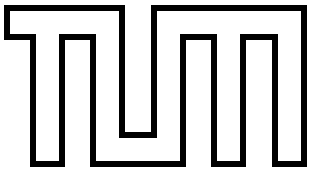


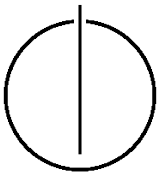
FAKULTÄT FÜR INFORMATIK

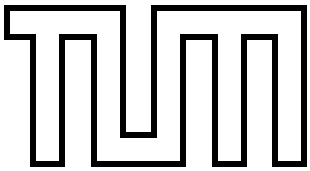
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Masters Thesis in Information Systems

Using Web Annotations to Represent Relations between Structured and Unstructured Information in Semantic Wikis

Shivguru Rao BhimasenaRao VisweswaraRao





FAKULTÄT FÜR INFORMATIK

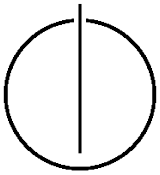
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Masters Thesis in Information Systems

Using Web Annotations to Represent Relations between Structured and Unstructured Information in Semantic Wikis

Nutzung von Web Annotations zur Reprsentation von Relationen zwischen strukturierten und unstrukturierten Informationen in Semantic Wikis

Author:	Shivguru Rao BhimasenaRao VisweswaraRao
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Daniel Braun
Date:	July 15, 2018



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, July 15, 2018 Shivguru Rao BhimasenaRao VisweswaraRao

Abstract

A semantic wiki is a wiki that has an underlying model of the knowledge described in its pages. The knowledge model of a semantic wiki is typically available as an entity model. The wiki we are considering in this thesis is hybrid wiki. Hybrid wikis combine unstructured information in the form of wiki texts with structured information. Structured information is usually represented in the form of key-value pairs. However, there is no link between the structured and the unstructured information. Therefore, if a piece of information is represented as both, structured and unstructured information, there is no way to know if the information is consistent, unless checked manually.

Aim of this thesis is to check for any existing standards to represent links, provide a way to create the links between structured and unstructured information and check for consistency of the created links over time. As data source SocioCortex is used. SocioCortex is an information system to organize semi-structured data within Enterprise Architecture Management, employing a dynamic and collaborative Wiki-based approach. Thus data is typically in the form of structured and unstructured information represented as Wiki-pages.

The approach of this thesis is to review the existing standards to model and represent web annotations like web annotation data model and JSON-LDs. Extending these models to represent the link between structured and unstructured information. Further, explore different ways to store the links so that it can be persisted and accessed across multiple tools and environments. Finally, implementing a prototype demonstrate creation of links, detecting changes to data which are linked and embedding links inside the HTML.

Contents

List of Figures

Listings

Outline of the Thesis

CHAPTER 1: INTRODUCTION

This chapter presents an overview of the thesis and it purpose. Research questions about the thesis is discussed in detail.

CHAPTER 2: FOUNDATION

This chapter presents foundation about the concepts used in the thesis. It provides deeper insights into the concepts of Hybrid Wikis and JSON-LDs

CHAPTER 3: RELATED WORK

This chapter presents the related work done relating to the topics of this thesis.

CHAPTER 4: EXTENDED STANDARD - JSON-LD FOR LINK CREATION

This chapter sets the foundation on how to extend the JSON-LD standard defined in Chapter 2 for the purpose of annotating and linking structured and unstructured data. It also provide insights into the link creation process.

CHAPTER 5: IMPLEMENTATION

This chapter presents the architecture and finer details about the implemented prototype. It also contains the screen shots of the final prototype.

CHAPTER 6: CONCLUSION

This chapter presents the conclusion and future work of the thesis. It also provides a recap of the research questions discussed in the Chapter 1.

1. Introduction

Wikis contain information above a specific topic or a group of related topics. This information is represented as free text, in the form of sentences and paragraphs. Some types of information demands more structured and concrete representation along with an elaborated unstructured representation. In Section 1.1 the motivation to link related content in the two above mentioned representations is provided. Further analysis of the problem, enables us to answer some research questions about creation and storage of links in Section 1.2. In Section 1.3, the solution space is discussed, where a standard is extended and implemented as a prototype based on SocioCortex system.

1.1. Motivation

As wikis represent more and more information about specific topics, the need to search for alternate representation of information arises. Information represented only in a structured layout, tabular format, cannot be used to convey the complete essence of the topic as it is used to summarize the topic rather than explaining it in detail. On the other hand, only unstructured information may make the reader miss some of the essential information. Hence, the hybrid approach where structured and unstructured information is combined, will provide a much better representation. As the information represented in both structured and unstructured format is essentially similar, there comes the need to establish link between related data but represented differently to maintain consistency and easily detect changes to the information.

A semantic wiki is a wiki with an underlying knowledge model to describe its pages. Regular (syntactic) wikis consists of structured text along with hyperlinks. On the other hand, semantic wikis captures and identifies information about data within pages and the relationships between pages, which can be queried or exported like a database through semantic queries.[12]

The knowledge model found in a semantic wiki is typically available in a formal language, so that machines can process it into an entity-relationship model or relational database. The formal notation may be included in the pages themselves, or it may be derived from the pages or the page names or the means of linking. Usually semantic wikis are heavy weight and has a fixed ontology, almost always.

Hybrid wikis has a lightweight semantic, comprising of a traditional wiki as a core component. In contrast to the heavyweight approaches like semantic wikis, the hybrid wiki does not annotate wiki content with semantic information with a fixed ontology, which is regarded as a top-down (i.e., ontology first) approach. In contrast, it allows a data-model to emerge from the content by enabling users to easily add structured content to any wiki page in the form of attribute-value tags. This bottom-up approach is complemented with the ability to establish certain constraints such as mandatory attributes or inheritance rela-

tionships between types.[8]

Hybrid Wikis are usually divided in to unstructured data and structured data. Unstructured data (or unstructured information) is information that either does not have a predefined data model or is not organized in a predefined manner. Unstructured information consists of free text in the form of paragraphs and can contain data such as dates, numbers, and facts along with text-rich content. This results in irregularities and ambiguities that make it difficult to understand using traditional programs as compared to data stored in fielded form in databases or annotated (semantically tagged) in documents.[17] Thus, some important information is also represented in a structured format. For the most part, structured data refers to information with a high degree of organization, such that inclusion in a relational database is seamless and readily search-able by simple, straightforward search engine algorithms or other search operations. Structured information in hybrid wikis takes the form of a predefined attribute-values schema.

Most of the time, same information is represented in both structured and unstructured format, in other words, same data different representation. Though, data is related there is no link established between them. It forces us to manually monitor the changes to related data. Hence, checking for data consistency is a tedious and manual process, more error prone. Many times, the relationship between the data is lost and may loose the overall meaning.

When a relationship is established between structured and unstructured information, consistency checks becomes a lot more easier. Easiest way to establish this relationship is through creation of links which contains references to structured attributes and parts of unstructured data.

1.2. Research questions

Once the motivation to create links between structured and unstructured information is established, below mentioned research questions needs to be answered. A link is a concept which can take different meanings. In general context, link is a relationship established between two things or situations, especially where one is related to the other. In the web context, link or hyperlink is a reference to data that the reader can directly follow either by clicking, tapping, or hovering. A hyperlink is a pointer pointing to an entire document or to a specific element within a document.[24]

In the context of linking structured and unstructured information, link is an additional information or meta-data that helps establishing relationship between relevant parts of unstructured information and structured attribute values.

1.2.1. What should be the format of the link?

Once we have established the need for creating links, the first question that comes to our mind is, in what format should link be represented. The format of the link should be extensible and reusable as it is used in the web context. Links are additional information added on top of the wiki document which does not affect the existing structure of the document. In the web context, idea behind annotation is adding more information about

selected text without changing the structure of the web document. Exploring the idea behind web annotations and enhancing it to represent a link could be a feasible solution.

1.2.2. Are there any annotation standards to link data?

Web annotations are represented in the form of JSON objects. Structure of JSON objects may vary from one implementation to another, depending on the type of data being annotated and purpose of annotation. Different annotation tools have different ways of structuring these JSON objects. In order, to understand the structure of these JSON objects developers has to rely on the documentation of implementer. Here comes the need for a serialization format which is easy to understand and self explanatory for the developers who are trying to understand the structure of a JSON object.

JSON for linked data or JSON-LD is one such serialization format to represent linked data. allows data to be serialized in a way that is similar to traditional JSON. JSON-LD is designed to allow existing JSON to be interpreted as Linked Data without much changes.

JSON-LD satisfies following design goals[10]:

1. **Simplicity:** No extra processors or software libraries are required to understand and use JSON-LD. The language provides developers with a very easy learning curve, where developers only needs to know about JSON and context meta data provided as the attribute *"@context"*. Simply speaking, a context is used to map terms to internationalized resource identifiers, which provides additional details on the data that is represented by a specific attribute.

2. **Compatibility:** A JSON-LD document should always be a valid JSON document. It ensures that standard JSON libraries can seamlessly with JSON-LD documents.

3. **Expressiveness:** The syntax serializes directed graphs. It ensures that almost any type of data can be expressed.

1.2.3. How can we store and retrieve links?

Links has to be stored in certain format for the purpose of persistence and retrieval. Hence, storing and retrieving links needs to be easy and straight forward.

There are many storage and retrieval options to be considered:

1. Save links in a dedicated database. The retrieval of links should be done explicitly by the software systems, which are designed to handled these links, when the HTML document of the wiki is retrieved.

2. Embed links as part of the wiki document. If links are part of the wiki, retrieval is implicitly taken care of when the wiki is retrieved. JSONs can be embedded as part of HTML document using script type as *"application/json"*. Though this is an effective way to store links without effecting the structure of the HTML document, it might raise some security concerns as additional information is added as part of original HTML document and software systems which deal with such documents might flag it as a security threat.

1.2.4. How to detect and handle the changes?

Software systems capable of creating and handling links should also be able to detect changes to the data to which it links. The underlying data changes can have adverse effects on links as it can either completely invalidate links or loose the relationship between the structured and unstructured data.

Detecting changes fairly straight forward, if exact state of the data linked at the time of link creation is stored and compared with the current state of data. On the other hand handling the changes is no that straightforward, as it is impossible to detect the reason behind the data changes and the intention of the person responsible for the data changes.

1.3. Purpose

The purpose of this master thesis is as follows:

- Answer the research questions listed in Section. 1.2.
- Create a web based software system capable of following:
  - Representing hybrid wikis with structured and unstructured data
  - Manually create links between structured and unstructured data. Links can contain additional comments about what it represents. Inter wiki links where unstructured data of one wiki can be linked with structured data on another wiki, also called as cross wiki document links.
  - Store and retrieve links along with the wiki, either as part of the wiki itself or separately in a dedicated database
  - Detect and handle the changes to the links. Changes should be highlighted in a way it is easy for the users to take well informed decision as to weather to delete and recreate the link or maintain the current state of the link.
- Creating and handling links should not change the current representation of wiki.
- Use or extend an existing standard for linking. Doing so facilitates re-usability, which enables other software systems which deals with the same wikis to handle the links.

1.4. Contribution

Once, purpose of the master thesis is established and clearly understood, its time to dig deeper into the existing standards that allows the creation of links and also existing Hybrid wiki systems which can be used as a source of wiki. Understanding the currently existing Hybrid wiki systems will provide further insights in to the structure of hybrid wiki and the relationship between structured and unstructured data of a wiki document.

1.4.1. Comparison of existing standards

Current existing standards for creating and representing annotations are compared and checked for the feasibility of using them to model the links. A way to combine the web annotations with the JSON-LD standard should be established.

