole renarration is created with the purpose of providing an alternative content to content within a source document which is defined as *document1*.

```
{
 "@id": "renarration1",
 "@type": "rn:Renarration",
 "rn:renarratedAt": "2015-09-14T00:00:00Z",
 "rn:hasMotivation": "rn:alternative",
 "rn:renarratedBy": {
  "@id": "http://ex.org/person1",
  "@type": "foaf:Person",
  "foaf:name": "Person One"
 },
 "rn:onSourceDocument": {
  "@id": "document1",
  "@type": "rn:Document"
  },
 "rn:hasRenarrationTransformation":{
  "@id": "renarrationTransformation1",
  "@type": "rn:RenarrationTransformation",
  "rn:sourceDocumentSelection": {
    "@id": "selector1",
    "@type": "rn:ByteSelector",
    "rn:offset": "10",
    "rn:byteNumber": "20"
  },
  "rn:hasNarration": {
   "@id": "narration1",
   "@type": "cnt:ContentAsText",
   "cnt:chars": "content of narration"
  },
  "rn:createdAt": "2015-09-14T22:20:00Z",
  "rn:actionOnDocument": "rn:replace"
 },
 "rn:toTargetDocument": {
  "@id": "renarratedDocument",
  "@type": "rn:Document"
  }
}
```

Figure 3.15. Serialization in JSON-LD format for creating an alternative renarration for a source document

3.1.5. Lists

In some cases, order for a renarration can be very important. A List is a linear data structure where each node is a resource of any type. **rn**:*List* provides an order for a set of resources that are all required to be processed in the order defined. For instance, a user *U* might renarrate a paragraph *P* with concatenation of some text *T1* and *T2* each obtained from different resources on the Web. For such a renarration, the order can be very important because if the order is different for each instance of resource, then the renarration is not the same for different orders.

Renarration(T1, T2, ordered) != *Renarration(T2, T1, ordered).*

In the Renarration Data Model, **rn**:*List* is defined as subClass of **rdf**:*Bag*. The model uses **rn**:*List* for both ordered narrations and ordered selections from documents. An example to ordered selections from documents would be selecting part of a text in a paragraph from a Web page. The first item in the list would be the element selected, in this case the paragraph, using **rn**:*XPathSelector*, and the second item would be selected text by using **rn**:*TextSelector*.

Figure 3.16 shows usage of **rn**:*List* as a renarration of a document. The source document assumed to be existing with **rn**:*onSourceDocument*relationship. There must be an instance of **rn**:*List* whose domain must be an instance of a **rn**:*Renarration* class. In the data model, **rn**:*List* class is defined as a subClass of **rdf**:*Bag* which is also subClass of **rdfs**:*Container*. The object of the relationship of **rn**:*nodes* should be an instance of **rdf**:*List* class. A list must have a list of nodes(resources) of **rn**:*RenarrationTransform* type. The Renarration Data Model recommends using **rn**:*RenarrationTransformation* class for types of each node in a list when the relationship between **rn**:*List* and **rn**:*Renarration* is **rn**:*hasRenarrationTransformation*. Table 3.8 shows explanation of terms used in Figure 3.16.

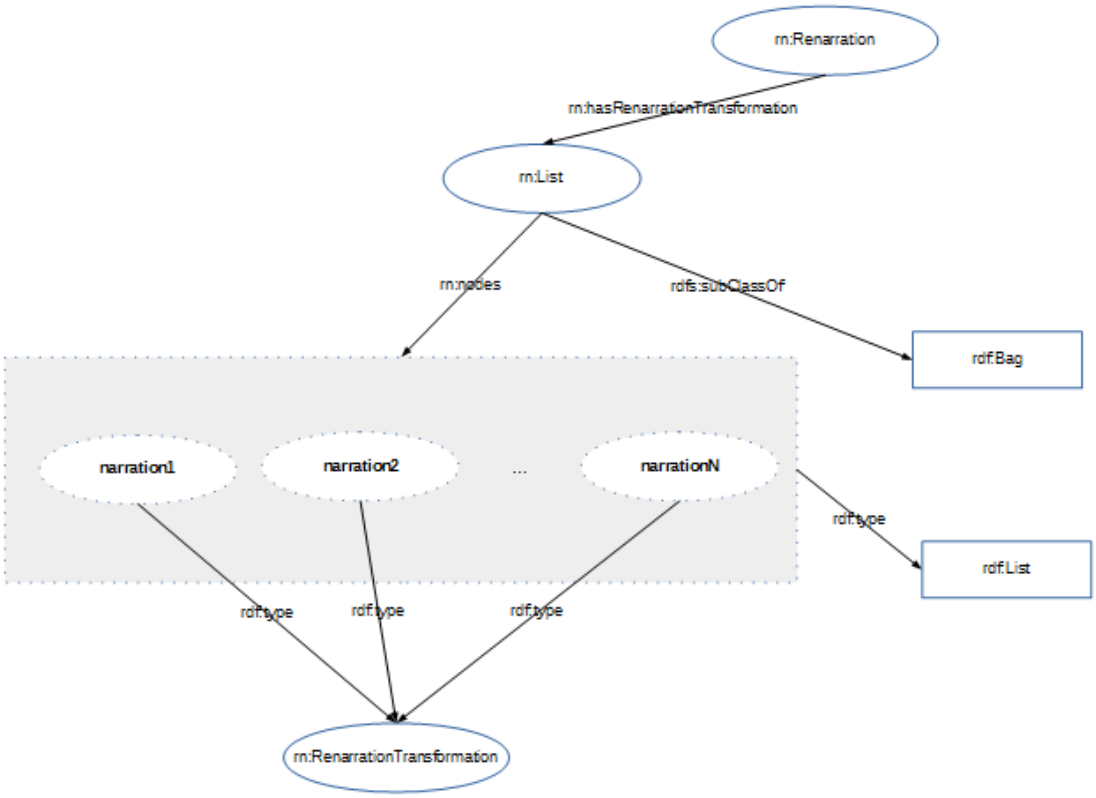

Figure 3.16. Renarration Transformation as a list-like structure.

Table 3.8. Definition of classes and object properties for list structures

Item	Type	Description
rn:*List*	Class	subClass of **rdf**:*Bag*. Defines a list of narrations, which should be interpreted in defined order, for a source document or part of a source document.
rdf:*bag*	Class	subClass of **rdfs**:*Container*. A container that can be used to define a set of elements.
rn:*nodes*	Object Property	The relationship between a resource and an instance of **rdf**:*List* class. When the relationship is **rn**:*RenarrationTransformation* **rn**:*nodes* is used to define a list of Renarration Transformation. When the relationship is **rn**:*hasNarration*, it is used to define a list of narrations(**rn**:*hasNarration*). Lastly, when the relationship is **rn**:*onSourceDocument*, it is used to define nested selections of source documents.

As it is explained in Table 3.8, **rn**:*List* is used for many goals. In Figure 3.16 it is used for a list of Renarration Transformations. The other usage is for **presentation relation**. We use term, presentation relation, for defining the order of narrations. For example, a renarrator might decide to first show a figure right after a text which would make the content easier to understand. Another usage of **rn**:*List* is for **selection order**. The reason behind this is that the content on the Web is mostly nested. For instance a renarrator might want to renarrate some text in a row which is in a table in a div element. In that case, the selection would be such as:

Selection Order: ["div/table/tr", "text"]

Figure 3.17 shows usage of **rn**:*List* for defining a selection order. In the example, document1 is selected by the use of two selectors. First, **rn**:*XPathSelector* is used to select an element in the source document. After that, **rn**:*ByteSelector* is used to select some text within the selected context.

```
{
 "@id": "renarration1",
 "@type": "rn:Renarration",
 "rn:renarratedAt": "2015-09-14T00:00:00Z",
 "rn:hasMotivation": "rn:alternative",
 "rn:renarratedBy": {
  "@id": "http://ex.org/person1",
  "@type": "foaf:Person",
  "foaf:name": "Person One"
 },
 "rn:onSourceDocument": {
  "@id": "document1",
  "@type": "rn:Document"
  },
 "rn:hasRenarrationTransformation":{
  "@id": "renarrationTransformation1",
  "@type": "rn:RenarrationTransformation",
    "rn:sourceDocumentSelection": {
     "@id": "list1",
     "@type": "rn:List",
     "rn:nodes": [
      {
       "@id": "selector1",
       "@type": "rn:XPathSelector"
      },
      {
      "@id": "selector2",
      "@type": "rn:ByteSelector"
      }
     ]
    },
    "rn:hasNarration": {
     "@id": "narration1",
     "@type": "cnt:ContentAsText",
     "cnt:chars": "content of narration"
    },
    "rn:createdAt": "2015-09-14T22:20:00Z",
    "rn:actionOnDocument": "rn:replace"
 },
 "rn:toTargetDocument": {
  "@id": "renarratedDocument",
  "@type": "rn:Document"
  }
}
```

Figure 3.17. Serialization in JSON-LD format for using **rn**:*List* for defining a selection order

4. IMPLEMENTATION

In order to examine the model described in Chapter 3 a prototype built at proof of concept. The prototype implements the transformation of content from a source web document using the rennaration model's alternative specification. The prototype focuses on the use of model introduced in the context of the Web protocols, existing specifications, and the significant work done in the W3C Workgroup on *Web Annotation Data Model* [1]. The main objective of using *Web Annotation Data Model* is to demonstrate the use of relationships among rennarated content and existing content. However, we are of the opinion that *annotation* is a highly significant activity and information on the web, and the *Web Annotation Data Model* work is very important in standardizing the content and protocols related to annotation. This will be demonstrated with examples in the Chapter 5 that evaluates this model.

This chapter is organized as follows: Section 4.1 outlines the technologies used for the implementation of the model discussed in Chapter 1 and proposed in Chapter 3; Section 4.2 presents the high level architecture of the implementation; and Section 4.5 provides the implementation details of the prototype.

4.1. Technologies

The prototype is implemented with publicly available tools and technologies. The Java language is used as the main programming language. The Spring Framework [44] to design Java-based EE application. The graphical user interface (GUI) of SemAnn and *SemRen* are implemented by using JavaScript [45] and HTML (HyperText Markup Language) [46]. In order to enable annotation and renarration of web resources, jsoup [47] Java Library, an open source project distributed under the liberal MIT license, is used. When working with ontologies, Jena Framework [48], an open source Java Framework for building Semantic Web and Linked Data application, is used. Finally, for the storage of annotations and renarrations of web resources, Mongo DB [49], an open source document database, is used.

4.2. System Architecture

The system architecture is divided into six main components :

- **The Web Annotation Data Model** specification describes a structured model and format to enable annotations to be shared and reused across different hardware and software platforms. [1] Fundamentals of the data model is mentioned in Chapter 2.
- **The Renarration Data Model** specifies a framework for creating alternative narrations of source documents by transforming elements of the documents.

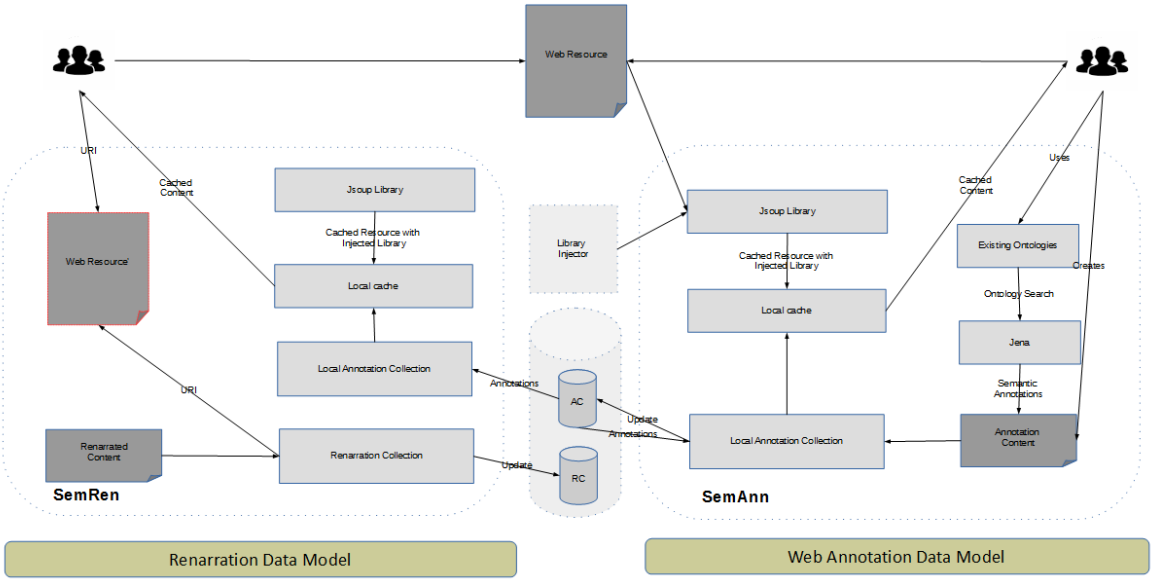

Figure 4.1. System Architecture

- *SemAnn* is an implementation that uses the Web Annotation Data Model. The implementation stores serialized annotations in a collection in a Mongo DB.
- *SemRen* is an implementation that uses the Renarration Data Model. The implementation stores serialized renarrations in a collection in a Mongo DB instance. In addition to storing renarrations in a database, it also enables renarrators to save target(renarrated) documents.
- **Library Injector** is a JavaScript library implemented to be used for both *SemAnn* and *SemRen* implementations. The library is used to enable source documents for creating annotations and renarrations.

- **Storage** is an instance of MongoDB used for storing serialized annotations and renarrations. Serializations are stored in two distinct collections.

4.3. Storage

The implementation uses MongoDB for the storage of serializations of annotations and renarrations. MongoDB is an open-source, document database designed for ease of development and scaling. [49]

Both data models, the Web Annotation Data Model, and the Renarration Data Model, use JSON-LD, a lightweight Linked Data format. Since a record in MongoDB is a document and documents are similar to JSON objects, MongoDB is selected as database for the storage. Another reason for using MongoDB is the support for dynamic schema.

Two different collections are needed in an instance of MongoDB. *annotations* collection is used for storing serialized annotations, and the collection named *renarrations* is used for renarrations.

Figure 4.2. Collections required in MongoDB instance

4.4. Library Injector

Library Injector is a JavaScript library implemented to be used for both *SemAnn* and *SemRen* implementations. The library is used to enable source documents for creating annotations and renarrations.

In order to enable creation of annotations and renarrations on web resources, the

library uses jsoup HTML parser. jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods [47]. By using jsoup external library, the web resource is cached on a local storage. After caching the resource, two main tasks are done by jsoup. The first one is to update all relative urls. The main reason of doing this is to make the page looking exactly similar to original content. The last task is to inject implemented JavaScript library.

