Summary: Within the scope of work package 2, two web services have been developed which allows semantic enrichment and disambiguation of scientific articles as well as mind maps. These services have been named Enrichment Service and Disambiguation Service. In the case of scientific articles, which are typically stored as PDF documents, the documents are first parsed, analysed and extracted. Here the PDFs are split into their structure, which allows us to re-create the original table of contents. Furthermore a set of metadata, for example the author names, are extracted. The algorithms, which have been developed to build the Enrichment Service, have been evaluated against existing or newly created data-set and the results have been published. The task of the Disambiguation Service is to disambiguate entities discovered by the Enrichment Service as well as disambiguate table columns. The results of this work have been published.
Commercially Empowered
Linked Open Data Ecosystems in Research

D2.2 – Refined Semantic Enrichment & Integration Workflow, Date: 2013-12-30

Revision | final
--- | ---
Authors | Roman Kern, Stefan Zwicklbauer, Michael Granitzer

Consortium:

Statement of originality: This document contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

This document reflects only the author’s views and the European Community is not liable for any use that might be made of the information contained herein. © CODE Consortium, 2012
Project Officer & Project Coordinators

<table>
<thead>
<tr>
<th>Project Officer</th>
<th>Stefano Bertolo</th>
</tr>
</thead>
<tbody>
<tr>
<td>European Commission</td>
<td></td>
</tr>
<tr>
<td>DG CONNECT Unit G3</td>
<td></td>
</tr>
<tr>
<td>EUROFORUM 01/293 - 10 rue Robert Stumper, L-2557 Luxembourg</td>
<td></td>
</tr>
<tr>
<td><a href="mailto:stefano.bertolo@ec.europa.eu">stefano.bertolo@ec.europa.eu</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project Coordinator</th>
<th>Stefanie Lindstaedt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Know-Center GmbH</td>
<td></td>
</tr>
<tr>
<td>Inffeldgasse 13, 8010 Graz, Austria</td>
<td></td>
</tr>
<tr>
<td>+43 316 873-30800 (phone)</td>
<td></td>
</tr>
<tr>
<td>+43 316 873-1030800 (fax)</td>
<td></td>
</tr>
<tr>
<td><a href="mailto:slind@know-center.at">slind@know-center.at</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scientific Coordinator</th>
<th>Michael Granitzer</th>
</tr>
</thead>
<tbody>
<tr>
<td>University of Passau</td>
<td></td>
</tr>
<tr>
<td>Innstrasse 33a, D-94032 Passau</td>
<td></td>
</tr>
<tr>
<td>+49(0)851-509-3305</td>
<td></td>
</tr>
<tr>
<td><a href="mailto:michael.granitzer@uni-passau.de">michael.granitzer@uni-passau.de</a></td>
<td></td>
</tr>
</tbody>
</table>

Document Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Author</th>
<th>Organization</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Draft</td>
<td>2013-12-17</td>
<td>Roman Kern</td>
<td>Know-Center</td>
<td>Initial Version</td>
</tr>
<tr>
<td>2nd Draft</td>
<td>2013-12-18</td>
<td>Stefan Zwicklbauer</td>
<td>Know-Center</td>
<td>Disambiguation Service</td>
</tr>
<tr>
<td>3rd Draft</td>
<td>2013-12-21</td>
<td>Roman Kern</td>
<td>Know-Center</td>
<td>Content completed</td>
</tr>
<tr>
<td>Final Version</td>
<td>2013-12-30</td>
<td>Michael Granitzer</td>
<td>Know-Center</td>
<td>Proof reading</td>
</tr>
</tbody>
</table>
Table of Contents

1 Introduction - 4 -
   OVERVIEW OF THE MAIN CHANGES, ADDITIONS AND REFINEMENTS STEPS - 4 -
   TOOLS - 5 -
   SOURCE CODE ACCESS - 5 -

2 Use Case and Requirements - 6 -
   UC1 TERM DEFINITION - 6 -
   UC2 CONTENT SUMMARISATION THROUGH FIGURES AND TABLES - 6 -
   UC3 ARTICLE NAVIGATION AND SUMMARISATION THROUGH STRUCTURE - 6 -
   UC4 ARTICLE NAVIGATION AND SUMMARISATION THROUGH ENTITIES - 6 -
   UC5 ENTITY SEARCH - 6 -
   UC6 NEW ENTITY CLASS CREATION - 7 -
   UC7 ENTITY MARKETPLACE - 7 -
   UC8 STRUCTURED DATA EXTRACTION FROM TABLE - 7 -
   UC9 CREATE MINDMAP PRESENTATION OF ARTICLE - 7 -
   UC10 ARTICLE NAVIGATION THROUGH REFERENCES - 7 -

3 Architecture & Tasks - 9 -
   ENRICHMENT SERVICE - 9 -
   CONCEPTUAL SOFTWARE ARCHITECTURE - 10 -
   EXECUTION SOFTWARE ARCHITECTURE - 11 -
   IMPLEMENTATION SOFTWARE ARCHITECTURE - 12 -
   DISAMBIGUATION SERVICE - 13 -