1.4.2. Extension of standard

JSON-LD is a generic format to represent a network of standards-based machine interpreted data across different documents and web sites. For the purpose of representing links in hybrid wikis, JSON-LD standard has to be extended, where representation of structured and unstructured data is facilitated. In a standard hybrid wiki like SocioCortex, both structured and unstructured data of a wiki page is part of same document. Linking parts of same wiki document can be done by recognizing specific parts of wiki documents that are linked and visualizing them as pseudo links.

1.4.3. Implementation of Prototype

Prototype is a web based software system capable of following:

- Representing hybrid wikis with structured and unstructured data
- Manually create links between structured and unstructured data. Links can contain additional comments about what it represents. Inter wiki links where unstructured data of one wiki can be linked with structured data on another wiki, also called as cross wiki document links.
- Store and retrieve links along with the wiki, either as part of the wiki itself or separately in a dedicated database
- Detect and handle the changes to the links. Changes should be highlighted in a way it is easy for the users to take well informed decision as to weather to delete and recreate the link or maintain the current state of the link.

2. Foundations

The Hybrid Wiki concept, SocioCortex platform and JSON-LD standard is integral to this master thesis. They provide foundation to model structured and unstructured information along with methods to conceptualize links. The SocioCortex system will play a crucial role in this thesis, as it not only provides the information for creating links, but it also provides a perfect environment to store created links. JSON for linked data specification provides foundation to represent the links in hybrid wikis, these links are structured on the concepts defined as a part of JSON-LD specification.

2.1. Hybrid-Wikis and Meta Model

Wiki-platforms are the side effects of the evolution of new web 2.0 paradigm. Underlying principle behind wiki-platforms is collective intelligence.[22] Basically wikis are content management systems for creating and editing content. Wikipedia, which has more than 5.2 million articles, in the English version, is the most famous instance of the wiki available on the internet.

As these collaborative wiki systems find growing interest in enterprises, the question arises, how to align the data with corresponding model. In the top-down approach data is generated from the related model (model-first). In the bottom-up approach data is generated first, then model is aligned with the data (data-first). More details on these approaches are provided by Reschenhofer et al.[23] A more collaborative approach is suggested in "Hybrid-Wikis: Empowering Users to Collaboratively Structure Information" by Matthes et al. This "Hybrid-Wiki" approach supports the evolution of the model and its data in a coherent and consistent manner.[21] It is called a hybrid wiki because it combines the unstructured data with structured information.

The main goal of such hybrid wiki is to reduce the learning curve for non-expert users. Without knowing much about the background, users should be able to enter structured information. To solve this problem these contents are enriched with simple keyword-like annotations by experienced users or experts. the user implicitly creates semantic, by filling data into fields, or create new fields in forms.[21] The corresponding data model of a Hybrid-Wiki approach is depicted in Figure 2.1. The concept is developed by the chair of Software Engineering for Business Information Systems (SEBIS) of the Technical University of Munich (TUM). It is based on the modeling framework of a former tool called Tricia.[21]

Structure data of a wiki page can contain multiple attributes, while each attribute can have a list of values. A *Value* is an abstract type, concrete *Values* might be *StringValues* or *LinkValues*. Also a structured data can have multiples tags assigned to it. On the left side of the diagram, the *Wikipage, TypeTag, Attribute* and *Value* are provided for structuring the data in Hybrid-Wikis. On the right side *TypeTagDefinition, AttributeDefinition* and *Validator*

are used to specify integrity constraints. So the creation of a *TypeTagDefinition* specifies the values an user is urged to enter. While *TypeTagDefinition* and *TypeTag* and *Attribute* and *AttributeDefintion* are loosely coupled, users only receive suggestions for new attributes or tags. Thus new attributes can be created top-down.

Integrity constraints defined in *AttributeDefinition* can be specified using *Validators*. For example, a *MultiplicityValidator* specifies how many values an attribute can have (at-least-on, at-most-one, exactly-one, etc.). *Validators* are apparent by showing feedback messages for violated integrity constraints. They are called "soft validators", while they only warn users, but do not forbid the user to break them.[21]

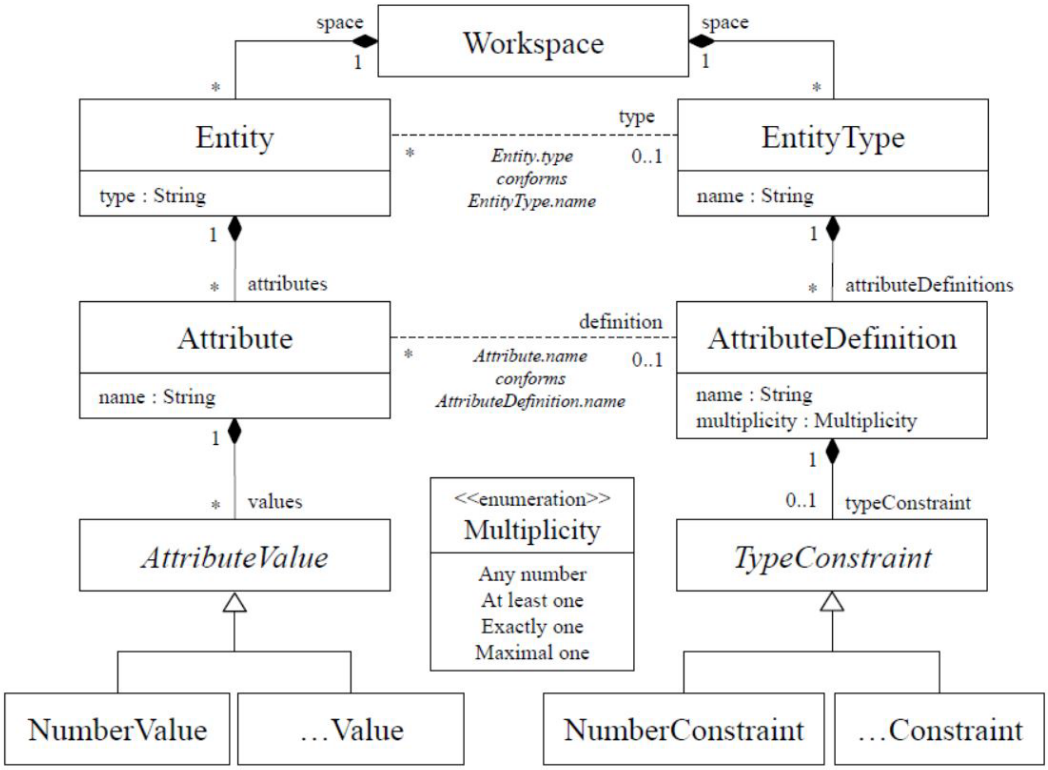


Figure 2.1.: HybridWiki Meta Model[23]

In this hybrid-wiki approach, the terms *Wiki* and *Pages* corresponds to *Workspace* and *Entity* respectively. A *Workspace* is a collection of related *Entities*. Each *Entity* has an associated *EntityType*. *EntityType* provides structure to the *Enitity*, which is a collection of *AttributeDefinition*. These above mentioned concepts are used throughout the thesis.

2.2. Web Annotation Data Model

A web annotation is meta data associated with a web resource, like a web page. Using an annotation system, a user can add, modify or remove information from a web resource without modifying the resource itself.[20] Annotations are typically used to convey infor-

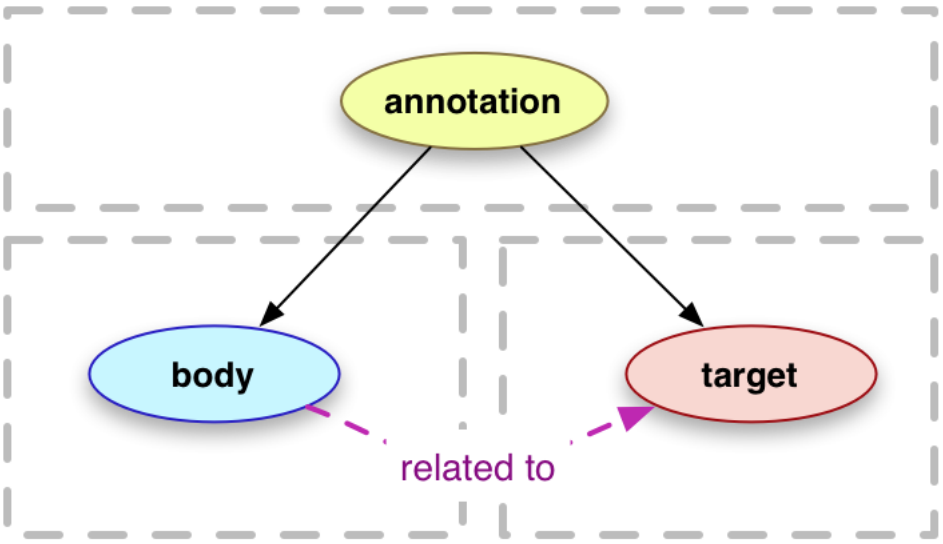


Figure 2.2.: Web Annotation[18]

mation about a resource or associations between resources. Simple examples include a comment or tag on a single web page or image, or a blog post about a news article.

The web annotation data model specification describes a uniform structured model and format so the annotations can be shared and reused across different hardware and software platforms. Some of the common use-cases are modeled in a way that is simple and convenient, at the same time enabling more complex requirements, including linking arbitrary content to a particular data point or to segments of timed multimedia resources.

The specification provides a specific JSON format for easy creation and consumption of annotations, which is based on the conceptual model that accommodates these use-cases and the vocabulary of terms that represents it.[18]

2.3. JSON for Linked Data

JavaScript Object Notation (JSON) is a data serialization, messaging and data exchange format. JSON for Linked Data (JSON-LD) is a lightweight syntax to serialize linked data in JSON. It is designed to interpret JSON as linked data with minimal changes. The primary intention behind JSON-LD is used to link data in web-based programming environments and to store Linked Data in JSON-based storage engines.

In addition to all the features JSON provides, JSON-LD introduces:

- a universal identifier for JSON-objects with the help of Internationalized Resource Identifier(IRIs).
- a way to differentiate keys shared among different JSON documents via a *context*. A context provides a way to map terms to IRIs. It is meta-data, that provides information about the data represented in the JSON-document.

- a mechanism where a JSON-object can reference another JSON-object on different web resource
- a way to annotate strings with other language
- a facility to represent one or more directed graphs, like a social network, in a single document

The following JSON-LD, represents data about *Person*. *Context* provides mapping between the attribute terms and the schema behind those terms. The attribute values need not be only an IRI, it can also contain simple string like name of a person. IRIs are a fundamental concept of Linked Data, for nodes to be truly linked, de-referencing the identifier should result in a representation of that node.

```
{
  "@context":
  {
    "name": "http://schema.org/name",
    "image": {
      "@id": "http://schema.org/image",
      "@type": "@id"
    },
    "homepage": {
      "@id": "http://schema.org/url",
      "@type": "@id"
    }
  },
  "name": "Manu Sporny",
  "homepage": "http://manu.sporny.org/",
  "image": "http://manu.sporny.org/images/manu.png"
}
```

Listing 2.1: JSON-LD representation of Person

2.3.1. JSON-LD Data Model

It is important to distinguish between the JSON-LD syntax, and the data model which is an extension of the RDF data model. Resource Description Framework is a standard model for data interchange on the Web.

Following summaries the data model[9]:

- A JSON-LD document is a collection of *graphs* which comprises of one *default graph* and zero or more *named graphs*.
- The *Default Graph* is anonymous and may be empty
- *Named Graph* is a pair, consisting of IRI or blank node identifier and a graph.
- A *graph* is a labelled directed graph, which consists of a set of *nodes* connected by *edges*.
- Every *edge* is an IRL or a blank node identifier and has a direction associated with it.