The injected library enables three mouse events on cached resource. The events enabled are : onmouseover, onmouseout and onclick. Each event is used differently for each implementation. For instance, when creating annotations, the library talks to annotation collection and queries related annotations for the source document being annotated. The library also includes some functions for finding xpaths of elements. The following is pseudo code of the getPathTo function which is used to find xpaths.

```
e is an element for which xpath is to be found
if e is HTML body then
   return '//html[1]/body[1]'
end if
let P be parent node of e
for each sibling of P do
   if sibling and e are the same elements then
      return concatenation of getPathTo(P), '/' name of e, '[', current sibling index, ']'
   end if
end for
```

Figure 4.3. Pseudo code of getPathTo function for retrieving xpaths of elements

In Figure 4.4, the Web page with http://en.wikipedia.org/wiki/Semantic_Web URI is being annotated. After injecting the library into cached page, the actions

mentioned before are activated on all the elements of the resource. It shows that the element is shown as dashed red border. Before showing the dashed red border, the library copies original style of the element in local cache. When the mouse is hovered on a different element, then original style of the element is copied back, and no border is shown on the element.

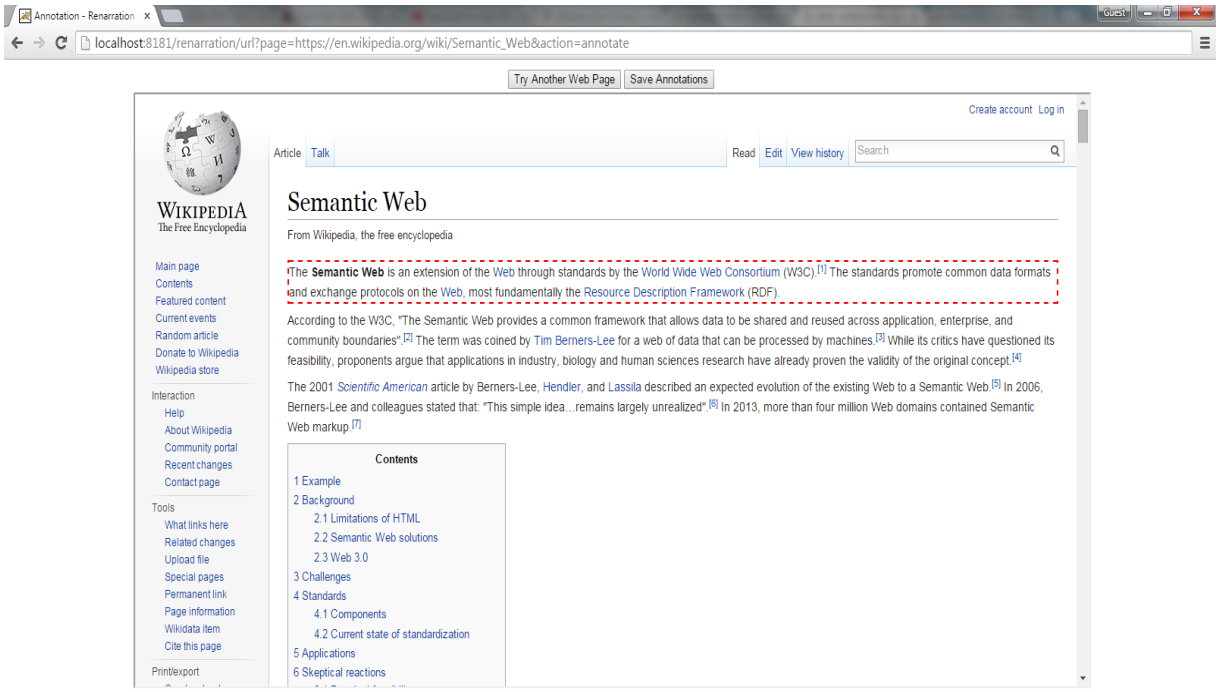

Figure 4.4. Highlighting an element on which mouse is hovered

4.5. Annotation Implementation

The implementation of annotations is named *SemAnn* in the architecture. Using the annotation implementation, annotators can annotate web elements or fragments of elements. The implementation allows different types of annotations such as : embedded content, visual representations, semantic annotations, and annotations which use *DBPedia* resources.

Figure 4.5 shows the system architecture of *SemAnn* framework. The framework uses Web Annotation Data Model as the base model. All annotations are stored in serialized JSON-LD format. Serialization of JSON-LD is done by JavaScript utilities.

When an annotator loads a web resource, the framework injects library using the Library Injector and the Jsoup library. The injection of the library enables all elements of web resources annotatable. The web resource is cached on application server where the deployment is made. The annotator may choose using existing ontologies, existing resources or can choose to create content for the body of the annotation. Created annotations are saved in a MongoDB instance.

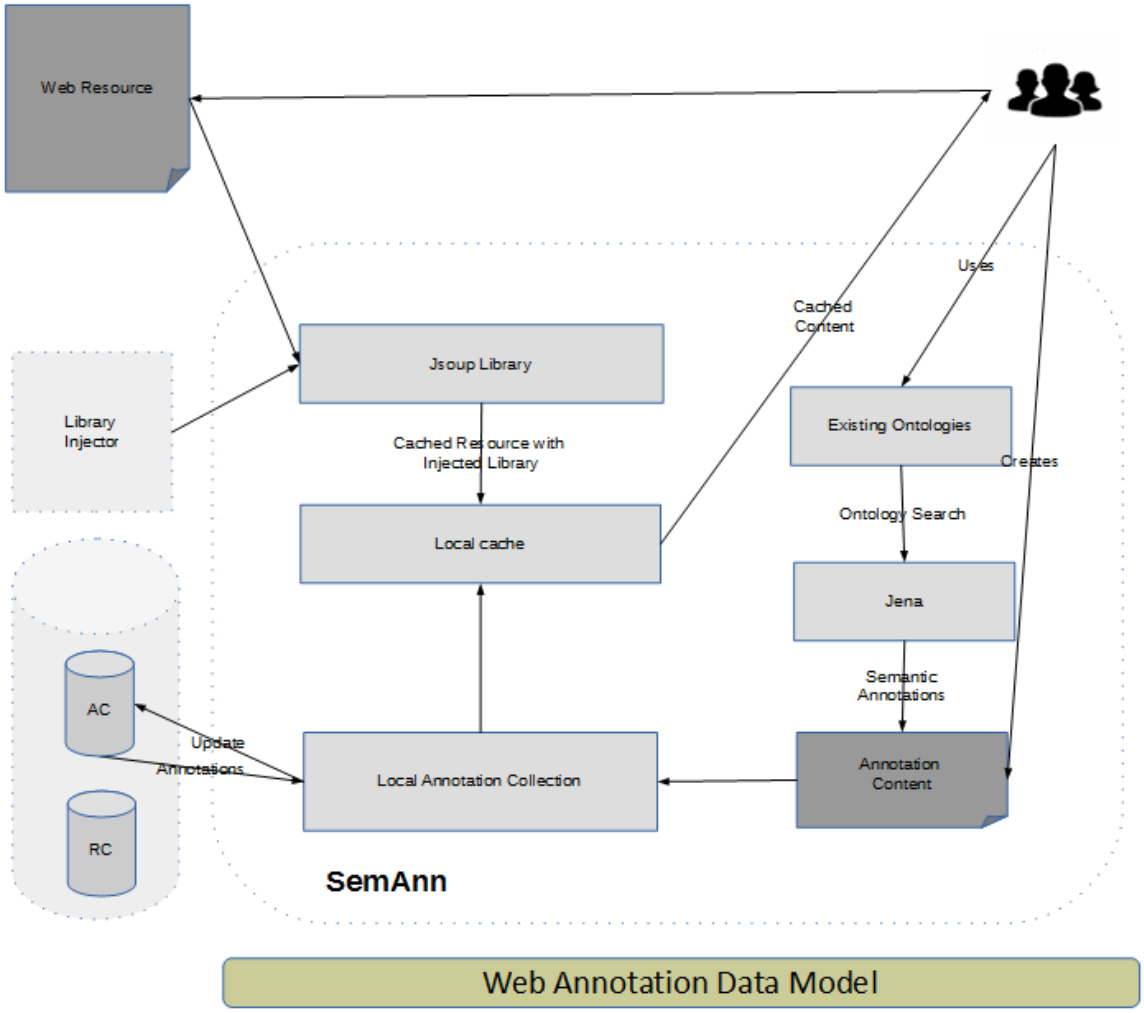

Figure 4.5. System architecture of *SemAnn* framework

4.5.1. Implementation of Annotation Target

Many Annotations refer to part of a resource, rather than all of it, as the Target. Examples include Annotations on an area within an image or video, a range of characters in text, a time segment in an audio file or a slice of a dataset. Equally, the

segment may be the Body of the Annotation, where the comment is given at a particular point in the video, or in a particular paragraph of text. [1]

SemAnn enables annotators selecting elements or fragments of elements of web resources. For instance, an annotator may chose to select all the elements of a web page. This would result selection of body, and XPathSelector would be good candidate for the selection. The value of xpath would be html[1]/body[1]. In the following sections, the details of defining targets are being described.

4.5.1.1. Annotations Using XPath. The implementation uses XPath when enabling users to annotate parts of source documents. When a user hovers an element, the framework highlights the hovered element. When the user clicks on the element a popup is shown for the user to create an annotation. In Figure 4.6, the screenshot of the popup is shown.

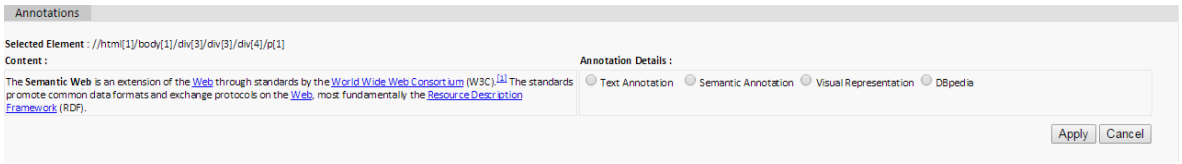

Figure 4.6. Annotation popup when clicked on an element.

In Figure 4.6, the selected element is a paragraph. This can be seen from the xpath on top left. The xpath of the element is "//html[1]/body[1]/div[3]/div[3]/div[4]/p[1]". In the popup the content of the source is also shown on the left. On the right of the popup, four different types of categories are shown. User can create a text annotation, a semantic annotation, or can link the content with a visual representation of it. Tagging with *DBPedia* resources is also allowed.

4.5.1.2. Annotations Targeting Fragments of Elements. *SemAnn* also enables annotators selecting part of elements of web resources. For instance, a user can choose a text in a paragraph or some part of an image. The *SemAnn* keeps track of different element types user is annotating. If a user is annotating an image, the framework allows selection of part of the image. However, if the content of the selection is composed of text

content, the framework allows selection of text.

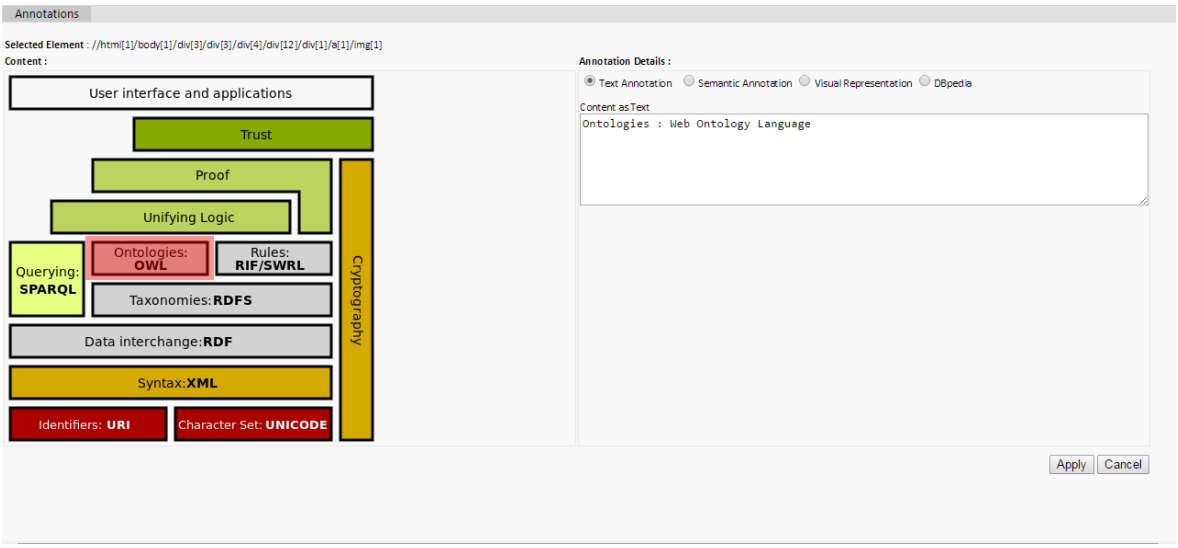

Figure 4.7. Annotation popup when clicked on an image with fragment selected

Figure 4.7 shows selection of part of an image. The box with "Ontologies:OWL" is selected as the target of the annotation. Selected part is shown in red color. The body of the annotation is a text content created by an annotator, and the content is "*Ontologies : Web Ontology Language*".

When such a selection is created by an annotator, the framework keeps track of the position of selected image, and stores values in cache of the local machine. After the annotator selects part of the image, stored values are used to find actual position of the selection relative to these values. In such annotations, the framework uses two stage selections. The first selection uses XPath selector which stored the xpath value of the element. Next selection is the fragment of the image which is stored as "xywh=89,179,140,48".

4.5.2. Implementation of Annotation Body

SemAnn enables annotators to create four types of annotation bodies. Annotators can create annotations by creating embedded content which is usually created by the annotator, by linking visual representations such as images, audio, and/or video. In addition to this, semantic annotations can also be created. The implementation

currently supports FOAF ontology. Lastly, source documents can be annotated by using *DBPedia* resources.

4.5.2.1. Embedded Content Annotations. Semantic Annotation implementation enables users to define embedded content which are then assigned to the body of annotations. Currently, the implementation allows users to create their own content; it doesn't allow usage of external content of other web resources.

In Figure 4.8, an example of embedded textual annotation is shown. On the left the selected content is shown, and the selection will be assigned to the target of the annotation. The content created by the user on the right, will be assigned to the body.