- A *node* can be an IRI, a blank node, a JSON-LD value, or a list.
- A *graph* must not contain unconnected *nodes*
- A JSON-LD value can bea typed value, a string, a number, true or false, or a *language-tagged-string*
- A typed value consists of a value, which is a string, and a type, which is an IRI
- A *language-tagged-string* consists of a string and a non-empty language tag
- A graphical illustration of the data model is represented in the Figure 2.2

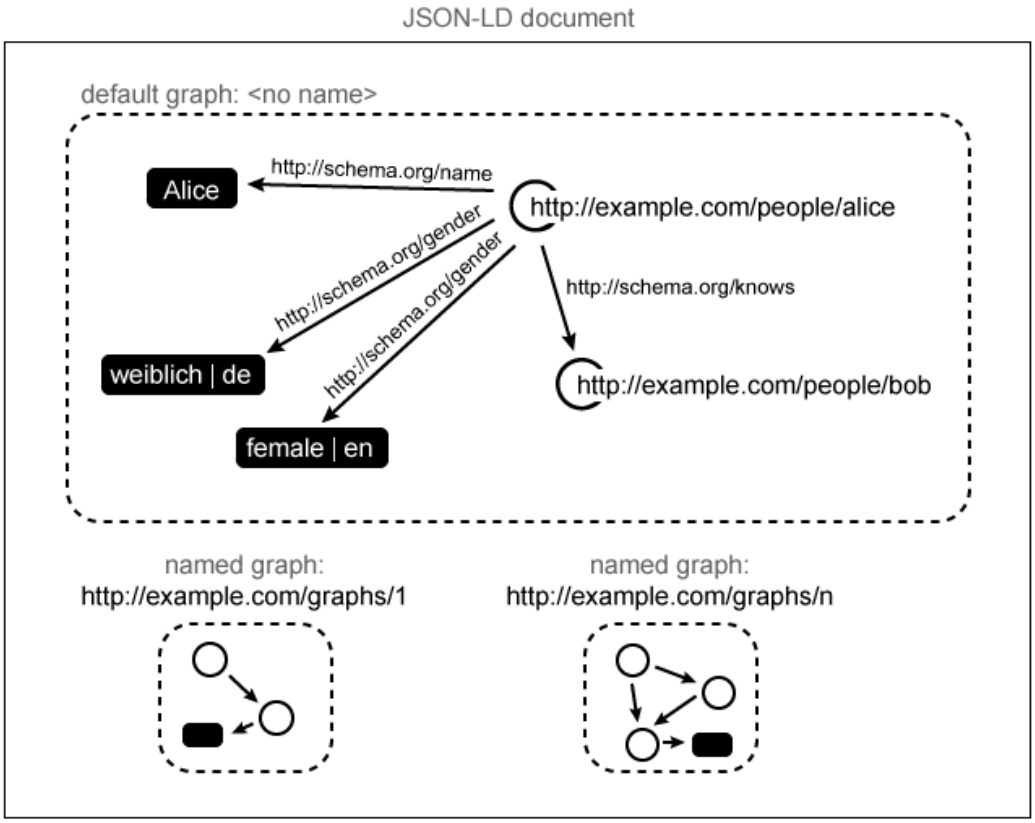


Figure 2.3.: JSON-LD Document

2.3.2. Embedding JSON-LD in HTML Documents

HTML script tags are not only used to add JavaScript Code but can also be used to embed blocks of data in documents. This method can be used to embed JSON-LD content easily inside an HTML document, by placing it in a script element with the *type* attribute set to *application/ld+json*. Following code snippet, provides an example of embedded JSON-LD as a part of HTML.

```
<script type="application/ld+json">
{
  "@context": "http://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
</script>
```

Listing 2.2: Embedded JSON-LD[9]

3. Related Work

3.1. Annotator.js Library

Annotator is an open-source JavaScript library using which we can easily add annotations to any web page. An annotation consists of comments, tags, links and other additional details. The new features and behaviours can be easily added to the library making it extensible.Annotator has 3rd party plugins allowing the annotation for not just web pages but also other file types like PDFs, EPUBs, videos, images and sound.[4]

An annotation is a JSON document which contains a number of fields that describe the position and content of an annotation within the specified HTML document:[3]

```
{
  "id": "39fc339cf058bd22176771b3e3187329",   # unique id (added by backend)
  "annotator_schema_version": "v1.0",         # schema version: default v1.0
  "created": "2011-05-24T18:52:08.036814",    # created datetime in iso8601
      format (added by backend)
  "updated": "2011-05-26T12:17:05.012544",    # updated datetime in iso8601
      format (added by backend)
  "text": "A note I wrote",                   # content of annotation
  "quote": "the text that was annotated",     # the annotated text (added by
      frontend)
  "uri": "http://example.com",                # URI of annotated document (added
      by frontend)
  "ranges": [                                 # list of ranges covered by
      annotation (usually only one entry)
    {
      "start": "/p[69]/span/span",            # (relative) XPath to start element
      "end": "/p[70]/span/span",              # (relative) XPath to end element
      "startOffset": 0,                       # character offset within start
          element
      "endOffset": 120                        # character offset within end
          element
    }
  ],
  "user": "alice",                            # user id of annotation owner (can
      also be an object with an 'id' property)
  "consumer": "annotateit",                   # consumer key of backend
  "tags": [ "review", "error" ],              # list of tags (from Tags plugin)
  "permissions": {                            # annotation permissions (from
      Permissions/AnnotateItPermissions plugin)
    "read": ["group:__world__"],
    "admin": [],
    "update": [],
    "delete": []
  }
```

```
26 }
```

Listing 3.1: Annotation Format[3]

Annotator uses XML Path Language (XPath) to indentify the starting and ending HTML elements. XPath uses ”path like” syntax to identify and navigate nodes in the HTML Document Object Model. In the above example, the relative XPath of the span element wrapped inside another span inside the 69th paragraph element is the start element. It also keeps track of the exact position of the annotated text inside the enclosing HTML element using *startOffset* and *endOffset* attributes. This is an effective way to keep track of the position of the annotated text, even for the texts ranging across multiple HTML elements. Annotator uses proprietary JSON structure which does not follow any W3C standard. In order to reuse this structure to represent a link, additional content has to be added to the current JSON structure to accommodate the position of both structured and unstructured data.

3.2. Annotea project

Annotea a RDF standard which enhances collaboration of documents via shared document meta-data based on tags, bookmarks, and other annotations.[20] In this context annotation mean comments, notes, explanations, or other types of external remarks that can be attached to any Web document or a selected part of the document without actually needing to touch the document. It is a predecessor to the web annotation data model proposed by W3C. Annotea is part of the Semantic Web efforts. The annotation meta-data can be stored locally or in one or more annotation servers and presented to the user by a client capable of understanding this meta-data and capable of interacting with an annotation server with the HTTP service protocol. Annotea standard is a predecessor to Web annotation data model. [18]

Resource Description Framework (RDF) is a model to interchange data designed for the web. Main feature of RDF is to facilitate merging of data even if the underlying schemas differ. It supports the evolution of schemas over time without requiring to change all the data consumers. RDF is an extention to the linking structure of the web, which uses URIs to name the relationship between things as well as the two ends of the link referred to as *triple*. Structured and semi structured data can be mixed, exposed and shared across different applications using this simple structure. This linking structure contains a directed and labeled graph. The edges represent the named link between two resources and nodes are the resources.[11]

RDF provides a simple and flexible framework for describing properties of any Web resource. In its most simple level, RDF provides (resource, property, value) triples. A single triple is a statement that indicates that a resource has a given property with a given value. The resource can be any Web resource identified by a URI. The value may be a literal string or may be the URI of another Web resource.

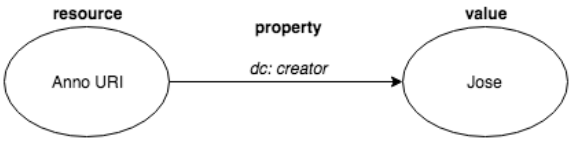


Figure 3.1.: RDF Triple Schema[20]

In Annotea, annotations are described using a dedicated RDF schema and are stored in dedicated annotation servers. The annotation server stores the annotations in an RDF database. Users query this server to retrieve an existing annotation, post a new annotation, modify an annotation or delete an annotation. HTTP protocol is used to communicate with the annotation server.

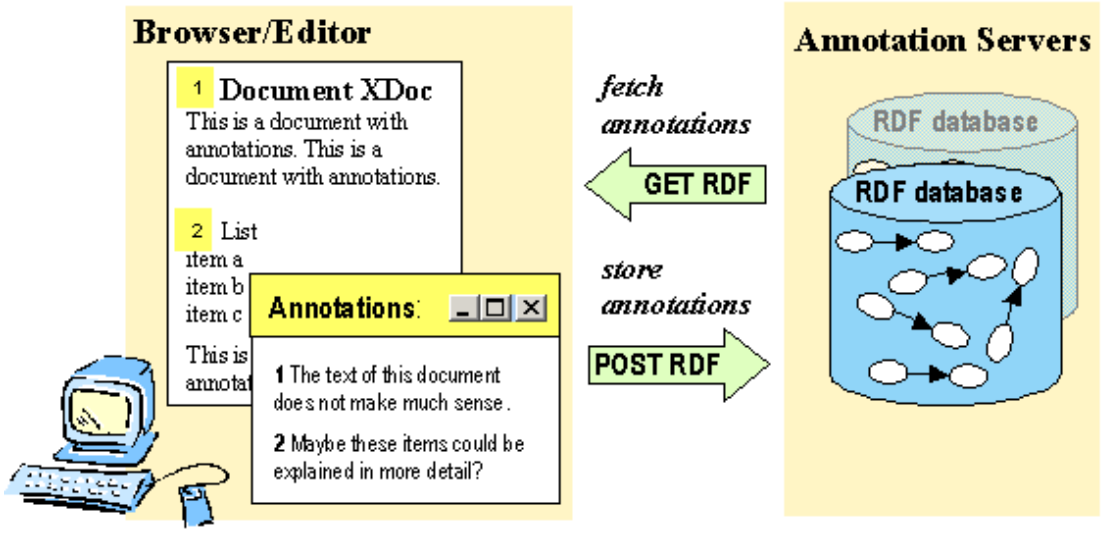


Figure 3.2.: Annotea Architecture

3.3. Entity RecOgnition in Context of Structured data (EROCS)

Entity RecOgnition in Context of Structured data (EROCS) is a novel system for linking a given text document, which is mostly unstructured, with relevant structured data. EROCS views structured data as a set of *entities* and identifies the *entities* that best match the given document. EROCS also embeds the entities in the document, effectively creating links between the structured data and specific text within the document.[25]

Inputs to the EROCS system are a text document filtered to contain only relevant terms and the given database, viewed as a set of entities and associated content information (structured information). These entities are defined in terms of a collection of entity templates that specify the location of each entity and its context information in the relational database. EROCS matches the context information of the current entity with the docu-

ment and finds the entities that match best to the current entity along with their embedding. This task performed by EROCS is similar in spirit to dictionary-based named-entity recognition.[19]

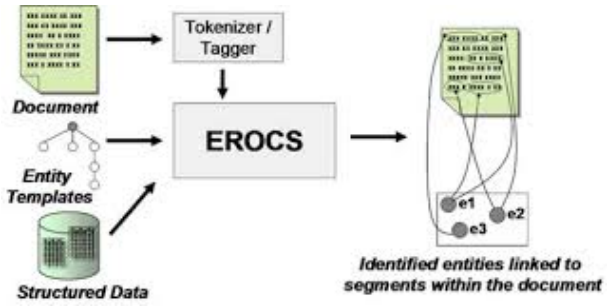


Figure 3.3.: EROCS Overview[25]

4. Extended Standard - JSON-LD for link creation

4.1. Web Annotation Data Model

Annotations convey information about a resource or association between resources. Simplest example is a comment or a tag on a section of web-page or an image or a blog post. Annotating, the act of creating associations between distinct information, is a pervasive activity online in many guises. Web users makes comments on various online resources using either built-in tools in the host website, external web-services or an annotation client. In addition, there are a plenty of ”sticky note” systems and stand-alone annotation systems.

Web annotation data model specification describes a common approach to express these annotations in a standard way. The world wide web consortium describes the data model as *”an extensible, inter-operable framework for expressing annotations such that they can easily be shared between platforms, with sufficient richness of expression to satisfy complex requirements while remaining simple enough to also allow for the most common use cases, such as attaching a piece of text to a single web resource”*[10]. The primary aim of the model is to provide a standard description model and format and to facilitate the sharing of annotations between systems. This interoperability is either for sharing with others, or exporting private annotations across devices or platforms. The shared annotations should be easily intigrated into existing collections and reused without loss of significant information. This model uses JSON-LD serialization for annotation documents.

The Web Annotation Data Model is defined using the following basic principles:[10]

- An *Annotation* is a directed graph that represents a relationship between resources
- There are two types of resources in this relationship, *Bodies* and *Targets*
- Annotations have 0 or more *Bodies*
- Annotations have 1 or more *Targets*
- The contents of *Body* and *Target* resources are related to each other, usually in the direction from *Body* to *Target*
- Annotations, Bodies and Targets may have their own properties and relationships, typically including creation and descriptive information
- The intention behind creating annotations is represented with the help of *Motivation* resource

The web annotation data model is combined with extended JSON-LD standard to create a new Link data model to store the link.

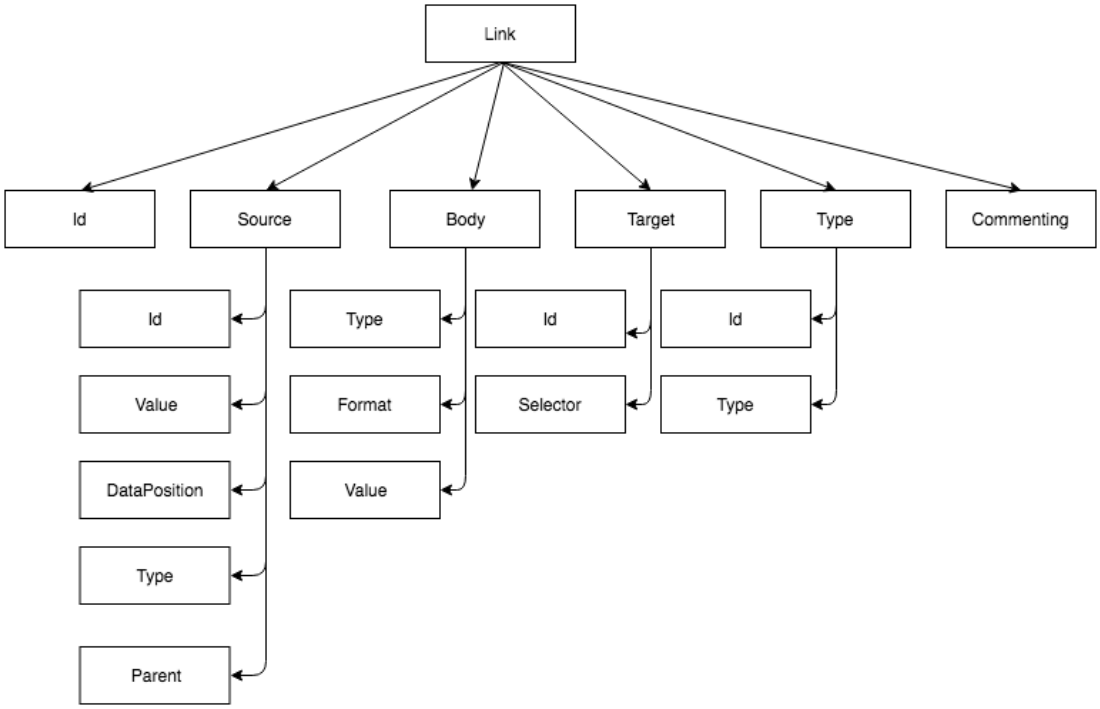


Figure 4.1.: Link Data Model

The attribute resources of Link data model are:

- **ID:** Identifier for the JSON-LD.
- **Source:** Represents linked structured data
  - **ID:** Attribute ID of SocioCortex Wiki entity
  - **Value:** Structured content at the time of creation
  - **DataPosition:** Position of the value in the attribute value array (0 - by default)
  - **Parent:** Represents the Wiki entity the unstructured data belongs to
  - **Type:** Says if the structured data linked is part of same or different wiki entity.
- **Body:** This resource along with Target represents linked Unstructured data
  - **Type:** Represents unstructured content type
  - **Format:** Format of the linked unstructured content
  - **Value:** Unstructured content at the time of creation
- **Target:** Represents the target HTML element of the linked unstructured content.
  - **Id:** ID of the HTML selector
  - **Selector:** Span Tag HTML selector that represent the linked unstructured content.

- **Type:** Type of link (Same or related link)
- **ID:** Additional User comments

Following code snippet provides an example annotation based on the model.

```
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/anno1",
  "type": "Annotation",
  "body": "http://example.org/post1",
  "target": "http://example.com/page1"
}
```

Listing 4.1: Web Annotation JSON-LD

The Web Annotation Data Model only provides a way to annotate certain parts of web resources. It can be extended for our use case to represent relationship between two different types of resources, such as, structured and unstructured information. The *Body* and *Target* resources are used to represent the *Unstructured data* to be annotated. The *Source* resource points to the *Structured data* to be annotated. Hence, establishing a link between structured and unstructured data in a single JSON-LD object.

Following code snippet provides an example of such a link:

```
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "link_2",  //ID of the span element embedded as unstructured data
  "body": { // Represents unstructured content
    "value": "2017 ",
    "format": "text/plain",
    "type": "TextualBody"
  },
  "source": { //Represents attribute in structured data linked
    "id": "1t4b4rniixb8s", // Attribute ID from SocioCortex of the structured
        data linked
    "value": 2017,
    "DataPositionSelector": 0 // Position of the value in the attribute array
  },
  "created": "2018-06-02T23:10:15.299Z",
  "motivation": "describing",
  "target": { // Target HTML element in the unstructured content
    "id": "c7gt17zhps0l", // Entity ID from SocioCortex of the hybrid wiki
    "selector": {
      "value": "span#link_2",
      "type": "CssSelector"
    }
  },
  "type": { // Type of link
    "id": 0, // 0 - same link, 1 - related link
    "type": "Same Link"
  },
  "creator": { // Author who established the link
    "name": "Shivguru Rao",
    "type": "Person"
  }
```

```
}
```

Listing 4.2: JSON-LD Link with Structured and Unstructured data

-

4.2. Annotating unstructured data

Locating structured information is straight forward as each *AttributeKey* of a structured *Entity* in the hybrid wiki has a unique identifier. Where as, unstructured data is in the form of free text, usually a collection of HTML elements with no proper structure.

Locating the position of text in the unstructured data can be narrowed down to the deepest enclosing HTML element surrounding the text. But, locating text with in this element has to be done by either counting the number of characters within the element or surrounding the annotation text with an enclosing HTML element, without effecting current HTML DOM structure.

Inline HTML element *span* can be used for this purpose, with a unique identifier, through which the corresponding linked JSON-LD is identified. An example link span looks like:

```
<span id="link_2" class="linked-span">2017</span>
```

Listing 4.3: Span tag representing linked Unstructured data

4.3. Cross-Entity Links

Related data need not be present in the scope of a single *Entity*, rather it can span across multiple *Entities*. The above link representation, does not facilitate for cross-entity links. To overcome this shortcoming, additional information about the *type* of link has to be added to the JSON structure along with the details of the *Entity* to which the structured data belongs to inside the *Parent* resource.

An example which represent cross-entity links:

```
"source": {
    "id": "1t4b4rniixb8s",
    "value": 2017,
    "parent": {
      "id": "1rn347knpl6ii",
      "name": "Q2-2017"
    },
    "DataPositionSelector": 0,
    "type": "Inter Entity"
  }
```

Listing 4.4: Cross Entity Links

4.4. Link comments

Web annotations contain additional information in the form of comments, facilitated by web annotation data model. Same feature can be extended to the link JSONs as well, by using *Commenting* resource. An example link JSON-LD with additional comments:

```
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "link_2",
  "body": {
    "value": "2017 ",
    "format": "text/plain",
    "type": "TextualBody"
  },
  "commenting": "Cross-entity link between Q1-2017 and Q2-2017",
  "source": {
    "id": "1t4b4rniixb8s",
    "value": 2017,
    "parent": {
      "id": "1rn347knpl6ii",
      "name": "Q2-2017"
    },
    "DataPositionSelector": 0,
    "type": "Inter Entity"
  },
  "created": "2018-06-02T23:10:15.299Z",
  "motivation": "describing",
  "target": {
    "id": "c7gt17zhps0l",
    "selector": {
      "value": "span#link_2",
      "type": "CssSelector"
    }
  },
  "type": {
    "id": 0,
    "type": "Same Link"
  },
  "creator": {
    "name": "Shivguru Rao",
    "type": "Person"
  }
}
```

Listing 4.5: JSON-LD with additional comments

5. Prototype Implementation

In the following chapter, the prototypical implementation is presented. The first section explains the technical foundation for the implementation. The second part elaborates the overall system architecture along with the component diagram. In the third section, core features are explained in detail, which help us answer some of the research questions. In the final section, the API used to retrieve and store the data and the database structure used to store the created links is provided.

5.1. Technical Foundation

SocioCortex system is used as the back-end system and Angular 5 framework along with Bootstrap 4 library is used to implement the front-end tool of the prototype implementation. Introduction to these systems and frameworks will provide the technical foundation.

5.1.1. SocioCortex as a Hybrid Wiki source and storage for Links

SocioCortex is a model based collaboration environment, which is a project of the chair "Software Engineering for Business Information Systems (Sebis)" at the Technical University of Munich. They describe SocioCortex as "The Social Information Modelling Platform for Collaborative, Evolutionary Data and Process Management".[13] SocioCortex uses Hybrid Wiki approach explained in Section 2.1. The complete SocioCortex ecosystem is provided in Figure 5.1. On the application side, there is the default client suite and the vertical applications. In addition to these there are also content sources and identity providers connected to SocioCortex. In the default client suite the application modeler, content manager and visualizer is present. Some of the vertical applications are Spread-sheet 2.0 and Lexalyze.[14]

SocioCortex provides an easily accessible and well documented REST API and encourages people to implement their own applications on top of SocioCortex. The prototype of this thesis uses SocioCortex as the source of hybrid wiki pages. Each wiki page contain structured and unstructured information represented as an *Entity* inside a *Workspace*. SocioCortex is also used as a database to store the created links in the form of JSON-LDs. A dedicated *Entity* is created to store and retrieve the links. The data storage and API usage is elaborated in detail in Section 5.4.