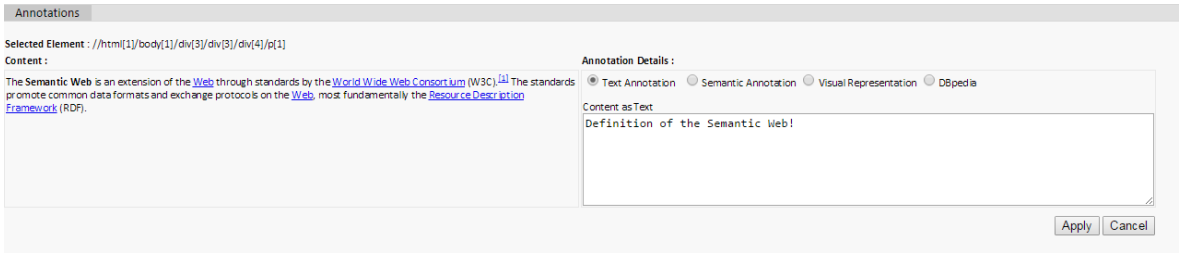

Figure 4.8. Creation of textual body of an annotation

4.5.2.2. Visual Representations. Although most of the content on the Web is text based, visual representations also play very important role. Taking this into consideration, the implementation also enables users to annotate with visual content. For example, the users can annotate with three types of representations which are image, sound and moving image. When users choose these types, the implementation uses DCMI(Dublin Core Metadata Initiative) Metadata Terms [22]. After defining the type, the url of the resource should be defined.

Figure 4.9 shows an example of an annotation for which an image is assigned to the body of the annotation.

4.5.2.3. Semantic Annotations. The prototype allows creation of semantic annotations. Currently, it only supports FOAF Vocabulary Specification [23]. It is required

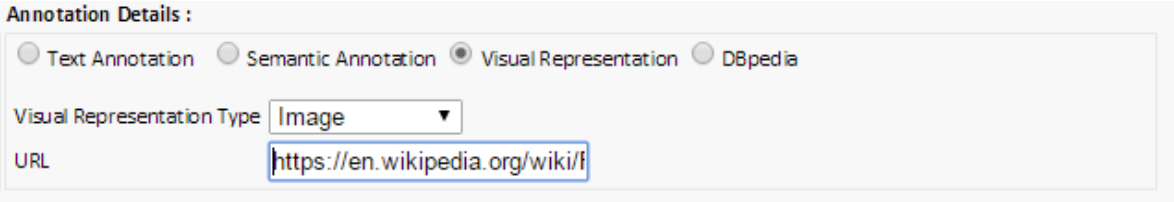

Figure 4.9. Annotation body using visual representations.

for a user to define the type of entity where it corresponds to classes defined in the specification. When a user selects an entity, it is allowed to define subjects and/or objects. In addition defining instances, the users can also define relationships between instances.

Figure 4.10 shows SPARQL query which is used to find all properties when an entity is chosen by the user. In the Figure, `$ONTOLOGY_CLASS` shows the parameter which is defined by the user through the user interface.

The query finds three sets of data and unions them. The first is a set of all properties which are defined as a subclass for the class which is defined by the parameter. The second set is list of properties of the class. Lastly, the properties which are not defined for a specific class - owl:Thing is the set of all individuals.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT distinct ?property where
{
 { ?property a rdf:Property . ?property rdfs:domain ?class . <$ONTOLOGY_CLASS> rdfs:
     subClassOf ?class . }
 UNION
 { ?property a rdf:Property . ?property rdfs:domain <$ONTOLOGY_CLASS> }
 UNION
 { ?property a rdf:Property . ?property rdfs:domain owl:Thing }
}
```

Figure 4.10. SPARQL query to find all properties for a specific class

Figure 4.11 shows an example of defining an instance of foaf:Person. Within the figure, two properties, foaf:firstName and foaf:lastName, are being used.

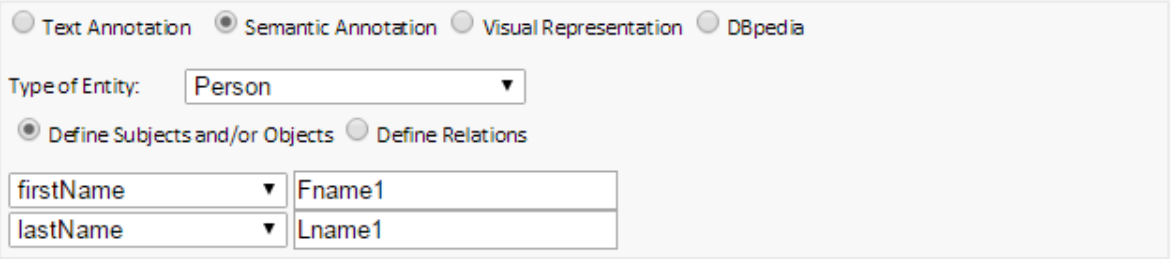

Figure 4.11. Defining an instance for foaf:Person class using foaf:firstName and foaf:lastName data properties

This section assumes that there are two instances of foaf:Person class which are defined in annotation collection in MongoDB. Users can define object properties(relations) between those instances. In Figure 4.12, the type of the entity is chosen as *foaf:Person*, and "Define Relations" is selected. When defining relations, the implementation queries the ontology(FOAF) using SPARQL to find object properties. When the user chooses an object property, the implementation again uses SPARQL to find types of objects allowed for the object property. For instance, in the figure, *foaf:knows* is chosen as the object property. When this is selected, the type of the object can only be an instance of foaf:Person class. After the user selects the type of the object, the annotation collection is queried to find all instances that match the following triple : *"foaf:Person foaf:knows foaf:Person"*.

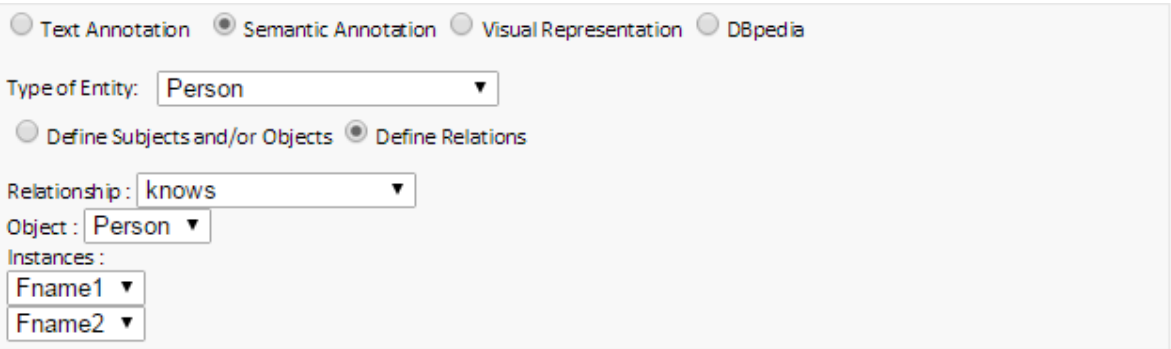

Figure 4.12. Defining foaf:knows object property between Fname1 and Fname2 which are two instances of foaf:Person class

4.5.2.4. Annotations Using *DBPedia* Resources. *DBPedia* is a crowd-sourced community effort to extract structured information from Wikipedia and make this information

available on the Web. *DBPedia* supports sophisticated queries against Wikipedia and to link Web based data sets to Wikipedia data. [50].

The screenshot in Figure 4.13 shows an example of annotation which uses *DBPedia* resources. The content selected is a textual content "*Java (programming language)*". The screeshot shows the result of query of a resource for which object is a Software (`http://DBPedia.org/ontology/Software`) and the value of the data property used for the regular expression is *"Java"*. The result of the query is a list of resources (triples) which satisfy the conditions. In the example, the resource whose URI is, `http://DBPedia.org/resource/Java_(programming_language)`, selected.

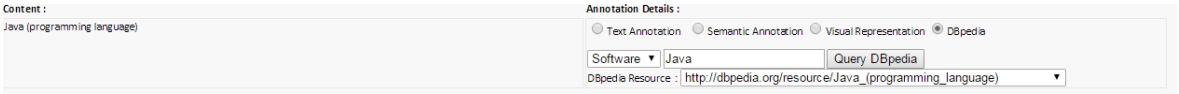

Figure 4.13. Defining semantic annotation using *DBPedia* resources to find list of software via querying *DBPedia* SPARQL service using Java as keyword

SemAnn provides 5 categories to be searched through *DBPedia* SPARQL API. These are listed in Table 4.1.

Table 4.1. Classes and Data Properties Used to Query *DBPedia*

URI of Property	URI of Class
`http://xmlns.com/foaf/0.1/name`	`http://xmlns.com/foaf/0.1/Person`
`http://DBPedia.org/property/name`	`http://DBPedia.org/ontology/Place`
`http://DBPedia.org/property/name`	`http://DBPedia.org/ontology/Animal`
`http://DBPedia.org/property/name`	`http://DBPedia.org/ontology/Software`
`http://DBPedia.org/property/name`	`http://DBPedia.org/ontology/Food`

Figure 4.14 shows the query used within the implemented prototype. Within the query DATA_PROPERTY is the predicate used. For Java programming example, it is replaced with `http://DBPedia.org/property/name` (Line 14). OBJECT_TYPE (Line 15) is `http://DBPedia.org/ontology/Software` with the regular expression: "Java" (Line 16).

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX : <http://dbpedia.org/resource/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT distinct ?s
WHERE
{
 ?s <DATA_PROPERTY> ?o .
 ?s rdf:type <OBJECT_TYPE> .
 FILTER (regex(?o, \"REGEX\", \"i\"))
} ORDER BY ?s
```

Figure 4.14. Query template used for *DBPedia* SPARQL Service

4.6. Renarration Implementation

The implementation is named *SemRen* in the architecture. By using the renarration implementation renarrators can renarrate web resources. The implementation allows different types of renarration creation such as creating embedded content, renarrations which use visual representations (image, audio, video).

The implementation also allows users to use annotations. When an annotation is created for a web resource, during renarration process, renarrator can see all annotations. In addition to annotation data, reference data is extracted from *DBPedia* if an annotation is created using *DBPedia* resources.

Figure 4.15 shows the system architecture of *SemRen* framework. The framework uses an ontology created to model renarrations of web resources which is covered in Chapter 3. Very similar to *SemAnn*, *SemRen* framework also uses Jsoup library in order to cache the web resource. The framework uses annotation collections which can help renarration of web resources. The contents renarrated by users are stored in renarration collection in an MongoDB instance. When the users finish with the renarration, another resource is created which can be accessed via a URI.

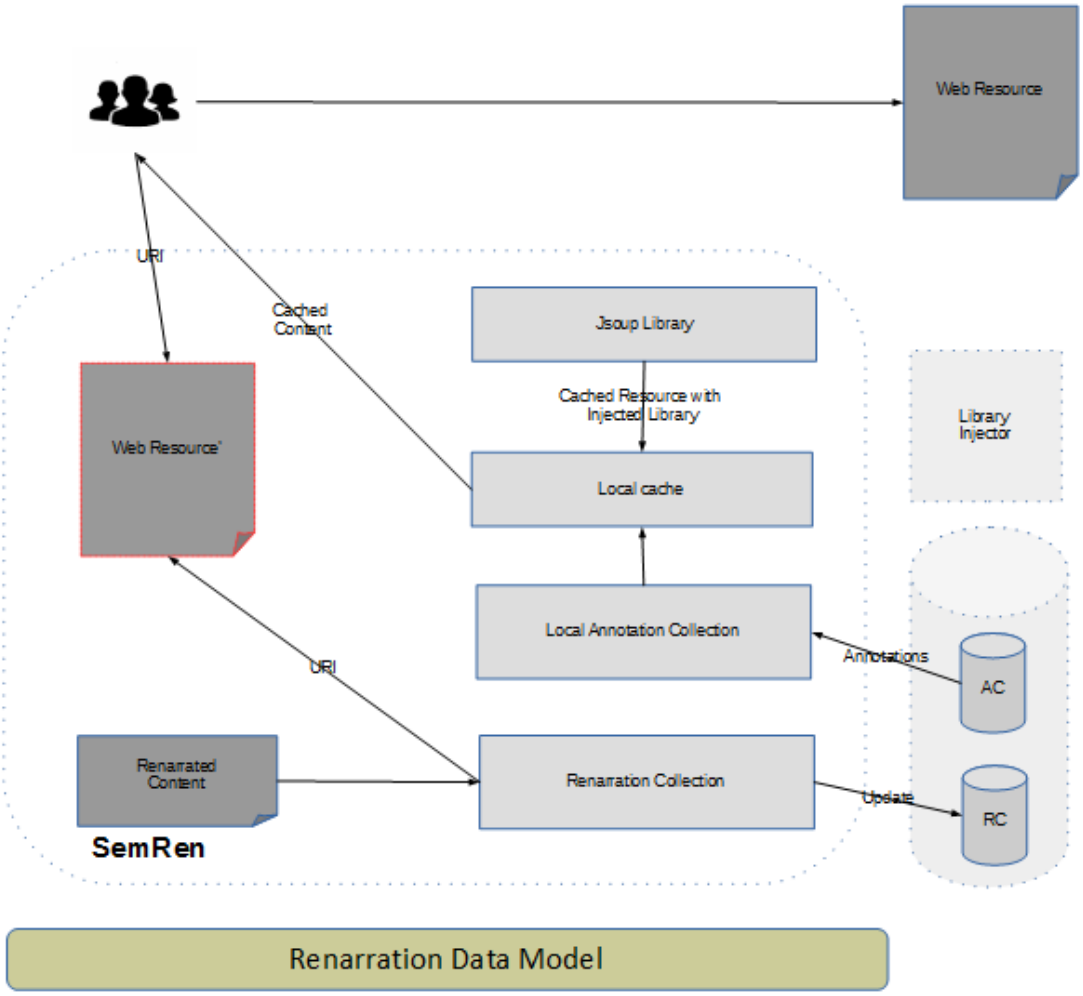

Figure 4.15. System architecture of *SemRen* framework.

4.6.1. Renarration Actions

SemRen allows two types of renarration actions, remove and replace. These are explained in the following sections.

4.6.1.1. Remove. When renarrating content in source documents, removing some content can be so useful. The reasons behind removing a content can be because of the content is not suitable for the target audience for which the renarration is being done. The *SemRen* framework allows renarrators to remove elements of web pages. For instance, in Figure 4.16 a div element is choosen for renarration transform. The xpath of

the element is : //html[1]/body[1]/div[3]/div[3]/div[4]/div[1]. The renarrator decides to remove the content of the chosen element. The action of the renarration transform is chosen as 'Remove'.

Figure 4.16. Defining removal of selected content via user interface

When this transform is applied, the element is hidden from the user. It's not removed from the source document not to change xpaths of elements. Currently, the prototype just updates the style of the element and sets its display to 'none'.

4.6.1.2. Replace. Another action that *SemRen* Framework allows on elements of source documents is replace. Renarrators can choose to replace the content in source documents with text content, images, audios, and/or videos. In Figure 4.17, 'Replace' is chosen as the action of Renarration Transform. The renarrator can choose four types content to be replaced with the source content.

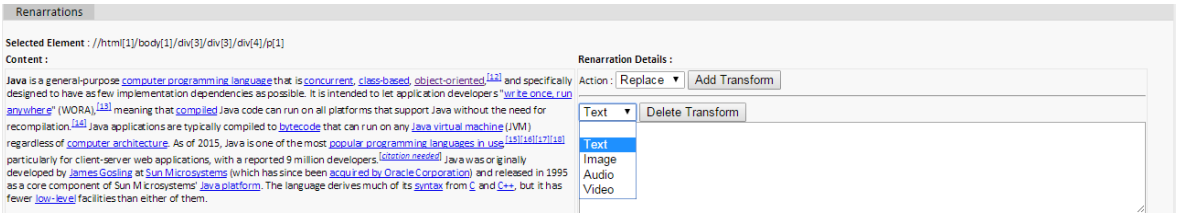

Figure 4.17. Defining replacement of selected content with a text via user interface

The *SemRen* framework, replaces source content with target content within the selected element. In other words, it just creates necessary HTML tags within the selected context. For instance, in Figure 4.17 all content defined by renarrator is to be replaced with html content of element with xpath://html[1]/body[1]/div[3]/div[3]/div[4]/p[1].

The replacement of source content can also be done by selection of content within the selected element. Figure 4.18 shows, the replacement of text '2015' with '2016' within the selected element.

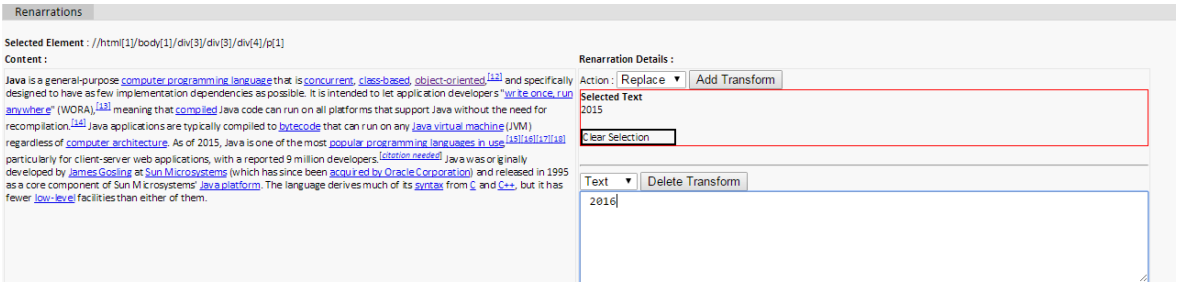

Figure 4.18. Replacement of text '2015' with '2016' using nested selection

The prototype currently doesn't allow replacement of elements from which xpath can change. An example, to this could be replacing an image with a paragraph. All the xpaths of elements after the insertion of new paragraph can change. The reason behind this restriction is not to change the xpath of the elements while allowing renarration of source documents, and being able to keep relationships between source documents and renarrated content.

4.6.2. Renarration Transforms as Collections

When renarrating source documents, not always one to one relationship is used. Often, it is very important for a user interface to allow renarration of an element by using more than one transform. *SemRen* framework allows collections of transforms as a renarration transform. Collections of renarration transforms are stored as ordered instances of rdf:List class.

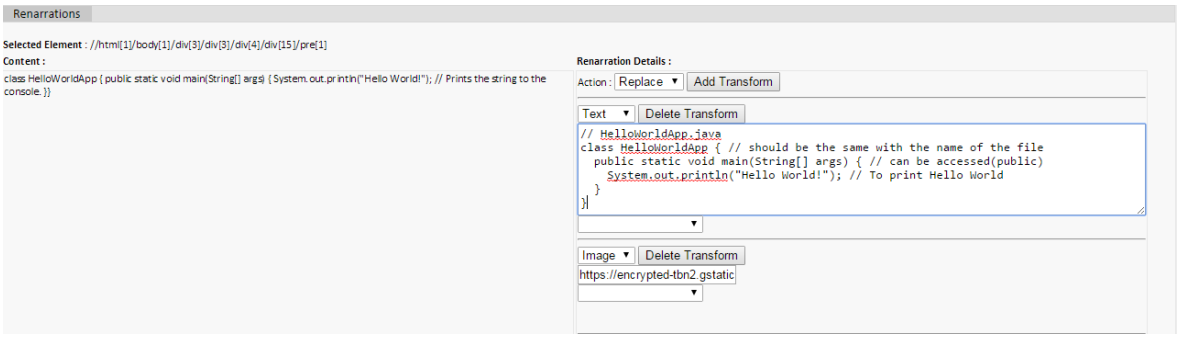

Figure 4.19. Defining a list of renarration transforms whose elements are textual content and an image

Figure 4.19 shows an example of a list of Renarration Transforms. In the example,

the content which can be accessed via xpath : //html[1]/body[1]/div[3]/div[3]/div[4]/div[15]/pre[1] is replaced with a text content and an image. The text content aims to give some explanations for each line of code. The image [65] illustrates that a class file is created from .java file and created class file can be executed in different platforms.

The target of the renarration in source document can be seen in Figure 4.20.

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Prints the string to the console.
    }
}
```

Figure 4.20. An example java code to be renarrated

Figure 4.21 below shows the renarration of source content when it is replaced by a text and an image.

```
// HelloWorldApp.java
class HelloWorldApp { // should be the same with the name of the file
  public static void main(String[] args) { // can be accessed(public)
    System.out.println("Hello World!"); // To print Hello World
  }
}
```

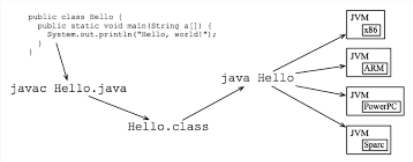

Figure 4.21. Replacement of java code with comments added as well as an image showing how it is compiled

4.6.3. Usage of Annotations in *SemRen* Framework

When doing renarrations, *SemRen* framework also allows renarrators using data created in annotations created by annotators. When a user renarrates a web page through the framework, first the annotations are queried for the resource. The renarrator may choose to use data from annotations.

The prototype also retrieves referenced data when it's available. Currently, when an annotation is created using *DBPedia* resources, data type property `http`:

`//DBPedia.org/ontology/abstract` is used to fetch abstract information related to the annotation.

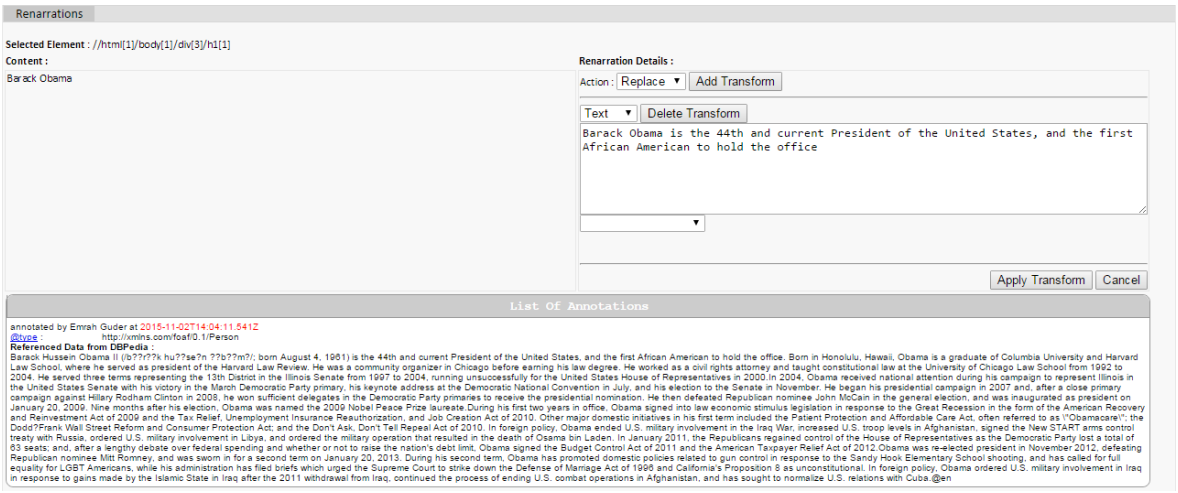

Figure 4.22. Renarration transform using reference data from an annotation.

In Figure 4.22, an annotation is already created by an annotator and the text "Barack Obama" is annotated by using `http://DBPedia.org/page/Barack_Obama` *DBPedia* resource. *SemRen* framework, uses *DBPedia* SPARQL service to query abstract information for the resource used in the annotation. When renarrator chooses to use some of the data from annotation itself or from referenced data, a link is created between the renarration transform and the annotation.

4.6.4. Target Audience and Deployment

Renarrations targeted for an audience can be defined by using *SemRen* framework. The audience used within the framework is modeled in the Renarration Data Model as a subclass of skos:Concept. The audience used within the framework is specifically defined for use cases for the evaluation of the model and the implementation which are all covered in the next chapter.

When a renarrator saves transform(s) of a renarration, the framework shows an interface for defining a target audience and whether the page is desired to be saved within the application server storage area. If a name for deployment is defined, the content of the renarration is saved in HTML format.

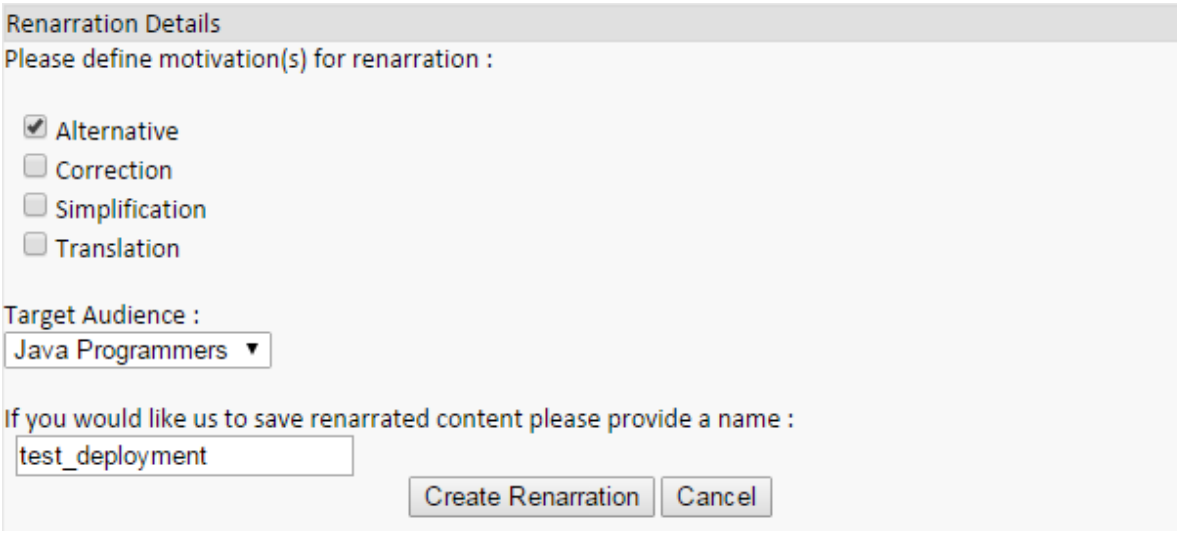

Figure 4.23. Defining target audience and deployment name for renarration of a web resource.

5. EVALUATION

In this chapter, we are evaluating our model using some pages which are manually prepared. The pages are created using HTML. Section 5.1 discusses the adequacy of the Renarration Data Model using different types of renarrations. In Section 5.2, we have experimented different types of renarrations using the prototype we have developed.

5.1. Evaluation of the Renarration Data Model

In order to evaluate the proposed data model, we've created a suite of test cases to prove that different types of renarrations can be defined using the data model. Test cases can be found in a repository created on GitHub [51]. The url of the repository for test cases is : https://github.com/EmrahGuder/Renarration/tree/master/`Test_Cases`.

The following is a list of for the subset of selected test cases.

(i) Replacement of paragraph with paragraph
(ii) Replacement of paragraph with audio
(iii) Removal of content
(iv) Replacement using content from an annotation
(v) Insertion of new content between two elements

In addition to evaluation with above test cases, we've also evaluated the data model by using a manually created web page about FIR (First Information Report) [52] which is a document prepared by police organizations in Bangladesh, India and Pakistan.

5.1.1. Evaluation of Proposed Data Model Using Test Cases

5.1.1.1. Replacement of paragraph with paragraph. The motivation behind this test case is to demonstrate that replacement of text content can be defined using the pro-

posed data model. For this purpose, we assume that the following content "Text1" is to be renarrated with "Text2".

```
<html>
 <body>
  <p>Text1</p>
 </body>
</html>
```

Figure 5.1. HTML code of a web page consisting of a paragraph with textual content "Text1"

Renarration of textual content "Text1" with "Text2" can be defined by the Renarration Data Model in JSON-LD format as in Figure 5.2. It shows that the model can define text selections by using a text selector.

```
{
 "transform": {
  "@id": "20ca990d-b233-4047-b40a-258a8278fada",
  "@type": "rn:RenarrationTransformation",
  "sourceSelection": {
   "@id": "fb60d831-7c18-4e83-b8da-341e46482106",
   "@type": "rn:XPathSelector",
   "value": "#xpointer(//html[1]/body[1]/p[1])"},
  "createdAt": "2015-11-20T13:07:27.967Z",
  "action": "rn:Replace",
  "narration": {
   "@id": "554f2f73-0d93-4426-851f-56613d1194d1",
   "@type": "cnt:ContentAsText",
   "content": "Text2"
  },
  "targetSelection": {
   "@id": "6322109a-e221-4c61-9e51-5d31287a4dc9",
   "@type": "rn:XPathSelector",
   "value": "#xpointer(//html[1]/body[1]/p[1])"
  }
 }
}
```

Figure 5.2. A renarration specification in JSON-LD of a web page for which "Text1" is replaced with "Text2"

After applying transformations on source document, the target document can be constructed as in Figure 5.3.