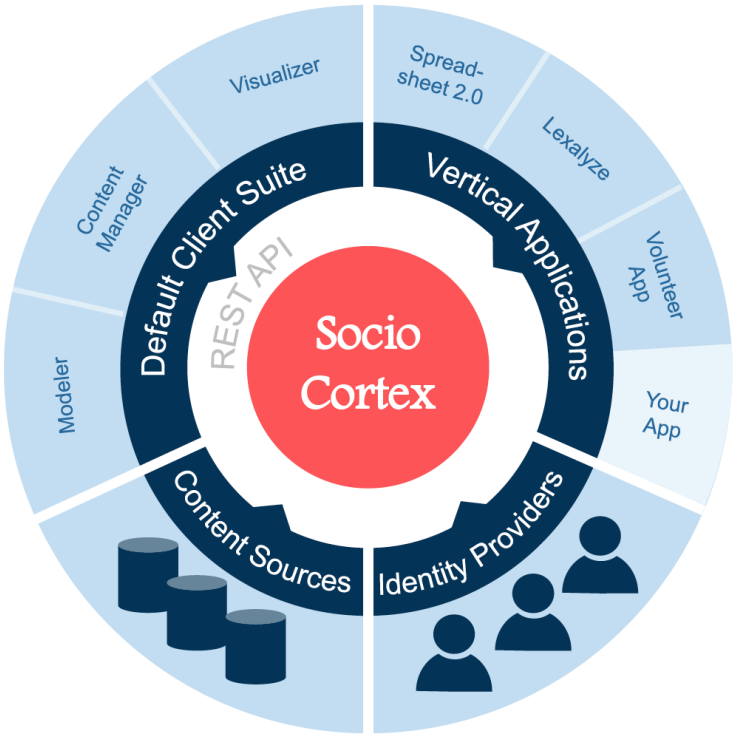


Figure 5.1.: SocioCortex Ecosystem[14]

5.1.2. Angular 5 Framework

Angular is a development platform used to build mobile and desktop front-end web applications in Typescript/JavaScript.[2] The basic building blocks of angular application are *modules* and they provide a compilation context to for *components*. An app always has at least a *root module* which is bootstrapped at the start of the application and collection of *feature modules* loaded as required. /*Components* define *views* or *templates*, which contain the set of screen elements built with HTML and CSS. *Components* contain the business logic for the *views* it contain and provides them with the necessary data to render the content. Every app has at least a *root component*. *Components* also uses services, which provide specific functionality not directly related to *views*. *Services* are injected as dependencies in the components.[1] The architecture of the angular application, which explains how *modules*, *components*, *templates* and *services* are related, is provided in Figure 5.2.

The architecture of the prototype front-end is based on the architecture of the angular application, which is explained in detail in component diagram Section 5.2.2.

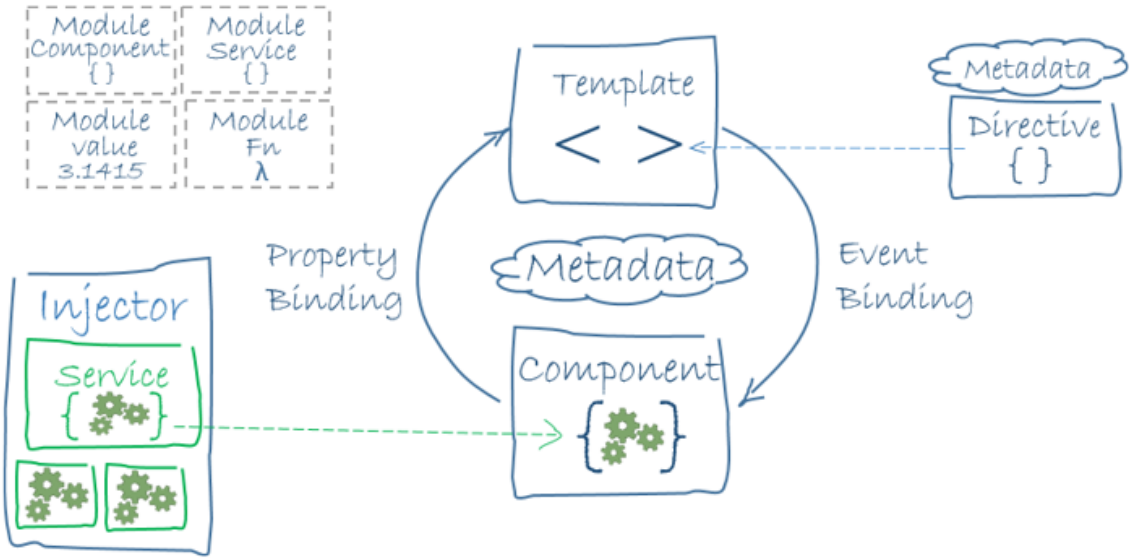


Figure 5.2.: Angular Architecture[1]

5.1.3. Bootstrap 4 Library

Bootstrap is a front-end library which is used to design websites using HTML and CSS based UI components.[7] Bootstrap provides a grid system, which divides the web page into fluid rows and columns.[6] Bootstrap is responsive, as the contents are organized in a web page based on the size of the screen in which it is rendered. Bootstrap provides an extensive set of reusable UI components, some of the important components are: Navbar, Button, Input, Modal, Popover. Implementation demonstrations can be found in the examples section of bootstrap website[5] and in the core features of the prototype explained in Section 5.3. The layout of the prototype is generated using Bootstrap's grid layout and the components mentioned before are used extensively in the prototype implementation.

5.2. Architecture

In this section the overall system architecture of the Prototype along with the detailed component diagram.

5.2.1. System Architecture

The system architecture is a client server architecture where the Angular 5 prototype application is the client and the SocioCortex system is the server. The client and server interact using RESTful API provided by SocioCortex. JavaScript Object Notation (JSON) is the data exchange format used for the communication between the client and the server. Detailed explanation of the API endpoints used is provided in Section 5.4.

The client application is implemented using Angular 5 Framework. Angular application is made up of *modules* which contain hierarchy of *components* and related *templates* which

forms the view of the application. Components also use reusable *services* to interact with the server and use shared business logic and resources. The detailed component hierarchy is explained in Section 5.2.2. When the prototype is loaded on the browser, Angular application is compiled into plain web application with HTML, CSS and JavaScript files.

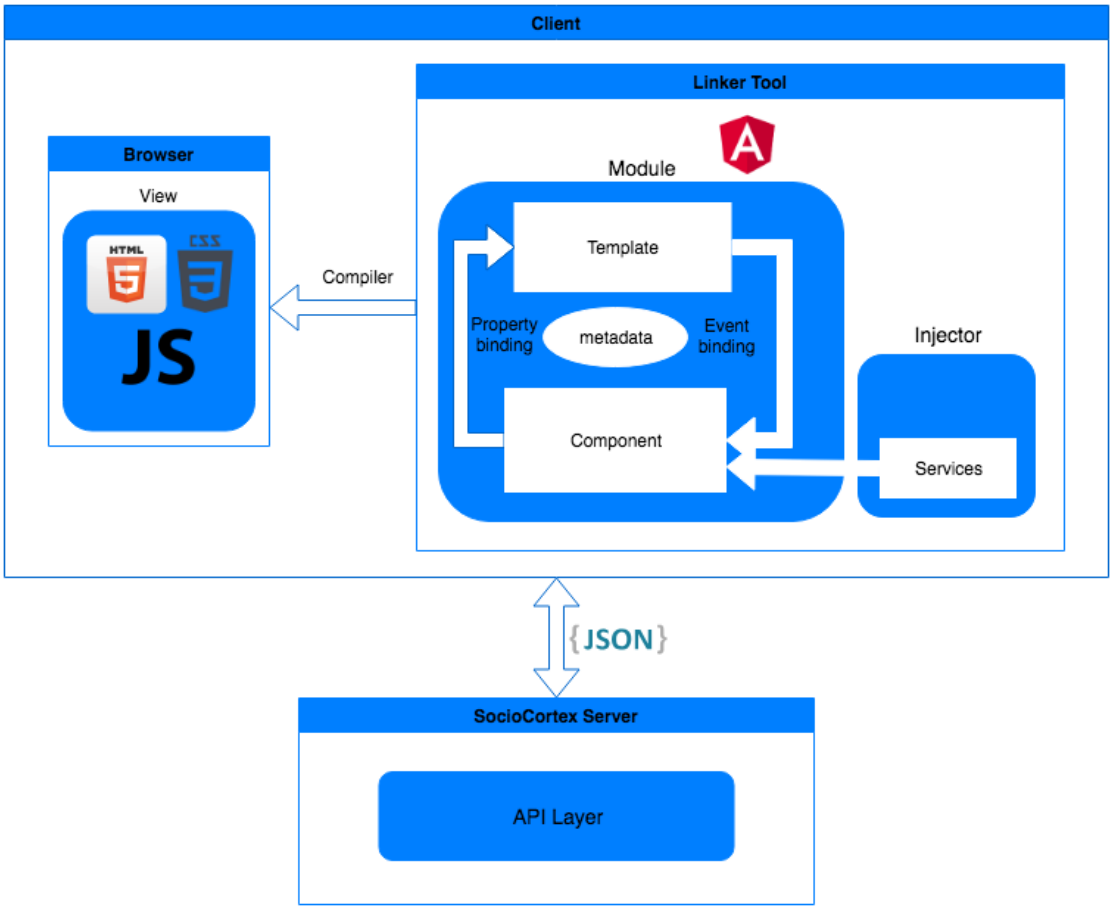


Figure 5.3.: System Architecture

5.2.2. Component Diagram

Prototype consists of client web application implemented using Angular 5 Framework and SocioCortex system as the data source and the database for links.

As explained in the technical foundation of Angular framework in Section 5.1.1, an angular application consists of *modules* and *components*. Similarly, our prototype **LinkerTool** consists of one default module *App module*. *App module* consists of hierarchy of *components* and *services* which is depicted in the Figure 5.4.

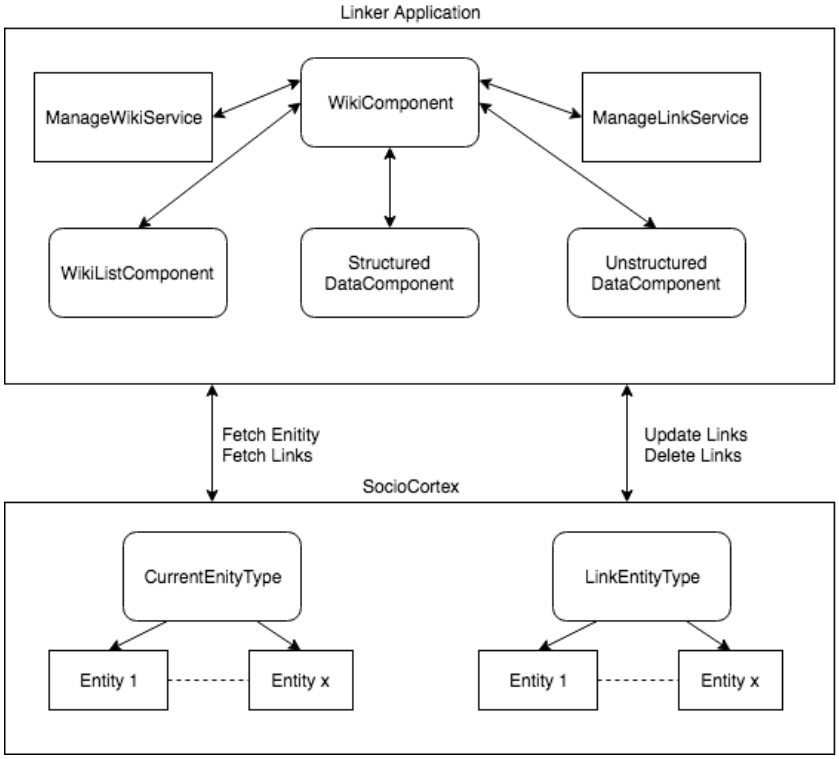


Figure 5.4.: Prototype Component Diagram

A component controls a patch of screen called a *view*. It consists of a *template* with HTML and angular related markup, CSS styling and the business logic corresponding to the *view*. Various components that make the prototype application are:

- **WikiComponent:** Is in the top of the hierarchy, whose view cover the entire web page. The main task of this component is to fetch the wiki pages and send the wiki page object (Entity) to its children. All the event handlers(click on a link and hover on a link) are registered in this component as well, when some changes are detected to the current active wiki page object.

  The children of the WikiComponent are:

  - **WikiListComponent:** This component is the left most view of the wiki web page in terms of the page layout. It keeps track of all the wiki pages imported from SocioCortex and display them as a list. Upon selection of a wiki page from the list it triggers the change event of current active wiki page, which is propagated across the entire application.
  - **Unstructured Component:** This component is the middle view of the wiki web page. It is used to render the unstructured content of the wiki page. It is also responsible for handling the creation of links.
  - **Structured Component:** This component is the right most view of the wiki web page. It is used to render the structured content of the wiki page in a tabular

format. Structured data in the form of key-value pairs are rendered in a table of two columns.