```
<html>
 <body>
  <p>Text2</p>
 </body>
</html>
```

Figure 5.3. HTML code of a web page renarrated via replacing "Text2" with "Text1" in original content

5.1.1.2. Replacement of paragraph with audio. The motivation behind this test case is to demonstrate that an element in a Web page can be replaced with an element of different type such as replacing a paragraph with an audio element. For this purpose, we assume that the HTML code of original web page is the same as in Figure 5.1. The aim of the renarration is to replace the paragraph with an audio with the url of http://audio1.ogg.

Renarration of textual content "Text1" with audio can be defined by the Renarration Data Model in JSON-LD format as in Figure 5.4. It shows that the model can define replacement of different types of HTML elements. It also shows that selections can be done using rn:XPathSelector. After transformation applied on source Web page, the content of the renarrated page can be seen in Figure 5.5.

5.1.1.3. Removal of content. As we define renarration, the process of rewriting of web resources, keeping relations between elements of resources becomes crucial when the process changes xpath of elements. One of the renarration trasformation that changes xpath of elements is removal of them. So, the motivation of this section is to prove that the proposed model can handle removal of elements. Figure 5.6 shows the source in HTML format. Originally, there are three paragraph elements in the source. The renarration aims to remove second paragraph and to create an alternative for the third paragraph with textual content 'Text4'.

Figure 5.7 shows the removal process defined with the proposed ontology in JSON-LD format.

```
{
 "@type": "rn:Renarration",
 "renarratedAt": "2015-11-20T13:08:00.650Z",
 "source": {
  "@id": "replacement_of_paragraph_with_audio_original.html",
  "@type": ["foaf:Page", "rn:Document"]
  },
 "transform": {
  "@id": "20ca990d-b233-4047-b40a-258a8278fada",
  "@type": "rn:RenarrationTransformation",
  "sourceSelection": {
   "@id": "6322109a-e221-4c61-9e51-5d31287a4dc9",
   "@type": "rn:XPathSelector",
   "value": "#xpointer(//html[1]/body[1]/p[1])"
   },
  "createdAt": "2015-11-20T13:07:27.967Z",
  "action": "rn:Replace",
  "narration": {
   "@id": "http://audio1.ogg",
   "@type": "dctypes:Sound",
   "format": "audio/ogg"
   },
 "targetSelection": {
  "@id": "fb60d831-7c18-4e83-b8da-341e46482106",
  "@type": "rn:XPathSelector",
  "value": "#xpointer(//html[1]/body[1]/audio[1])"}
 },
 "target": {
  "@id": "replacement_of_paragraph_with_audio_renarrated.html",
  "@type": ["foaf:Page", "rn:Document"]
 }
}
```

Figure 5.4. A renarration specification in JSON-LD of a web page for which "Text1" is replaced with audio with url "http://audio1.ogg" in audio/ogg format

```
<html>
 <body>
  <audio controls>
   <source src="http://audio1.ogg" type="audio/ogg">
   Your browser does not support the audio tag.
  </audio>
 </body>
</html>
```

Figure 5.5. HTML code of a web page renarrated via replacing "Text2" with an audio

```
<html>
 <body>
  <p>Text1</p>
  <p>Text2</p>
  <p>Text3</p>
 </body>
</html>
```

Figure 5.6. HTML code of a web page consisting of three paragraphs

Figure 5.8 shows that second paragraph whose xpath is //html[1]/body[1]/p[2] is removed. Also third paragraph in original page with 'Text3' textual content is renarrated with 'Text4' where the renarrated content's xpath is //html[1]/body[1]/p[2].

This test case shows that the proposed data model can express, removal of content, and transforms being done after removal of content. This evaluation also shows that when a transform which can change xpath of elements is done, all transformations after the one changing xpaths, must have *targetSelection* object property.

5.1.1.4. Replacement using content from an annotation. In this test case, we have evaluated usage of annotation in a renarration. We have assumed that the annotation is already created by an annotator. The annotation created assumed to be a text annotation with textual content 'Annotation Text'.

For this test case we assume that original page is the same as in Figure 5.6, and the annotation is assumed to be created on the second paragraph whose content is 'Text2'.

Annotation created can be seen in Figure 5.9 in JSON-LD format. Lines between 23 and 25 shows that the annotation is created for 'Text2', and the line 24 shows that annotation is an embedded content type annotation with value of 'Annotation Text'.

We assume that renarrator uses content from annotation and renarrates the second paragraph. Line 27 in Figure 5.10 shows that 'Annotation Text' is text. Lines between 28 and 30 indicates that this content is accessed from annotation.

```
{
 "@type": "rn:Renarration",
 "source": {
  "@id": "transform_after_removal_of_content_original.html",
  "@type": ["foaf:Page", "rn:Document"]
 },
 "transformList": {
  "@type": "rn:List",
  "nodes": [
   {
    "@id": "19ca990d-b233-4047-b40a-258a8278fada",
    "@type": "rn:RenarrationTransformation",
    "sourceSelection": {
     "@id": "6322109a-e221-4c61-9e51-5d31287a4dc9",
     "@type": "rn:XPathSelector",
     "value": "#xpointer(//html[1]/body[1]/p[2])"
    },
    "createdAt": "2015-11-20T13:07:27.967Z",
    "action": "rn:Remove"
   },
   {
    "@id": "20ca990d-b233-4047-b40a-258a8278fada",
    "@type": "rn:RenarrationTransformation",
    "sourceSelection": {
     "@id": "6322109a-e221-4c61-9e51-5d31287a4dc9",
     "@type": "rn:XPathSelector",
     "value": "#xpointer(//html[1]/body[1]/p[3])"
    },
    "createdAt": "2015-11-20T13:07:27.967Z",
    "action": "rn:Replace",
    "narration": {
     "@id": "243b6a94-d16f-4c30-aff4-7800be645415",
     "@type": "cnt:ContentAsText",
     "content": "Text4"
    },
    "targetSelection": {
     "@id": "fb60d831-7c18-4e83-b8da-341e46482106",
     "@type": "rn:XPathSelector",
     "value": "#xpointer(//html[1]/body[1]/p[2])"
    }
   } ]
 },
 "target": {
  "@id": "transform_after_removal_of_content_renarrated.html",
  "@type": ["foaf:Page", "rn:Document"]
 }
}
```


Figure 5.7. Renarration done on original page in JSON-LD format for transformations on two paragraphs including remove and replace

```
<html>
 <body>
  <p>Text1</p>
  <p>Text4</p>
 </body>
</html>
```

Figure 5.8. HTML code of renarrated page when a paragraph is replaced after removal of previous paragraph

```
{
        "@id": "88b64849-fbdb-4157-8393-08bb509bcd80",
        "@type": "oa:Annotation",
        "annotatedAt": "2015-11-20T11:46:48.747Z",
        "serializedAt": "2015-11-20T11:46:48.747Z",
        "annotatedBy": {
                "@id": "749bf220-4e8b-481d-8b7a-e5e735e684f1",
                "@type": "foaf:Person",
                "name": "Annotator"
        },
        "body": {
                "@id": "d1f273c6-3f93-4e37-b93c-8b58d 01017ec",
                "@type": "oa:EmbeddedContent",
                "value": "Annotation Text",
                "format" : "text/plain"
        },
        "target": {
                "@id": "c25cf552-2bcf-40c8-8424-f46b77287595",
                "@type": "oa:SpecificResource",
                "selector": {
                        "@id": "7b0fa8bc-d7c4-4685-81e5-6a90a15d7241",
                        "@type": "oa:TextQuoteSelector",
                        "exact": "Text2",
                        "prefix": "Text1",
                        "suffix": "Text3"
                },
                "source": {
                        "@id": "transform_after_removal_of_content_original.html",
                        "@type": "foaf:page"
                }
        }
}
```

Figure 5.9. Annotation in JSON-LD created for Text2 using oa:TextQuoteSelector

```
{
 "@id": "fbae30fa-2687-499e-9a85-1b12d75367d7",
 "@type": "rn:Renarration",
 "renarratedAt": "2015-11-20T13:08:00.650Z",
 "renarrator": {
  "@id": "e8f551f8-33ea-429d-8932-9f16bd8ef3b7",
  "@type": "foaf:Person",
  "name": "A Person"
 },
 "source": {
  "@id": "using_annotation_content_original.html",
  "@type": ["foaf:Page", "rn:Document"]
 },
 "transform":      {
  "@id": "20ca990d-b233-4047-b40a-258a8278fada",
  "@type": "rn:RenarrationTransformation",
  "sourceSelection": {
   "@id": "6322109a-e221-4c61-9e51-5d31287a4dc9",
   "@type": "rn:XPathSelector",
   "value": "#xpointer(//html[1]/body[1]/p[2])"
  },
  "createdAt": "2015-11-20T13:07:27.967Z",
  "action": "rn:Replace",
  "narration": {
   "@id": "243b6a94-d16f-4c30-aff4-7800be645415",
   "@type": "cnt:ContentAsText",
   "content": "Annotation Text",
   "accessedFrom": {
    "@id": "88b64849-fbdb-4157-8393-08bb509bcd80",
    "@type": "oa:Annotation"
   }
  },
  "targetSelection": {
   "@id": "fb60d831-7c18-4e83-b8da-341e46482106",
   "@type": "rn:XPathSelector",
   "value": "#xpointer(//html[1]/body[1]/p[2])"
  }
 },
 "target": {
  "@id": "using_annotation_content_renarrated.html",
  "@type": ["foaf:Page", "rn:Document"]
 }
}
```

Figure 5.10. Renarration done on original page in JSON-LD format for replacing second paragraph by using content from an annotation

Assuming that renarrations transforms applied on source page, and generated page is deployed as 'using_annotation_content_renarrated.html', renarrated page would be as in Figure 5.11.

```
<html>
 <body>
  <p>Text1</p>
  <p>Annotation Text</p>
  <p>Text3</p>
 </body>
</html>
```

Figure 5.11. Renarrated page for which second paragraph's content is referenced from an annotation

5.1.1.5. Insertion of new content between two elements. Insertion of new content can be very useful during the process of renarration. Renarrators may choose to insert new content at specified locations in web resources. In order to evaluate if the proposed model can define such operations, we have assumed that renarrator inserts a new paragraph between two paragraphs.

The source page for which a paragraph should be inserted between two paragraphs can be seen in Figure 5.12.

```
<html>
 <body>
  <p>Text1</p>
  <p>Text3</p>
 </body>
</html>
```

Figure 5.12. HTML code of a web page consisting of two paragraphs

A renarration specification in JSON-LD of the web page describing insertion of new paragraph can be seen in Figure 5.13. Lines between 7 and 42 in the figure shows the insertion of new paragraph.

```
{
 ...
 "source": {
  "@id": "insert_between_two_elements_original.html",
  "@type": ["foaf:Page", "rn:Document"]
 },
 "transform":      {
  "@id": "20ca990d-b233-4047-b40a-258a8278fada",
  "@type": "rn:RenarrationTransformation",
  "sourceSelection": {
   "@id": "6322109a-e221-4c61-9e51-5d31287a4dc9",
   "@type": "rn:BetweenSelector",
   "between": {
     "@id": "243b6a65-d16f-4c30-aff4-7800be645415",
         "@type": "rn:List",
         "nodes": [
          {
            "@id": "7544109a-e221-4c61-9e51-5d31287a6bh8",
                "@type": "rn:XPathSelector",
                "value": "#xpointer(//html[1]/body[1]/p[1])"
          },
          {
            "@id": "7544109a-e221-4c61-9e51-5d31287a6bh9",
                "@type": "rn:XPathSelector",
                "value": "#xpointer(//html[1]/body[1]/p[2])"
          }
         ]
    }
   },
  "createdAt": "2015-11-20T13:07:27.967Z",
  "action": "rn:Insert",
  "narration": {
   "@id": "243b6a94-d16f-4c30-aff4-7800be645415",
   "@type": "cnt:ContentAsText",
   "content": "Text2"
   },
  "targetSelection": {
   "@id": "fb60d831-7c18-4e83-b8da-341e46482106",
   "@type": "rn:XPathSelector",
   "value": "#xpointer(//html[1]/body[1]/p[2])"
   }
  },
 "target": {
  "@id": "insert_between_two_elements_renarrated.html",
  "@type": ["foaf:Page", "rn:Document"]
 }
}
```

Figure 5.13. A renarration specification in JSON-LD of the web page describing insertion of new paragraph

After applying transformations defined in Figure 5.13 on original page, renarrated page can be constructed as in Figure 5.14.

```
<html>
 <body>
  <p>Text1</p>
  <p>Text2</p>
  <p>Text3</p>
 </body>
</html>
```

Figure 5.14. HTML code of renarrated web page including inserted paragraph

5.1.2. Evaluation Using FIR Web Page

In order to demonstrate that the proposed model can also be used for renarration of pages on the Web, we have manually created a web page using content on Wikipedia [52]. The content is about FIR (First Information Report) which is a document prepared by police organizations in Bangladesh, India, and Pakistan. The content of the resource can be seen in the following figure. Even though the page is just composed of text content, there are nested elements as there would be in real web pages. During the evaluation of the model, we have assumed that HTML document is well formed [12]. Figure 5.15 shows the content of the FIR page.

The purpose of the renarration is assumed to be that the page can be constructed for low literate people so that the content can be easily consumable.

For the first transform, it is assumed that renarrator decides to renarrate a sentence. The content to be renarrated is "*It is generally a complaint lodged with the police by the victim of a cognizable offense or by someone on his or her behalf, but anyone can make such a report either orally or in writing to the police*" [52]. Looking at the structure of the sentence, it may be hard to understand for low literate. We have assumed that this sentence is replaced with the following content: "*If you have been a victim, or a witness to a crime or someone who has information of a crime and want to punish the criminal you can report to police.*" [53]. Figure 5.16 shows the replacement of this sentence. The action also shows that nested selection is used. The

Figure 5.15. Web page created using content from Wikipedia about how to file first information report

way that this is expressed is done by using first XPathSelector (Lines between 9 and 13) and then PrefixSuffixSelector (Lines between 14 and 20).

For the second transform, we have assumed that the renarrator decides the second paragraph in original page does not help low literate to understand the content. So, the renarrator decides to remove this paragraph. Lines between 30 and 40 in Figure 5.16 shows how this is expressed within the model specification.

For the next transform, we have assumed that an annotator created an annotation for text content "cognizable offence" in original page. The annotation created assumed to be a textual annotation with the following content :

"*Generally, cognisable offence means a police officer has the authority to make an arrest without a warrant. The police is also allowed to start an investigation with or without the permission of a court*" [54].

Lines between 5 and 22 in Figure 5.17 shows the selection after the third paragraph. The meaning of these lines is that the content of the narration will be inserted after the third paragraph (whose xpath is "//html/body/div[1]/p[3]") in the source. Lines between 33 and 35 in the figure, shows that the inserted content is referenced from an annotation.

Lastly, we have assumed that the renarrator decides that it can be easier to understand if the content about how to file a first information report is replaced with an interactive resources. For the interactive resource, we have used content from Prezi [53]. The content interactively guides user what to do in such a situation. Line 17 in the Figure 5.18 shows that DCMI Metadata Term can be used to define an interactive resource.

After all transforms applied on the source content, and assuming that the page is saved, the renarrated page can be constructed as in Figure 5.19.

5.2. Evaluation Using Implemented Prototype

In previous section, we have evaluated the model if it is expressive enough to define different types of renarrations. In this section, we have created two web pages in text/html format, and used the implemented prototype.

The first use case, includes renarration of a web page which is about method overriding in C++, an object oriented programming language. The purpose of the

```
{
 "@id": "3cc9b52d-26db-4ccc-8450-0baec2435e99",
 "@type": "rn:RenarrationTransformation",
 "action": "rn:Replace",
 "sourceSelectionList": {
  "@id": "77d9f63f-a806-4e06-8022-66ff4fc1ffdd",
  "@type": "rn:List",
  "nodes": [
   {
    "@id": "e0c0956d-3914-462 1-bf0e-76936d68f2fb",
    "@type": "rn:XPathSelector",
    "value": "#xpointer(//html/body/div[1]/p[1])"
   },
   {
    "@id": "2f305684-d300-4493-88ce-217cfb836d49",
    "@type": "rn:PrefixSuffixSelector",
    "prefix": "Pakistan when they receive about the commission of a cognizable offence.",
    "suffix": "",
    "text": "It is generally a complaint lodged with the police by the victim of a cognizable offense or by someone on his or her behalf, but anyone can make such a report either orally or in writing to the police."
   }
  ]

"narration": {
 "@id": "c41e9d39-2d09-442d-9ea1-018986c53801",
 "@type": "cnt:ContentAsText",
 "content": "If you have been a victim, or a witness to a crime or someone who has information of a crime and want to punish the criminal you can report to police."
},
"createdAt": "2015-11-05T13:48:00Z"
},
{
 "@id": "3cc9b52d-26db-4ccc-8450-0baec2435f99",
 "@type": "rn:RenarrationTransformation",
 "action": "rn:Remove",
 "sourceSelection": {
  "@id": "77d9f63f-a806-4e06-8022-66ff4fc1ffed",
  "@type": "rn:XPathSelector",
  "value": "#xpointer(//html/body/div[1]/p[2])"
 },
 "createdAt": "2015-11-05T13:50:00Z"
}
```

Figure 5.16. A renarration transform specification in JSON-LD of a web page describing replacement and removal of two paragraphs

```
{
 "@id": "3cd9b52d-26db-4ccc-8450-0baec2455f99",
 "@type": "rn:RenarrationTransformation",
 "action": "rn:Insert",
 "sourceSelectionList": {
  "@id": "6322109a-e221-4c61-9e51-5d31287a4dc9",
  "@type": "rn:BetweenSelector",
  "between": {
   "@id": "243b6a65-d16f-4c30-aff4-7800be645415",
   "@type": "rn:List",
   "nodes": [
   {
    "@id": "7544109a-e221-4c61-9e51-5d31287a6bh8",
        "@type": "rn:XPathSelector",
        "value": "#xpointer(//html/body/div[1]/p[3])"
   },
   {
        "@id": "7544109a-e221-4c61-9e51-5d31287a6bh8",
        "@type": "rn:NoSelector"
   }
   ]
  }
 },
 "targetSelection": {
  "@id": "77d9f63f-a806-4e06-8022-66ff4fc1ffed",
  "@type": "rn:XPathSelector",
  "value": "#xpointer(//html/body/div[1]/p[2])"
 },
 "narration": {
  "@id": "c41e9d39-2d09-442d-9ea1-018986c53801",
  "@type": "cnt:ContentAsText",
  "content": "Generally, cognisable offence means a police officer has the authority
      to make an arrest without a warrant. The police is also allowed to start an
      investigation with or without the permission of a court.",
  "accessedFrom": {
   "@id": "f23ede35-837b-4316-a6a1-110c359e7002",
   "@type": "oa:Annotation"
  }
 },
 "createdAt": "2015-11-05T15:50:00Z"
}
```

Figure 5.17. A renarration transform specification in JSON-LD of a web page describing insertion of alternative content by using BetweenSelector.

```
{
 "@id": "3cd9b52d-89db-4ccc-8450-0baec2455f99",
 "@type": "rn:RenarrationTransformation",
 "action": "rn:Replace",
 "sourceSelection": {
  "@id": "77d9f63f-a806-4e06-8022-66ff4fc1ffed",
  "@type": "rn:XPathSelector",
  "value": "#xpointer(//html/body/div[2])"
 },
 "targetSelection": {
  "@id": "77d9f63f-a806-4e06-8022-66ff4fc1ffed",
  "@type": "rn:XPathSelector",
  "value": "#xpointer(//html/body/embed[1])"
 },
 "narration": {
  "@id": "https://prezi.com/embed/fxk9h96h0zyp/?bgcolor=ffffff...",
  "@type": "dctypes:InteractiveResource"
 },
 "createdAt": "2015-11-05T16:05:00Z"
}
```

Figure 5.18. A renarration transform specification in JSON-LD of a web page describing replacement of content with an interactive resource.

renarration is to renarrate the content so that Java programmers can consume the page. Another use case includes renarration of a simple text. In this scenario, the content in English, cannot be consumed by people who can't understand the language. The aim of the renarration is to renarrate the content so that it can be understood by Turkish speakers.

5.3. Use Case 1

In this use case, we have created a Web page in text/html format. The page is about method overriding in C++, an object oriented language. Content used to create pages and renarrations is originated in Wikipedia [55]. The purpose of this use case is to check whether the model and the prototype is capable of renarrate the content. The content of the page can be seen in Figure 5.20.

Figure 5.19. Web page renarrated by applying transformations on the FIR source page

List of renarration transforms applied to the source are as follows :

(i) Textual content "Function" is replaced with "Method". This renarration transforms aims to assert that "Method Overriding" is most suitable in Java Programming Language. Renarrtion Transform can be seen in Figure 5.21.

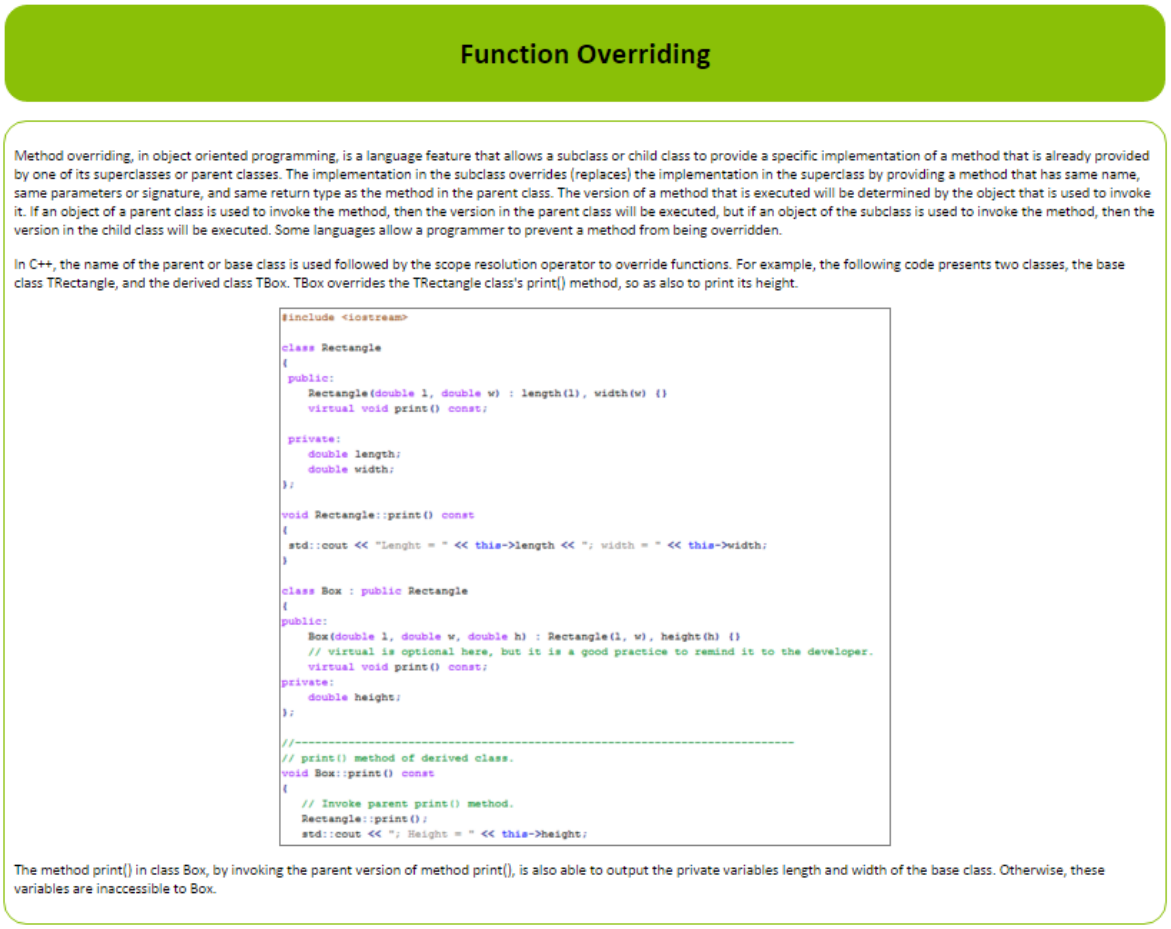

Figure 5.20. Web Page about overriding feature in C++ Programming Language.

(ii) Content which can be accessed via xpath(//html/body/div/div[2]/div[1]/p[2]) is replaced with alternative content. The original paragraph is about overriding a method in C++ programming language, and the paragraph explains subsequent image. This paragraph is renarrated and the renarration mainly talks about overriding methods of superclasses in Java programming language. Lines between 1 and 16 in Figure 5.22.

(iii) The image whose xpath is //html/body/div/div[2]/img, is replaced by alternative one. The alternative image gives an example override in Java. Lines between 17 and 31 in Figure 5.22.

After using the prototype for the renarrations, generated json-ld can be seen in Figure 5.21 and Figure 5.22. All renarration transforms are in the order they applied on the source content. Also, please note that the renarration is done using an application

server on a local machine.

```
{
 "@id": "095555b3-40f9-4120-bb67-2db9f4dc40b6",
 "@type": "rn:RenarrationTransformation",
 "sourceSelectionList": {
  "@id": "67d115be-ae56-470f-b a59-dc4b4c0290c9",
  "@type": "rn:List",
  "nodes": [
   {
    "@id": "23b6e472-3cbf-421 c-8100-59be8d021556",
    "@type": "rn:XPathSelector",
    "value": "#xpointer(//html[1]/body[1]/div[1]/div[1]/h1[1])"
   },
   {
    "@id": "5d40b6da-888b-4c91-8079-1f016b887 2f9",
    "@type": "rn:PrefixSuffixSelector",
    "prefix": "",
    "suffix": "",
    "text": "Function"
   }
   ]
  },
 "createdAt": "2015-11-19T11:48:26.719Z",
 "action": "oa:Replace",
 "narration": {
  "@id": "6322109a-e221-4c61-9e51-5d31287a4dc9",
  "@type": "cnt:ContentAsText",
  "content": "Method"
 }
}
```

Figure 5.21. A renarration transform specification (in JSON-LD) of textual content describing replacement of text for a web page about method overriding in C++ to one for the Java Programming Language

5.4. Use Case 2

In this use case, we have created a Web page in text/html format in which only a short message exists. The aim of this use case is also to show that the model can be used for renarration of short messages which are very common in social networking websites.

The short message in the Web resource is about a black guy, Jonathan Fleming,

```
{
 "@id": "04 17e211-2535-4168-8b53-b808f5fe1d89",
 "@type": "rn:RenarrationTransformation",
 "sourceSelection": {
  "@id": "c02b505e-b6f2-4c2a-b570-4b81472ebcb4",
  "@type": "rn:XPa thSelector",
  "value": "#xpointer(//html[1]/body[1]/div[1]/div[2]/div[1]/p[2])"
 },
 "createdAt": "2015-11-19T11:50:00.154Z",
 "action": "oa:Replace",
 "narration ": {
   "@id": "0f586d5a-ad05-4df7-a4c8-b8aa2980ad5a",
   "@type": "cnt:ContentAsText",
   "content": " In Java, when a subclass contains a method that overrides a method of
       the superclass, it can also invoke the superclass method by using the k eyword
       super. Example shows that Box class overrides the print() method of Rectangle
       class. Additionally, it prints height information."
 }
},
{
 "@id": "366d2697-6550-4e39-94fc-3d0c3b30da7b",
 "@type": "rn:RenarrationTransformation",
 "sourceSelection": {
  "@id": "8e113cc8-70b7-4a9b-9c3d-38d7dedde661",
  "@type": "rn:XPathSelector",
  "value": "#xpointer(//html[1]/body[1]/div[1]/div[2]/img[1])"
 },
 "createdAt": "2015-11-19T11:50:43.472Z",
 "action": "oa:Replace",
 "narration": {
  "@id": "http://localhost:8181/renarration/resources/java_override.png",
  "@type": "dctypes:Image",
 }
}
```

Figure 5.22. A renarration transform specification (in JSON-LD) describing replacement of a paragraph and an image of a web page about method overriding in C++ to create an alternative for the Java Programming Language

who spent 25 years in prison for murder he did not commit. Discovered evidence - a phone receipt - provided that he was vacationing with his family at Disney World in Bay Lake, Florida. [73] The text of the short message is as follows :

"Wrongly convicted black man has been freed after spending nearly 25 years behind bars."

The message doesn't say anything about the event; it does not even say who the person is. It is very common to see such messages on the Web. The following list of actions are assumed for the evaluation of the use case. The purpose of the renarration is to make the short message more meaningful for Turkish speakers. The text in the short message and the content in renarration transforms are referenced from morallowground.com [56] and www.commondreams.org [57].

(i) An annotator who knows a little English creates an annotation and asks whether someone to explain what the message is all about. Figure 5.26.
(ii) Another annotator creates an annotation for the textual content "black man" and the content of the annotation is "Jonathan Fleming" as in Figure 5.25.
(iii) Renarrator sees all the annotations and decides to create a renarration of the short message. The renarration includes one transformation which is an instance of rn:List. Line number between 16 and 43 in Figure 5.27.

The HTML code of short message in Figure 5.23 can be seen in Figure 5.24.

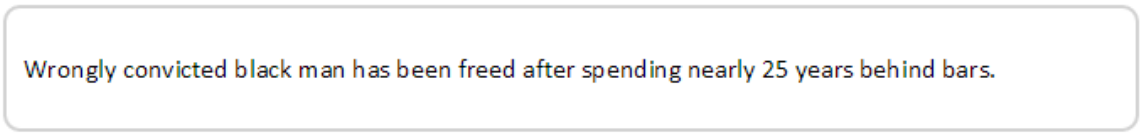

Figure 5.23. Short message about a black guy who spent nearly 25 years in prison.

A semantic annotation for the text "black man" within the short message can be seen in Figure 5.25. Another annotation questioning the event in Turkish can be seen in Figure 5.26.

Renarrator renarrates the short message and first summarizes what happened to Jonathan Fleming between lines 30 and 33 in Figure 5.27. After summarization, renarrator talks about evidences that might prevent him being put in prison. Lines between 34 and 38 in Figure 5.27.

As it can be seen in the renarration target, the page is deployed as jonathan_fleming