- **LoginComponent:** This component takes care of user authentication where it allows users to login to the SocioCortex system and maintains the validity of the access token once logged in.

A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well. A service is a singleton which usually contain shared resources or business logic required across multiple components. Services has to be injected in a component before use. Prototype consists of following services:

- **ManageWikiService:** This service is used to fetch the wiki page objects (Entities) using SOcioCortex API and and them to the wiki list. It also keeps track of the current active wiki page.
- **ManageLinkService:** This service is used to fetch all the links for a particular entity, creation of a new link, deletion of links.
- **LoginService:** This service is used for user authentication management.

The SocioCortex system consists of wiki pages in the form of *entities* and these *entities* are of particular *entity type* and they belong to a specific *workspace*. These *entities* are fetched and modified with the help of SocioCortex API which is discussed in detail in Section 5.4.

5.3. Core Features

In this section, the features of the prototype are explained in detail. It also helps us answer some of the research questions mentioned in Section 1.2.

5.3.1. Import Entity

An *entity* is a wiki page which consists of structured and unstructured information. This section describes different ways in which an entity can be imported in the prototype.

An *entity* can be imported using two methods:

- **Entity ID:** Each *entity* has an entity ID. We can directly import an *entity* if we know its ID. Figure 5.5 shows the UI for entering entity ID to fetch the *entity* object.

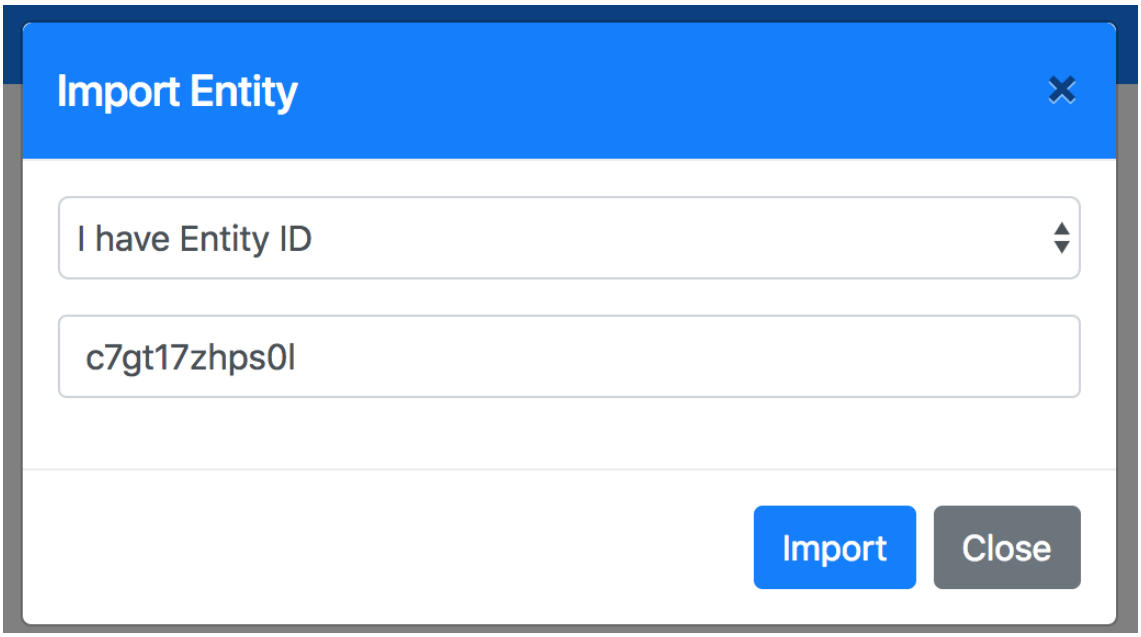


Figure 5.5.: Import Entity using ID

- **Workspace Name:** An *entity* is of a specific *entity type* and belongs to a *workspace*. Using *workspace* name all the *entity types* inside that. *workspace* is fetched. Using *entity type* all the *entities* that belong to the *entity type* is fetched. Finally from the *entity* name actual *entity* object is fetched. Figure 5.5 shows the UI for importing the *entity* Q1-2017 of *entity type* Quarter Result which belongs to the *workspace* Apple Quarter Results.

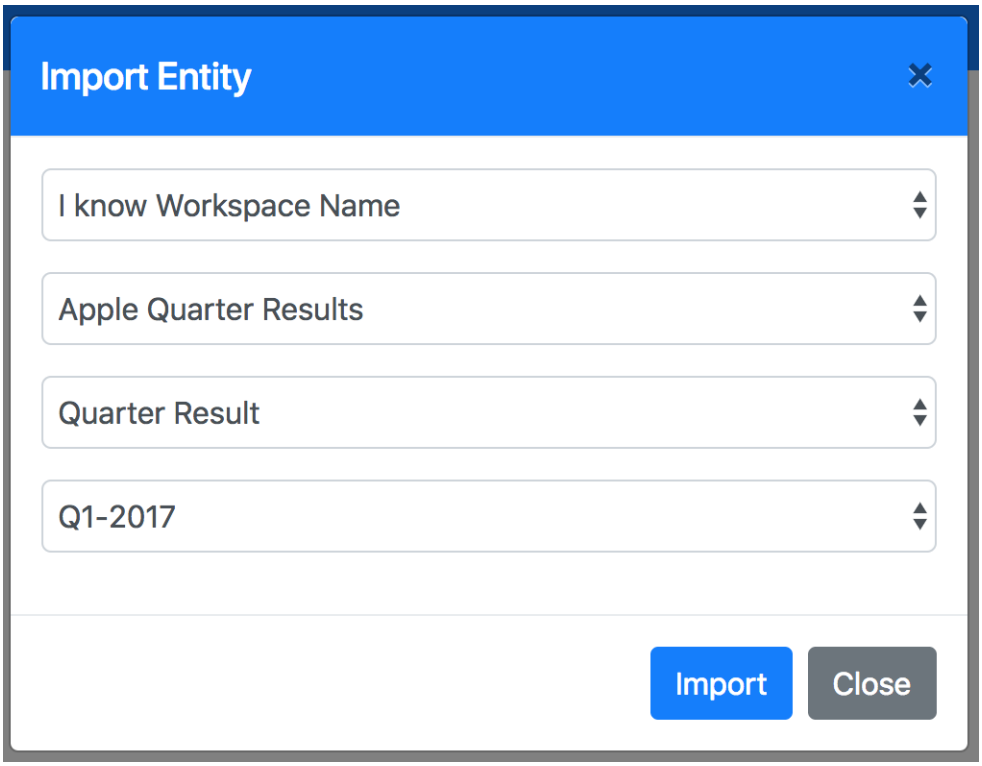


Figure 5.6.: Import Entity from Workspace

Once the *entity* is imported all the links for this *entity* is also imported and stored as part of imported *entity* object. How the links are stored and fetched from SocioCortex is explained in detail in Section 5.4.

5.3.2. Display Wiki

Once the *entity* is imported it is stored in the list of *entities*.An *entity* object consists of unstructured data in the form of HTML text and structured data in the form of *attributes* and *values*. Figure 5.6 shows the rendered wiki page of *entity* Q1-2017. Entire web page is divided into 3 vertical sections or columns. The left most section consists of list of imported *entities* and is rendered using WikiListComponent. The middle section is where the unstructured data is displayed along with *entity* name and person who has edited the *entity* recently. This section is rendered and handled using UnstructuredDataComponent. The source HTML structure is preserved and rendered as it is fetched. The right most section is where the structured data is displayed in tabular format. StructuredDataComponent is responsible for rendering and handling this section. While displaying the wiki, all the links of the *entity* is embedded as part of HTML page.

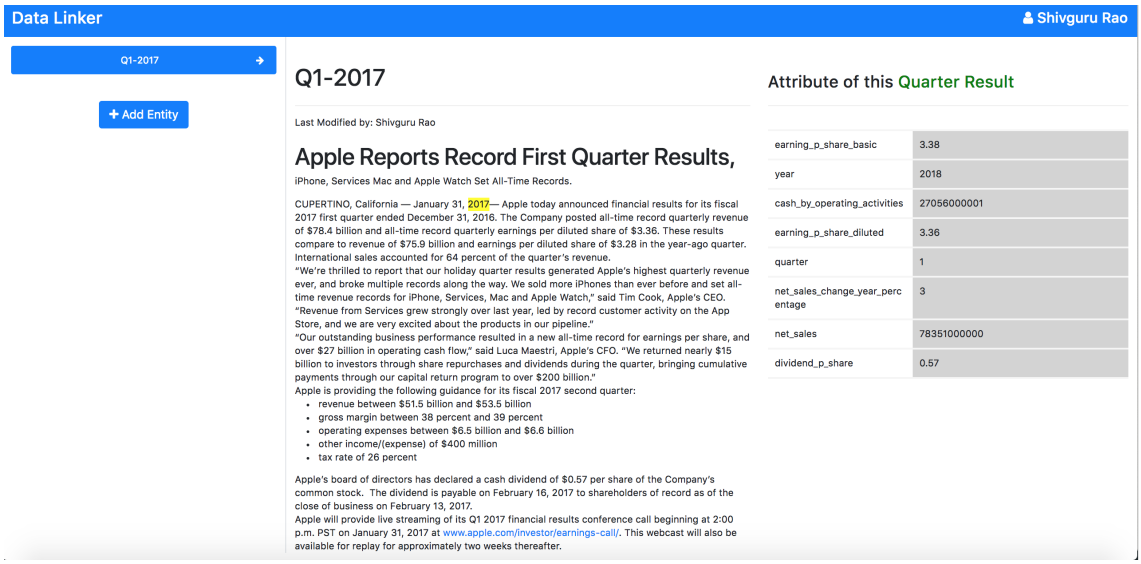


Figure 5.7.: Displayed Wiki with Structured and Unstructured Data

5.3.3. Create link

Creation of a link is done by selecting some text in the structured data section. A pop-up is created with the list of attributes and its values shown as a hierarchical tree structure. When the user selects an attribute and click on link button, JSON-LD for the link is created and stored in SocioCortex. This JSON-LD is also added to the embedded JSON-LD which is part of the HTML. Additional user comments can be added as a part of the link. The *entity* to which structured and unstructured data belongs to is stored as part of JSON-LD in order to differenciate between different entity types Three types of links can be created:

- **Annotations:** A simple web annotation can be created by not selecting any structured attribute to be linked to the selected unstructured data. Additional comment has to be added to this type of annotations.
- **Intra-Entity Links:** Links where both structured and unstructured data belongs to same *entity*. By default the structured data attributes of the same *entity* is loaded while creating a link.