```
<?xml version="1.0" encoding="utf-8"?>
<html>
 <head>
  <title>Black man spent 25 years in prison</title>
 </head>
 <body>
  <div align="center">
   <br>
   <div style="width:650px;border-radius: 10px;border:2px solid lightgray;padding: 10
       px;height=30px;" align="center">
    <div align="left">
     <p align="left">
      Wrongly convicted black man has been freed after spending nearly 25 years behind
           bars.
     </p>
    </div>
   </div>
  </div>
 </body>
</html>
```

Figure 5.24. HTML code of a short message about a wrongful conviction

_turkish.html. Deployed page can be used for people who know Turkish, and when they visit the page in which the short message exists, they can be shown the renarrated page instead. The renarrated page can be seen in Figure 5.28.

5.5. Results and Discussion

The amount of data on the Web is massive and continually growing. Because of characteristics of the content, it is not accessible by huge amount of people due to a variety of barriers. The barriers to accessibility range from limited Internet connectivity, to physical impairment, linguistic differences, and also social, cultural and economic factors [6]. In order to create accessibility to the content, it is very important to create meaningful relations between Web elements and to provide accessibility.

Annotation is one of the methods to increase accessibility for Web content. The way that annotation method uses is to attach data to some piece of data. However, considering evaluations in this chapter in earlier sections, attaching some piece of data may be inefficient. For instance, for the evaluation created for how to create an FIR,

```
{
 "@type": "oa:Annotation",
 "annotatedAt": "2015-11-20T11:46:48.747Z",
 "serializedAt": "2015-11-20T11:46:48.747Z",
 "annotatedBy": {
   "@id": "749bf220-4e8b-481d-8b7a-e5e735e684f1",
   "@type": "foaf:Person",
   "name": "Person1"
 },
 "motivation": "oa:commenting",
 "body": {
  "@id": "d1f273c6-3f93-4e37-b93c-8b58d 01017ec",
  "@type": "foaf:Person",
  "foaf:firstName": "Jonathan",
  "foaf:lastName ": "Fleming"
 },
 "target": {
  "@id": "c25cf552-2bcf-40c8-8424-f46b77287595",
  "@type": "oa:SpecificResource",
  "selector": {
   "@id": "c2cca973-b418-484b-be6e-5 8afd271edfd",
   "@type": "oa:List",
   "members":
   [
    {
     "@id": "5294370a-f13f-4596-a 687-2c0868a6c28e",
     "@type": "rn:XPathSelector",
     "xpath": "#xpointer(//html[1]/body[1]/div[1]/div[1]/div[1]/p[1])"
    },
    {
     "@id": "7b0fa8bc-d7c4-4685-81e5-6a90a15d7241",
     "@type": "oa:TextQuoteSelector",
     "exact": "black man",
     "prefix": "",
     "suffix": ""
    }
   ]
  },
  "source": {
   "@id": "http://localhost:8181/renarration/resources/jonathan_fleming.html",
   "@type": "foaf:page"
  }
 }
}
```


Figure 5.25. Annotation in JSON-LD format created for textual content "black man"

```
{
 "_id": ObjectId("564f08b413c11b5d787b3074"),
 "@context":
 [
  "http://www.w3.org/ ns/oa-context -20130208.json",
  {"rn": "http://localhost:8181/renarration/ns/20151001.json"}
 ],
 "@id": "03b2f719-341a-4db6-b6e0-5700da0dd6d5",
 "@type": "oa:Annotation",
 "annotatedAt": "2015-11-20T11:49:05.620Z",
 "serialized At": "2015-11-20T11:49:05.620Z",
 "annotatedBy": {
  "@id": "e8f551f8-33ea-429d-8932-9f16bd8bd3b7",
  "@type": "foaf:Person",
  "name": "Person2"
 },
 "serializedBy ": {
  "@id": "http://localhost:8181/renarration/",
  "@type": "prov:SoftwareAgent",
  "name": "Code v1.1",
  "homepage": "http://localhost:8181/renarration/"
 },
 "motivation": "oa:questioning",
 "body": {
  "@id": "b780c41c-2edb-49db-a5e1-0630 62ef3689",
  "@type": "oa:EmbeddedContent",
  "value": "Tam olarak olayın nasıl olduğunu anlatabilir misiniz?",
  "format": "text/plain"
 },
 "target ": {
  "@id": "19ca990d-b233-4047-b40a-258a8278fada",
  "@type": "oa:SpecificResource",
  "selector": {
   "@id": "02f5474c-90b9-48c6-8e19-859347884945",
   "@type": "rn:XPathSelector",
   "xpath": "#xpointer(//html[1]/body[1]/div[1]/div[1])"
  },
  "source": {
   "@id": "http://localhost:8181/renarration/resources/jonathan_fleming.html",
   "@type": "foaf:page"
  }
 }
}
```


Figure 5.26. Annotation questioning the meaning of short message in Turkish

```
{ ...
 "transformList": {
  "@id": "a8d33924-cd22-42dc-b8aa-385bb728341c",
  "@type": "rn:List",
  "nodes": [
   {
    "@id": "e14c99ed-1d39-43f1-894c-25e162e4d749",
    "@type": "rn:RenarrationTransformation",
    "sourceSelection": {
     "@id": "fb60d831-7c18-4e83-b8da-341e46482106",
     "@type": "rn:XPathSelector",
     "value": "#xpointer(//html[1]/body[1]/div[1]/div[1]/div[1]/p[1])"
    },
    "createdAt": "2015-11-20T13:07:27.967Z",
    "action": "oa:Replace",
    "narrationList": {
     "@id": "e5a992f2-ac54-424b-af2a-4da794635bd1",
     "@type": "rn:List",
     "nodes": [
      {
       "@id": "58438a86-9c97-43b7-b82e-37dbb8417a5d",
       "@type": "cnt:ContentAsText",
       "content": "Jonathan Fleming ",
       "accessedFrom": {
        "@id": "88b64849-fbdb-4157-8393-08bb509bcd80",
        "@type": "oa:Annotation"
       }
      },
      {
       "@id": "7664abaf-3fe4-471c-b7f1-a5040eb8428c",
       "@type": "cnt:ContentAsText",
       "content": "1989 yilinda Broklyn'da uyusturucu saticisi Darryl Alston'i
           oldurmek sucundan muebbet hapse mahkum edildi. Uzun bir zamandan sonra
           saklanan kanitlar ortaya cikinca serbest birakildi. Fleming daima  olayin
           oldugu anda Florida'da tatilde oldugu uzerinde israr ediyordu..."
      },
      {
       "@id": "243b6a94-d16f-4c30-aff4-7800be645415",
       "@type": "cnt:ContentAsText",
       "content": " Fleming'in cinayeti islemedigine dair en guclu kanit Orlanda
           otelinde saat 21:27'ye ait odenmis bir telefon faturasinin bulunmasiydi.
           Cinayet ise sonraki gun sabaha karsi 02:15'de islendiginden cinayeti
           islemedigi acikca ortadaydi. Bu kanitin neden ilk durusmada ortaya
           cikarilmadigi ise buyuk bir muamma..."}]
... }
```

Figure 5.27. Renarration in json-ld format for creation of an alternative page in Turkish for the short message about Jonathan Fleming who wongfully put in prison for 25 years.

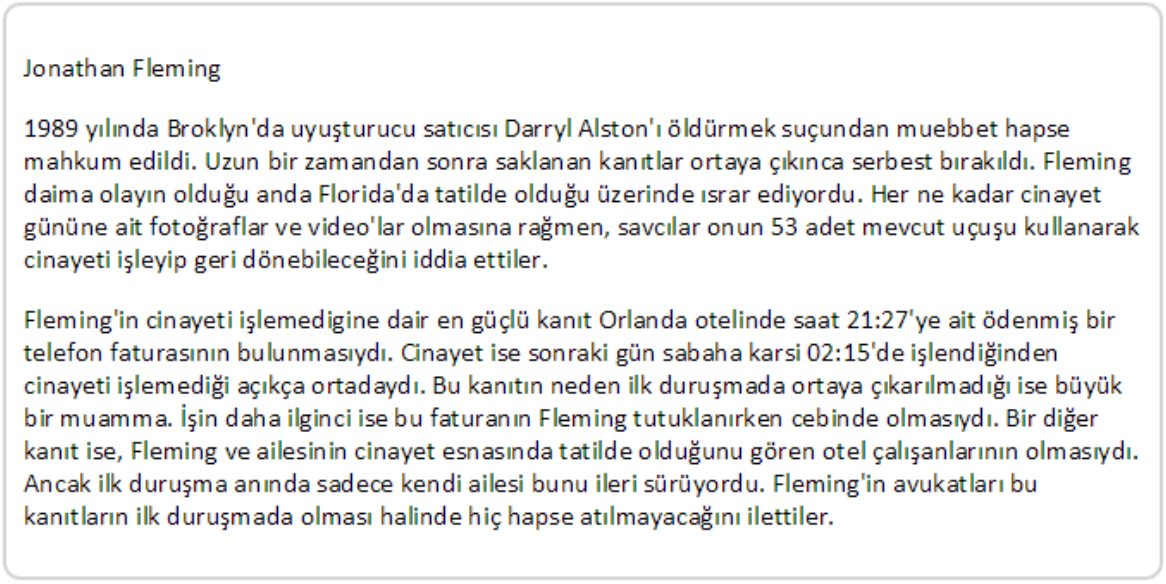

Figure 5.28. Deployed renarrated page of short message in HTML format.

attaching some data wouldn't make the content consumable for low literate consumers.

Renarration approach focuses on content and the renarrated content while creating semantic relationships. For example, a low literate consumer visiting FIR page can be redirected for the renarrated page which is targeted for low literate people. Alternatively, assuming there is a audio of the content for creating FIR report, a consumer doesn't know how to read might be directed to the audio instead.

Alipi prototype implements renarration as a service. Using the service, a user can choose a web page for renarration and specify target groups and publish the renarration. Any number of renarrations may be created for any given web page. The renarrator can define translations, simplifications or provide alternative media to the target audience. However, the approach doesn't use a predefined vocabulory, an ontology, to define relationships between content and the related content. Using proposed Renarration Data Model, the relationships can be defined.

Even though the prototype implemented doesn't cover some special transformations such as the ones that change xpath of elements, it shows that it can be used as a basis for renarrations of documents. The prototype also doesn't cover of removal of

elements which again may change the xpath hierarchy of documents. However, solution to this can be implemented by simple algorithms as one can be found in Appendix.

Experiments show that the data model provides a standardized method for creating renarrations of content. The data model scales from very simple to more complex use cases such as removal and replacement of web elements. The model also allows usage of external resources such as annotations and content in another web pages.

6. FUTURE WORK AND CONCLUSIONS

This thesis presents a framework which enables renarrating contents to make it available to a wider audience by creating semantic relations between URIs and contents of web pages.

In order to address the accessibility problem, an ontology framework model and prototype implementation which generates semantically processable data, have been developed. Web Annotation Data Model was chosen as the framework model for defining annotations. The Web Annotation Data Model provides an extensible framework for defining annotations so that they can easily be shared between platforms. The proposed approach supports working with this data model as well as other knowledge bases.

Evaluations show that our proposed model is capable of defining renarrations of web pages while keeping semantic relations between elements. It is observed that even though annotation is a very useful method for making contents more accessible and consumable, renarration provides more comprehensive framework. The main reason behind that the proposed data model keeps semantic relations between original content and the renarrated content. However, it is also observed that annotation provides a useful mechanism for linking external sources such as resources from DBpedia.

Proposed data model uses external ontologies for defining different types of content and narrations. The Dublin Core Schema is a small set of vocabulary terms which can be used to define web resources including images, video, web pages, etc. However, evaluations show that including links, tables in an ontology and using in renarration framework would definitely improve expressiveness of renarrations.

The potential benefits of annotation are demonstrated by the design and implementation of SemAnn (Semantic Annotation) framework. SemAnn provides a user interface for creating annotations of different types such as text, image, semantic tags,

etc. Advantages of annotations are demonstrated by the use of renarration implementation.

Semantic Renarration framework provides a user interface for creating renarrations. Evaluations on this framework shows that such user interfaces can easily be developed for creating renarrations of web resources and linking individual contents.

The evaluations show that the designed model and the implemented prototype would be a useful system to increase accessibility of content.

Throughout this study, we have learned a lot from the maturity of the Web Annotation Data Model even though the model was still in draft version. It was also a great opportunity to join weekly meetings for the data model. This showed us that even a small amount of work is concluded after enormous amount of discussions and meetings.

As future work, the proposed renarration approach could be improved in several directions.

- Annotations can be created using a smart text editor to allow different types of annotations. The current implementation doesn't cover annotation of web resources such as audio, video, pdf, etc. These types can be implemented by integration of open source implementations.
- SemRen can be improved so that it may cover removal of elements. Currently the framework just hides the element from the user. In addition to this, the implementation may support transformations that change xpath of elements between source and target resources.
- A Social Network can be formed in SemRen and SemAnn frameworks. Users can be represented based on their annotations and renarrations. Social network analysis could be performed using the tool and users can be related with each other.
- Renarration Data Model can be extended with a domain-specific ontology so that

it covers all necessary elements used in web pages.

- The proposed model can be extended so that it supports renarration of renarrations. In addition to that, it could be improved so that it supports versioning of renarrations.

REFERENCES

1. *Web Annotation Data Model*, `http://www.w3.org/TR/2014/WD-annotation-model-20141211`, accessed at February 2015.

2. *Google Search Engine*, `https://www.google.com.tr`, accessed at September 2014.

3. *Yahoo Search Engine*, `https://www.yahoo.com`, accessed at September 2014.

4. *Semantic Web - W3C*, `http://www.w3.org/standards/semanticweb/`, accessed at September 2014.

5. *"W3C Semantic Web Activity". World Wide Web Consortium (W3C).*, `http://www.w3.org/2001/sw/Activity`, .

6. Dinesh, T., S. Uskudarli, S. Sastry, D. Aggarwal and V. Choppella, "Alipi: A framework for re-narrating web pages", *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*, p. 22, ACM, 2012.

7. Oren, E., K. Möller, S. Scerri, S. Handschuh and M. Sintek, "What are semantic annotations", *Relatório técnico. DERI Galway*, 2006.

8. *Extensible Markup Language - XML*, `http://www.w3.org/XML/`, accessed at September 2014.

9. Sanderson, R., P. Ciccarese, H. Van de Sompel, S. Bradshaw, D. Brickley, L. J. G. Castro, T. Clark, T. Cole, P. Desenne, A. Gerber *et al.*, "Open annotation data model", *W3C Community Draft*, 2013.

10. Sanderson, R., P. Ciccarese and H. Van de Sompel, "Designing the W3C open annotation data model", *Proceedings of the 5th Annual ACM Web Science Conference*, pp. 366–375, ACM, 2013.

11. Antoniou, G. and F. Van Harmelen, *A semantic web primer*, MIT press, 2004.

12. *Well-formed element - Wikipedia, the free encyclopedia*, https://en.wikipedia.org/wiki/Well-formed\textunderscoreelement, accessed at September 2014.

13. *HTML 4 Document Type Definition*, http://www.w3.org/TR/html4/sgml/dtd.html, accessed at December 2014.

14. *RDF - Semantic Web Standards*, http://www.w3.org/RDF/, accessed at December 2014.

15. *RDF Schema 1.1*, http://www.w3.org/TR/rdf-schema/, accessed at December 2014.

16. *OWL Web Ontology Language*, http://www.w3.org/TR/owl-features/, accessed at December 2014.

17. *SPARQL Query Language for RDF*, http://www.w3.org/TR/rdf-sparql-query/, accessed at December 2014.

18. *JSON-LD 1.0 Specification*, http://www.w3.org/TR/json-ld-syntax/, accessed at September 2014.

19. *Schema.org Specification*, http://schema.org/, accessed at September 2014.

20. *Dublin Core Elements*, http://purl.org/dc/elements/1.1/, accessed at December 2014.

21. *Dublin Core Terms*, http://purl.org/dc/terms/, accessed at December 2014.

22. *DCMI Metadata Terms*, http://purl.org/dc/dcmitype/, accessed at December 2014.

23. *FOAF Vocabulary Specification*, http://xmlns.com/foaf/spec/, accessed at De-

cember 2014.

24. *Provenance Ontology*, `http://www.w3.org/ns/prov\#`, accessed at December 2014.

25. *RDF 1.1 XML Syntax*, `http://www.w3.org/TR/rdf-syntax-grammar/`, accessed at December 2014.

26. *SKOS Simple Knowledge Organization System Reference*, `http://www.w3.org/TR/skos-reference/`, accessed at December 2014.

27. Liakata, M., L. N. Soldatova *et al.*, "Semantic annotation of papers: Interface & enrichment tool (sapient)", *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, pp. 193–200, Association for Computational Linguistics, 2009.

28. Michelson, M. and C. A. Knoblock, "Semantic annotation of unstructured and ungrammatical text", *International Joint Conference on Artifical Intelligence*, Vol. 19, p. 1091, Lawrence Erlbaum Associates Ltd, 2005.

29. Ciccarese, P., M. Ocana, L. J. Garcia-Castro, S. Das and T. Clark, "An open annotation ontology for science on web 3.0.", *J. Biomedical Semantics*, Vol. 2, No. S-2, p. S4, 2011.

30. Lausen, H. and J. Farrell, "Semantic annotations for WSDL and XML schema", *W3C recommendation, W3C*, 2007.

31. Narr, S., E. W. De Luca and S. Albayrak, "Extracting semantic annotations from twitter", *Proceedings of the fourth workshop on Exploiting semantic annotations in information retrieval*, pp. 15–16, ACM, 2011.

32. Momeni, E., "Semi-automatic semantic moderation of web annotations", *Proceedings of the 21st international conference companion on World Wide Web*, pp. 167–172, ACM, 2012.

33. Bloehdorn, S., K. Petridis, C. Saathoff, N. Simou, V. Tzouvaras, Y. Avrithis, S. Handschuh, Y. Kompatsiaris, S. Staab and M. G. Strintzis, "Semantic annotation of images and videos for multimedia analysis", *The semantic web: research and applications*, pp. 592–607, Springer, 2005.

34. *Django Girls website*, `https://djangogirls.org/`, accessed at October 2015.

35. *Accessibility W3C*, `http://www.w3.org/standards/webdesign/accessibility`, accessed at December 2014.

36. *Turkiye Istatistik Kurumu*, `http://www.tuik.gov.tr/`, accessed at December 2014.

37. *Duolingo*, `https://www.duolingo.com/`, accessed at September 2014.

38. *Google Translate*, `https://translate.google.com`, accessed at September 2014.

39. *Annotator - Annotating The Web*, `http://annotatorjs.org/`, accessed at January 2015.

40. *Daniel Cebrián Robles*, `http://danielcebrian.com/?lang=en`, accessed at January 2015.

41. *Daring Fireball: Markdown*, `http://daringfireball.net/projects/markdown/`, accessed at January 2015.

42. *XML Schema*, `http://www.w3.org/2001/XMLSchema#`, accessed at December 2014.

43. *Representing Content in RDF 1.0*, `http://www.w3.org/TR/Content-in-RDF/`, accessed at December 2014.

44. *Spring Framework*, `https://spring.io/`, accessed at December 2014.

45. *JavaScript Programming Language*, https://developer.mozilla.org/en-US/docs/Web/JavaScript, accessed at December 2014.

46. *W3C XHTML*, http://www.w3.org/MarkUp/, accessed at December 2014.

47. *Jsoup Java HTML Parser*, http://jsoup.org/, accessed at December 2014.

48. *Jena Semantic Web Framework website*, https://jena.apache.org/, accessed at December 2014.

49. *MongoDB website*, https://www.mongodb.org/, accessed at December 2014.

50. *DBPedia website*, http://wiki.dbpedia.org/, accessed at December 2014.

51. *GitHub website*, https://github.com/, accessed at December 2014.

52. *First Information Report*, https://en.wikipedia.org/wiki/First_Information_Report, accessed at September 2015.

53. *Prezi website*, https://prezi.com/fxk9h96h0zyp/first-information-report/, accessed at September 2015.

54. *Cognisable offence - Wikipedia, the free encyclopedia*, https://en.wikipedia.org/wiki/Cognisable_offence, accessed at September 2015.

55. *Method overriding - Wikipedia, the free encyclopedia*, https://en.wikipedia.org/wiki/Method_overriding, accessed at September 2015.

56. *Moral Low Ground website*, http://morallowground.com/2014/04/10/, accessed at September 2015.

57. *After Nearly 25 Years Behind Bars, Wrongfully Convicted Man Free - Common Dreams website*, http://www.commondreams.org/news/2014/04/08/after-nearly-25-years-behind-bars-wrongfully-convicted-man-free, ac-

cessed at September 2015.

APPENDIX A: JSON-LD Context And Implemented JavaScript For Prototype

A.1. JSON-LD Context

This section provides recommended JSON-LD context recommended for the Renarration Data Model. The usage of the context is recommended because it prevents developers to deal with complex json template as well as enrusing consistency.

Listing A.1 shows the JSON-LD context.

Listing A.1. JSON-LD context recommended for the Renarration Data Model