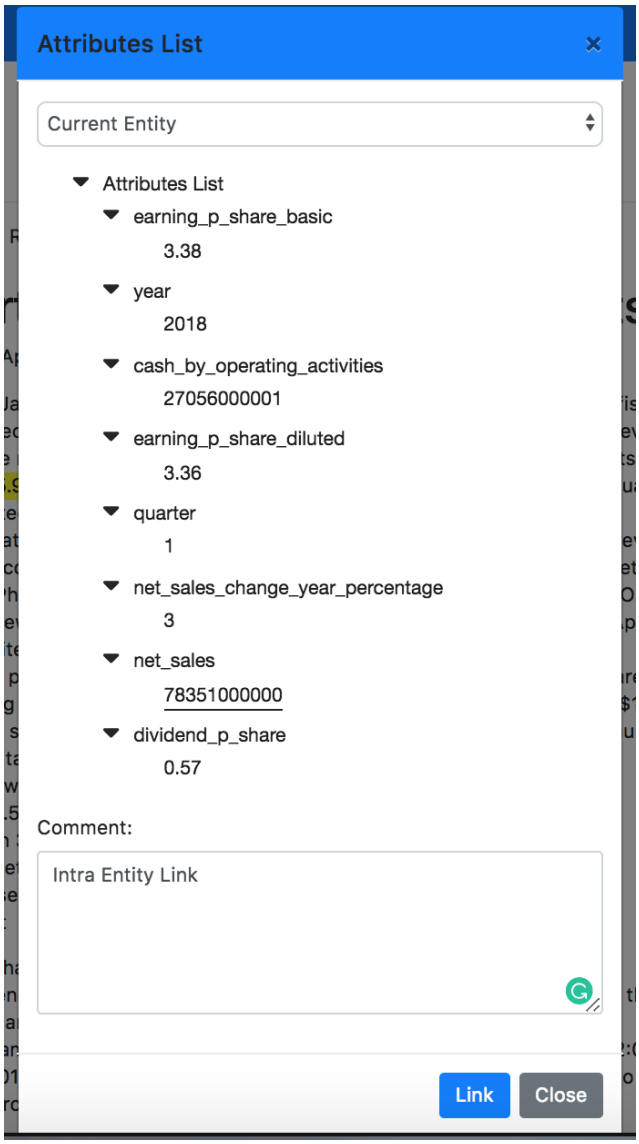


Figure 5.8.: Create Intra-Entity Link

- **Inter-Entity Links:** Links where structured and unstructured data belongs to different *entities*. In order to select a different entity, we need to fetch the structured data attributes of different entity and is done by allowing users to select an *entity* through *workspace-¿entity type. -¿entity* as discussed in section 5.3.1.

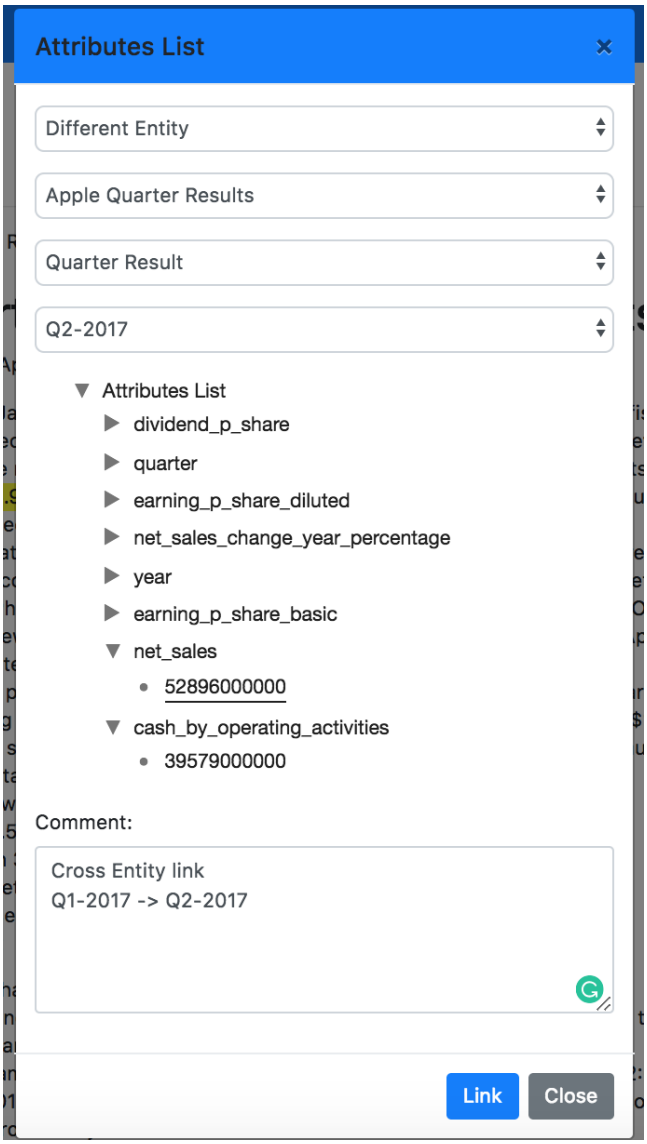


Figure 5.9.: Create Inter-Entity Link

5.3.4. Display link

When the wiki is rendered the Linker service handles all the existing links for this wiki. In the unstructured data section links are highlighted in *yellow* for links whose contents are not changed since link creation and in *red* for links where structured or unstructured data is changed after creation of the link. Change detection is explained in detail in the coming section.

When we hover over the link in unstructured section corresponding attribute value in the structured data section is also highlighted, if it is an Intra-Entity link. When a link is clicked all details of the link is displayed as a popover next to the highlighted text in the unstructured section. The popover contains details about following:

- **Type of link:** Whether it is an Intra or Inter-Entity link. Same link if the content of unstructured and structured data are same and similar link if they are different.
- **Structured Content:** Exact content of the structured attribute value linked. This will be empty if it is a simple annotation.
- **Unstructured Content:** Exact content of the unstructured data linked.
- **Created by:** Name of the user who has created the link
- **Created on:** Data and time at which the link was created.
- **Comments:** Additional comments added by the users
- **Variance:** Variance or deviation is only calculated for the numeric type of data. Both structured and unstructured contents are processed and converted in to numeric type if possible. Firstly, currency is removed and words are checked if they are of numeric equivalent (eg: mil = million = 1000000). If any numeric equivalent is found, it is multiplied by the preceding numeric value to get a whole number. Once the processing is done, if both the contents are numeric following formula is used to calculate deviation. **(Higher value - Lower Value)/Lower Value*100**

Figure 5.10.: Create Inter-Entity Link

5.3.5. Link change detection

One of the research questions of this master thesis is "How to detect and handle the changes?". Change detection to the link contents is fairly straight forward, the unstructured link content is enclosed by a *span* tag and the value at the time of creation stored in the link JSON-LD. These two values can be compared to check if there are any changed to the unstructured data. Similarly, Attribute id and the location of the attribute value of the structured content is stored as part of link JSON-LD, comparing the stored content with the current attribute value will give the changes, if there is any.

During the rendering of wiki page all the links are checked for consistency and if there are any changes, they are stored in the *entity* object. Corresponding unstructured content is highlighted in red color instead of yellow. In the link details popover, old and new values of the changed content are displayed.

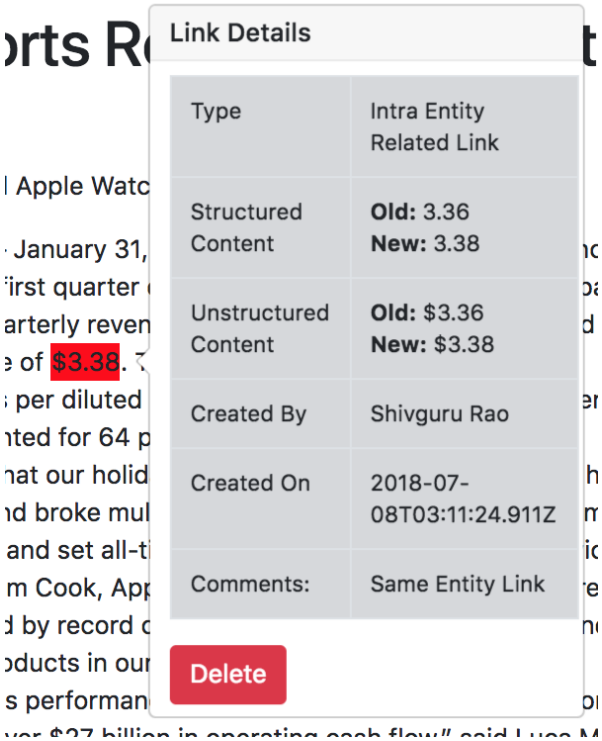


Figure 5.11.: Link

5.3.6. Delete Links

In the link details view, a delete button is provided to delete the links. When a link is deleted, the corresponding JSON-LD is removed from the embedded HTML and the JSON-LDs are updated to SocioCortex. In the next section the storage and retrieval of JSON-LDs is explained in detail.

5.4. Data Storage and Access using SocioCortex API

In this section the API architecture of the SocioCortex API is explained along with the endpoints used in the prototype implementation. The API

5.4.1. SocioCortex API

The API layer of SocioCortex is used to access the wiki pages (*entities*) as well as store the links created in its own entity. In this Section the API endpoints used to build the prototype is explained in detail along with the structure of the *entities* where the links are stored.

The API of the SocioCortex is designed in seven layers, each layer has a very specific purpose. The endpoints provided by the first three layers (bottom-up) are used to develop the prototype. Figure 5.12 defines the API structure of SocioCortex.

- **Annotated Versioned Linked Content Graph:** Provides a way to structure an unstructured entity by adding new attributes or links to other entities.
- **Multiple Dynamic Schemata:** This layer introduces concepts to incrementally design data models by defining types, properties, and integrity constraints. Together with the first layer, it implements the Hybrid Wiki concept.
- **Role-Based and Discretionary Access Control:** This layer provides user integration and management along with authentication and authorization.

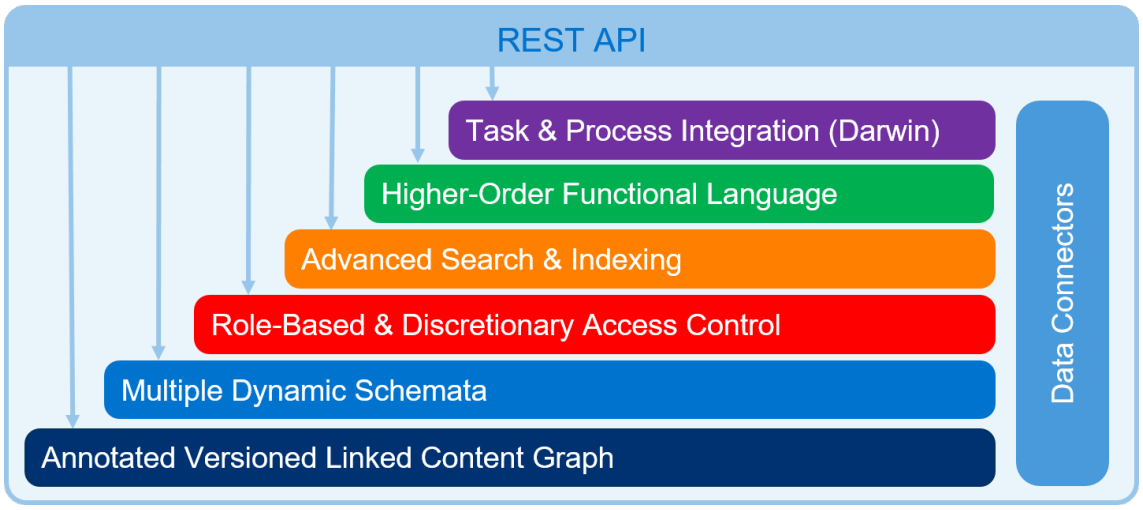


Figure 5.12.: SocioCortex API Structure[16]

The SocioCortex instance running on server.sociocortex.com is used in the prototype to retrieve and store the *entities* and *links*. General URI schema of a SocioCortex API endpoint is:

Figure 5.13.: URI Schema

List of API endpoints used and their purpose is explained below:[15]

- **User Login:** User credentials are sent to the server in the payload of a HTTP POST request. A JWT token along with expiry date and user information is received after successful authentication.

```
1  POST https://server.sociocortex.com/api/v1/jwt
2  BODY
3  {
4      "username":"$EMAIL@ADDRESS.com",
5      "password":"$PASSWORD"
6  }
```

Listing 5.1: User Login

- **Get all Workspaces:** List of all workspaces user has access to is sent back. Workspace id and name are among the information sent back.

```
1  GET https://server.sociocortex.com/api/v1/workspaces
```