```
{
 "@context": {
  "rn": "https://github.com/EmrahGuder/Renarration/#",
  "oa": "http://www.w3.org/ns/oa#",
  "foaf": "http://xmlns.com/foaf/0.1/",
  "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
  "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
  "cnt": "http://www.w3.org/2011/content#",
  "skos": "http://www.w3.org/2004/02/skos/core#",
  "dc": "http://purl.org/dc/elements/1.1/",
  "dctypes": "http://purl.org/dc/dcmitype/",
  "owl": "http://www.w3.org/2002/07/owl#",
  "xsd": "http://www.w3.org/2001/XMLSchema#",

  "sourceSelection" : {"@type":"@id", "@id" : "rn:sourceSelection"},
  "sourceSelectionList" : {"@type":"@id", "@id" : "rn:sourceSelection","@container" :
      "@list"},
  "targetSelection" : {"@type":"@id", "@id" : "rn:targetSelection"},
  "targetSelectionList" : {"@type":"@id", "@id" : "rn:targetSelection","@container" :
      "@list"},
  "audience"    : {"@type":"@id", "@id" : "rn:hasTargetAudience"},
  "source" : {"@type":"@id", "@id" : "rn:onSourceDocument"},
  "target" : {"@type":"@id", "@id" : "rn:onTargetDocument"},
  "transformList" : {"@type":"@id", "@id" : "rn:hasRenarrationTransformation","
      @container" : "@list"},
  "transform" : {"@type":"@id", "@id" : "rn:hasRenarrationTransformation"},
  "nodes" : {"@type":"@id", "@id" : "rn:nodes", "@container" : "@list"},
  "action" : {"@type":"@id", "@id" : "rn:actionOnDocument"},
  "between" : {"@type":"@id", "@id" : "rn:isBetween","@container" : "@list"},
  "narration" : {"@type":"@id", "@id" : "rn:hasNarration"},
  "narrationList" : {"@type":"@id", "@id" : "rn:hasNarration","@container" : "@list"},
```

```
  "accessedFrom" : {"@type":"@id", "@id" : "rn:accessedFrom"},
  "renarrator" : {"@type":"@id", "@id" : "rn:renarratedBy"},
  "motivation" : {"@type":"@id", "@id" : "rn:hasMotivation"},


  "content" : "cnt:content",
  "renarratedAt" : "rn:renarratedAt",
  "value" : "rdf:value",
  "createdAt" : "rn:createdAt",
  "prefix": "rn:prefix",
  "language" : "dc:language",
  "suffix": "rn:suffix",
  "text": "rn:selectedText",
  "name": "foaf:name",
  "mbox" : "foaf:mbox",
  "offset": { "@type": "xsd:nonNegativeInteger", "@id": "rn:offset" },
  "byteNumber": { "@type": "xsd:nonNegativeInteger", "@id": "rn:byteNumber" }
 }
}
```

A.2. Injected JavaScript Used For Prototype

This section provides JavaScript code which is used in the implemented prototype for enabling web resources annotatable and renarratable. The script is injected into the web resource, and defines three listeners for mouse events.

Listing A.2. Injected JavaScript for adding event listeners to web resources

```
// variables used
var mouseOverElement, mouseOverElementStyleBorderColor, mouseOverElementStyleBorder;
var mouseOutElement;
var clickedElement;

// Window Event Listeners
window.addEventListener("mousedown", bodyMouseDown, false);
window.addEventListener("mouseover", bodyMouseOver, false);
window.addEventListener("mouseout", bodyMouseOut, false);
// Window Event Listeners

// Event Listener Functions
function bodyMouseOver(event){
 mouseOverElement = event.target;
 if(getPathTo(mouseOverElement).toLowerCase()!='//html[1]/body[1]'){
  mouseOverElementStyleBorderColor = mouseOverElement.style.borderColor;
  mouseOverElementStyleBorder = mouseOverElement.style.border;
  mouseOverElement.style.border ='2px dashed red';
 }
```

```
 window.parent.Parent_mouseOverElement = mouseOverElement;
 window.parent.Parent_mouseOverElement_XPath = getPathTo(mouseOverElement).toLowerCase
      ();
}

function bodyMouseOut(event){
 mouseOutElement = event.target;
 if(window.parent.type=="annotation"){
  if(window.parent.findAnnotationUsingXPath(getPathTo(mouseOutElement).toLowerCase())
       >=0){
   if(getPathTo(mouseOutElement).toLowerCase()!='//html[1]/body[1]'){
    mouseOutElement.style.borderColor = 'yellow';
    mouseOutElement.style.border ='2px solid yellow';
   }
  }
  else{
   if(getPathTo(mouseOutElement).toLowerCase()!='//html[1]/body[1]'){
    mouseOutElement.style.borderColor = mouseOverElementStyleBorderColor;
    mouseOutElement.style.border = mouseOverElementStyleBorder;
   }
  }
 }
 else{
  if(getPathTo(mouseOutElement).toLowerCase()!='//html[1]/body[1]'){
   mouseOutElement.style.borderColor = mouseOverElementStyleBorderColor;
   mouseOutElement.style.border = mouseOverElementStyleBorder;
  }
 }
 //window.parent.hideAnnotationDivOnElement();
 //alert(currentElement.innerHTML);
}

function bodyMouseDown(event){
 clickedElement = event.target;
 window.parent.Parent_mouseOverElement = clickedElement;
 var x = new Number();
 var y = new Number();
 if (event.x != undefined && event.y != undefined)
 {
  x = event.x;
  y = event.y;
 }
 else // Firefox method to get the position
 {
  x = event.clientX + document.body.scrollLeft + document.documentElement.scrollLeft;
  y = event.clientY + document.body.scrollTop + document.documentElement.scrollTop;
 }

 updateParentDiv();
}
```

```
// Event Listener Functions

// Library Functions
function updateElementContent(element){
 element.innerHTML = 'This is a test';
}

function getPathTo(element) {
 if (element===document.body)
  return '//html[1]/' + element.tagName + '[1]';

 var ix= 0;
 var siblings= element.parentNode.childNodes;
 for (var i= 0; i<siblings.length; i++) {
  var sibling= siblings[i];
  if (sibling===element)
   return getPathTo(element.parentNode)+'/'+element.tagName+'['+(ix+1)+']';
  if (sibling.nodeType===1 && sibling.tagName===element.tagName)
   ix++;
 }
}

function getElementByXpath(path){
 return document.evaluate(path, document, null, XPathResult.FIRST_ORDERED_NODE_TYPE,
     null).singleNodeValue;
}

function updateParentDiv(){
 var divElement = window.parent.document.getElementById('ReNarrationDiv');

 divElement.style.display = 'block';
 // set parent element
 window.parent.Parent_selectedElement = clickedElement;
 window.parent.Parent_selectedElement_XPath = getPathTo(clickedElement).toLowerCase();
 window.parent.setContentInDiv();
}

function highlightElementsWithAnnotations(xpath){
 elem = getElementByXpath(xpath);
 elem.style.borderColor = 'yellow';
 elem.style.border ='2px solid yellow';
}
```

A.3. Algorithm For Handling XPath Change

This section provides an algorithm for handling XPath changes for transforms that can change the indices of HTML elements. Even though this is not implemented

in the prototype we have done some evaluation about how this can be achieved. Figure A.3 shows a recommended algorithm.

```
let WR be a web resource to be renarrated
create a local copy of web resource WR'
for each HTML element in WR' do
    find XPath of current element
    create a dummy attribute
    set value of dummy attribute to XPath of current element
end for
return WR'
```

Figure A.1. Pseudo code for handling XPath changes of HTML elements