Listing 5.2: Get Workspaces

- **Get all EntityTypes of a Workspace:** List of all entitytypes that belongs to a given workspace. Entitytype id and name are among the information sent back.

```
1  GET https://server.sociocortex.com/api/v1/workspaces/{workspaceid}/
       entityTypes
```

Listing 5.3: Get Entitytypes

- **Get all Entities in a Workspace:** List of all entities that belongs to a given workspace. Entity id and name are among the information sent back.

```
1  GET https://server.sociocortex.com/api/v1/workspaces/{workspaceid}/
       entities
```

Listing 5.4: Get Entities

- **Get single Entity:** Fetch a single entity using entity id. All the information about an entity is retrieved along with unstructured data (content) and structured data (attributes).

```
GET https://server.sociocortex.com/api/v1/entities/{entityid}/entities
```

Listing 5.5: Get Entity

- **Update Entity:** Provides a way to update an entity using HTTP PUT request. A JSON object with the changed data is sent as a part of the request body. In the below example, the content of the entity is changed.

```
PUT https://server.sociocortex.com/api/v1/entities/{entityid}
BODY
{
    "content":"Chnaged content goes here"
}
```

Listing 5.6: Update Entity

- **Update Attribute:** Provides a way to update an attribute using HTTP PUT request. A JSON object with the updated value is sent as a part of the request body.

```
PUT https://server.sociocortex.com/api/v1/attributes/{attributeid}
BODY
{
    "values":[{
        "id":"1d94htksgwy2i",
        "name": "Shivguru Rao"
    }]
}
```

Listing 5.7: Update Entity

5.4.2. Link Storage in SocioCortex

All created link JSON-LDs are stored in SocioCortex in a dedicated workspace **Structured Unstructured Annotations**. Entity type **Annotation** with the only attribute **AnnotationJSON** is created inside the workspace. For each entity whose data is linked using a prototype a corresponding entity is created in **Structured Unstructured Annotations** workspace of type **AnnotationJSON**. The name of the newly created entity is the entity id of the linked entity. JSON-LDs are stored as array of strings and every time a link is created or deleted the complete array is updated. This is the best feasible solution to store the links as the source structure of the unstructured data is not altered. Another way to store the link is to embed it as part of the HTML and store it as a part of unstructured data content inside script tag. Though it does not change the structure of the HTML document, it may cause a security issue as unstructured content is stored as a part of database and if allowed it can be vulnerable to security threats like cross site scripting (XSS) attacks.

6. Conclusion

In this chapter whole thesis is summarized and research questions of the thesis discussed in Chapter 1 is revisited. Furthermore future enhancements and improvements are proposed to the implemented prototype.

6.1. Summary

Wiki-platforms are the side effects of the evolution of new web 2.0 paradigm. Many of these platforms are designed using hybrid wiki model, which mainly combines structured and unstructured information. Many a time the data provided in the structured and unstructured parts are related, though there is no link established between them. In this thesis, we explore the existing JSON-LD standard to create web annotations and come up with a way to extend it which represents links between structured and unstructured information. A prototype tool is implemented based on SocioCortex HybridWikis to create and maintain the links which answers the research questions mentioned in Section 1.2.

6.2. Research questions revisited

6.2.1. What should be the format of the link?

A link in a hybrid wiki is the meta data added on top of the existing data. In the web, simplest way to add additional information to the existing web page is through web annotations. W3C has provided a standard for web annotations. The links we are trying to create is similar in concept to the web annotations. Chapter 3 answers this question by proposing a way to annotate structured and unstructured information using web annotation data model.

6.2.2. Are there any annotation standards to link data?

Web annotations specified in the web annotation data model takes the structure of a JSON. W3C also proposes a standard to represent linked data in the web ecosystem called JSON for Linked Data (JSON-LD). JSON-LD revolves around the concept of *context*, which specify the additional information about the content of the JSON. JSON-LDs are typically used to represent the linked web documents, in this use case it can also be used and extended to represent the link between structured and unstructured information. Chapter 3 also helps answer this question by specifying a way to represent structured and unstructured information as part of linked JSON-LD document.

6.2.3. How can we store and retrieve links?

As links takes the form of JSON-LD documents it can be easily embedded as a part of HTML web page without altering the overall structure of the HTML document. Embedding the JSON-LDs provides the added benefit of single source of data. As the prototype is a standalone tool, the link JSON-LDs for a wiki page is embedded as a part of the web page. In order to persist the links it has to be stored somewhere. This problem is solved in Chapter 5 Section 5.4.2, where a dedicated workspace and entity is created in SOcioCortex to store the JSON-LDs. When a wiki page is loaded in the prototype, corresponding links are also retrieved and embedded as part of web page which provides the best of both worlds.

6.2.4. How to detect and handle the changes?

One of the important aspects of linking is to maintain consistency of information between structured and unstructured data. Prototype provides a feature where the changes to the linked data is detected and clearly highlighted to the users. Detection of changes in unstructured data straight forward, as the data stored in the link is compared directly with the content of the corresponding linked *span*. For structured data, changes are detected by comparing the exact value of attributes in the corresponding linked entities (can be same or different entity). We have to manually handle the changes in a tool where the contents of the wiki pages can be changed.

6.3. Future work

Prototypical implementation of the thesis enables users to import a hybrid wiki page, create and delete links and detect changes to the linked contents. Providing a way to edit the contents of the wiki will come in handy when we want to change the wiki contents. In the current implementation we have to handle the changes to the linked data manually using 3rd party tools. Edit feature can overcome this and helps maintain the consistency of data easily.

As linked data in the unstructured content is surrounded by a *span* tag, cross element linking is not straightforward. Selected content is detected by the range functionality of the JavaScript and it will not detect the ranges for data that span across multiple HTML elements, which makes the creation of links for contents that span across multiple HTML elements challenging. Another possible enhancement would be to over come this problem and enable cross HTML element linking, which can be further extended by allowing the creation of overlapping links.

Finally, a great enhancement would be to provide a recommendation feature which processes the structured and unstructured data and compute a list of possible related data that can be linked together. Allowing automatic creation of links based on the computed list would make a nice recommender system.

Appendix

A. Prototype Installation and QuickStart guide

In this chapter, procedure to install the prototype along with all the dependencies are explained in detail.

A.1. Prototype Dependencies

A.1.1. NodeJS

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code server-side. Download the Node.js pre-built installer for your platform, from the Node.js website and install it on the system. Verify that you are running at least Node.js version 8.x or greater by running the below listed command in a terminal/console window. Older versions produce errors, but newer versions are fine.

```
1 node -v
```

Listing A.1: Installed Node version

A.1.2. Node Package Manager (npm)

npm is a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js. It consists of a command line client, also called npm, and an online database of public and paid-for private packages, called the npm registry. The registry is accessed via the client, and the available packages can be browsed and searched via the npm website. The package manager and the registry are managed by npm, Inc. npm is usually bundled with Node.js installer which can be downloaded and installed as explained in Section A.1.1. Verify that you are running at least npm version 5.x or greater by running the below listed command in a terminal/console window. Older versions produce errors, but newer versions are fine.

```
1 npm -v
```

Listing A.2: Installed npm version

A.1.3. Angular Command Line Interface (CLI)

Angular cli is a command line interface to scaffold and build angular apps using nodejs style (commonJs) modules. Not only it provides you scalable project structure, instead it handles all common tedious tasks for you out of the box. Install the Angular CLI globally using following command.

```
npm install -g @angular/cli
```

Listing A.3: Installed Angular CLI

A.2. Installing and Running the Prototype

A.2.1. Install prototype developer and production dependencies

Clone or download the source code from the prototype repository. Navigate to the project directory and install the node packages, which are the development and production dependencies of the prototype using following command. Once all the dependencies are installed the prototype is ready to be served as a web application.

```
cd StructuredUnstructuredLinker
npm install
```

Listing A.4: Install Node Packages

A.2.2. Serve the prototype

Go to the project directory and launch the server using following command. By default the application will be served in at http://localhost:4200/.

```
cd StructuredUnstructuredLinker
ng serve --open
```

Listing A.5: Install Node Packages

A.2.3. Deploy prototype application

For the simplest deployment, build for development and copy the output directory to a web server.

1. Start with the development build

   ```
   ng build
   ```

   Listing A.6: Build development version

2. Copy everything within the output folder (dist/ by default) to a folder on the server.

3. If you copy the files into a server sub-folder, append the build flag, –base-href and set the ¡base href¿ appropriately. For example, if the index.html is on the server at /my/app/index.html, set the base href to <base href="/my/app/"> like this. You'll see that the <base href> is set properly in the generated dist/index.html. If you copy to the server's root directory, omit this step and leave the <base href> alone.

   ```
   ng build --base-href=/my/app/
   ```

   Listing A.7: Append Build Flag

4. Configure the server to redirect requests for missing files to index.html.

Bibliography

[1] Angular architecture. https://angular.io/guide/architecture.

[2] Angular framework. https://en.wikipedia.org/wiki/Angular_(application_platform).

[3] Annotation format. http://docs.annotatorjs.org/en/v1.2.x/annotation-format.html.

[4] Annotator library. http://annotatorjs.org/.

[5] Bootstrap examples. https://getbootstrap.com/docs/4.0/examples/.

[6] Bootstrap grid layout. https://getbootstrap.com/docs/4.0/layout/grid/.

[7] Bootstrap library. https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework).

[8] Hybrid wiki. https://wwwmatthes.in.tum.de/pages/1xy6w6pb8rf9j/Hybrid-Wikis.

[9] Json-ld. https://www.w3.org/TR/json-ld/.

[10] Json-ld design goals. https://www.w3.org/TR/json-ld/#design-goals-and-rationale.

[11] Rdf - semantic web standard. https://www.w3.org/RDF/.

[12] Semantic wiki. https://en.wikipedia.org/wiki/Semantic_wiki.

[13] Sociocortex. https://http://sociocortex.com/.

[14] Sociocortex: A social information hub. http://sebischair.github.io/sociocortex_web/files/160209%20Michel%20SocioCortex%20Eco-System.pdf.

[15] Sociocortex api. http://www.sociocortex.com/rest-api-tutorial/#_updating_entities_and_entity_types.

[16] Sociocortex architecture. https://wwwmatthes.in.tum.de/pages/13uzffgwlh8z4/SocioCortex-Model-Based-Collaboration-Environment.

[17] Unstructured information. https://en.wikipedia.org/wiki/Unstructured_data.

[18] Web annotation data model. https://www.w3.org/TR/annotation-model/.

[19] W. Cohen and S. Sarawagi. "Exploiting dictionaries in named entity extraction: Combining semi-markov extraction processes and data integration methods.". *SIGKDD*, 2004.

[20] Eric Prud'Hommeaux Jose Kahan, Marja-Riitta Koivunen and Ralph R. Swick. "Annotea: An Open RDF Infrastructure for Shared Web Annotations". *WWW10 International Conference, Hong Kong*, 2001.

[21] C. Neubert Matthes, F. and A. Steinhoff (2011). "Hybrid Wikis: Empowering Users to Collaboratively Structure Information.". *ICSOFT (1) 11*, pp. 250.

[22] Tim O'reilly. *What is Web 2.0: Design patterns and business models for the next generation of software.* 2007.

[23] M. Bhat A. Hernandez-Mendez Reschenhofer, T. and F. Matthes. "Lessons learned in aligning data and model evolution in collaborative information systems.". *Proceedings of the 38th International Conference on Software Engineering Companion*, ACM, pp. 132.

[24] Byron Dom Soumen Chakrabarti and Piotr Indyk. "Enhanced hypertext categorization using hyperlinks". *ACM SIGMOD Volume 27 Issue 2*, 1998.

[25] Prasan Roy Venkatesan T. Chakaravarthy, Himanshu Gupta and Mukesh Mohania. "Efficiently Linking Text Documents with Relevant Structured Information". 2006.