3 APIs and Interfaces - 14 -
   ENRICHMENT SERVICE - 14 -
   ENDPOINT: ASYNCHRONOUS PDF ENRICHMENT - 14 -
   DISAMBIGUATION SERVICE - 19 -
   DISAMBIGUATION LOOKUP SERVICE - 19 -
   ENDPOINT: DISAMBIGUATION USER FEEDBACK SERVICE - 21 -
   ENDPOINT: RDF SUMMARIZATION SERVICE - 22 -
   ENDPOINT: TABLE DISAMBIGUATION SERVICE - 23 -
   ENDPOINT: CATEGORY SUGGESTION - 24 -

4 Developed Methods and Algorithms - 25 -
   ENRICHMENT SERVICE - 25 -
   DISAMBIGUATION SERVICE - 25 -
   IMPROVEMENTS OF THE DISAMBIGUATION BY LEARN TO RANK - 25 -
   DISAMBIGUATION OF WEB TABLES - 26 -
   PROBLEM SETTING - 26 -
   APPROACH - 27 -
   EVALUATION - 27 -

5 Showcase Application - 28 -
   MOCKUPS - 28 -
   DEVELOPMENT - 29 -

6 References - 32 -
1 Introduction

An important aspect of the CODE project is the extraction of named entities and factual information. It is necessary for this task to semantically annotate and analyse textual resources. These resources might be scientific articles, as in the case for papers managed by Mendeley, or Mindmaps as in the case of Meisterlabs. For scientific articles, which are predominantly stored as PDF files, it is necessary to first extract its content.

The outcome of the work on the semantic enrichment and integration are two web services, the so-called Enrichment Service (KC as Lead) and Disambiguation Service (UNI Passau as Lead). In addition Mendeley provided a custom build Mendeley Client (Mendeley as Lead) to make use of the automatic annotations and provide facilities to allow manual annotations.

The services provide an API to allow scientific articles (as PDFs) and Mindmaps (as JSON) to be semantically annotated. These documents represent the input for the service together with additional meta-data for the resources. In the case of the Disambiguation Service the input is comprised of the context of a named entity candidate. The output of all services is then given in JSON format to allow the clients to easily parse the result. In the development of the service much effort has been spent to achieve a high degree of scalability to allow the services to cope with the amount of data as foreseen. To keep the effort minimal to integrate the services, the Enrichment Service does foresee mechanisms to automatically trigger the invocation of the Disambiguation Service. Thus a single API call allows the complete annotation and disambiguation.

In Phase I of the project we developed initial versions of these services (see D2.1.2 and D2.1.3) and agreed on a number of workflows (see D2.1.1). In the past phase we focused on implementing the remaining use cases and refining the algorithms. In addition we worked on a show-case application to demonstrate the functionality of the semantic enrichment algorithms and to allow users to play around with the computed models. In addition the show-case application highlights the integration of the participating partners by employing their tools.

Overview of the main changes, additions and refinements steps

Main changes of the Enrichment Service:

1. Improvement of the Information Extraction algorithms with a focus on the Computer Science domain
2. Improvements in the Table Extraction algorithms with a focus on the coverage
3. Addition of a Reference Extraction algorithms that goes beyond state-of-the-art in this field
4. Improvements in the API specification by integrating JSON-LD and the NLP Interchange Format (NIF) [HLAB13]
5. Evaluation of the aforementioned algorithms

Main changes of the Disambiguation Service:

1. Improvements in the disambiguation algorithms with a focus on performance
2. Addition of the disambiguation of table cells
3. Evaluation of the aforementioned algorithms

Development of a web-based show-case application for the work conducted in work package 2 in the year 2 of the project.
Tools
In addition to the two web services, accompanying tools and demonstration applications have been developed. Those tools are targeted not at the general public, but more on researchers and serve as a test bed for our algorithms.

Show-Case Application:
Web-based application that demonstrates the complete workflow to annotate documents, train models and then to apply these models. The tentative URL for the service is:
http://knowminer.at:8080/code-showcase/

Demo Application:
Web-based application that highlights the PDF extraction pipeline, including table extraction and reference extraction.
http://knowminer.at:8080/code-demo/

Source Code Access
Both services are open-source and can be accessed via public repositories.

Enrichment Service:
https://www.knowminer.at/svn/opensource/projects/code/trunk
(username: anonymous, empty password)

Disambiguation Service:
https://github.com/Quhfus/DoSeR
2 Use Case and Requirements

Overview of the use cases for the enrichment algorithms and the derived requirements. The individual use cases are described in detail in D2.1.1. To keep this document self-contained, a short overview of the use case workflows together with their current state of integration is given here. The workflows are considered as fully implemented if the functionality is available through the services and exposed by either the custom build Mendeley client or the Meisterlabs client application.

UC1 Term Definition

**Description:** Researcher is reading an article and wants to look up the definition of a term.

**Workflow:** The user opens a scientific article in the Mendeley client, selects a text and gets a list of definitions for the term. In the background the Mendeley client invokes the Disambiguation Service to look-up the definition of the term.

**Status:** This use case is fully implemented.

**Work done in Phase II:** Refinement of the workflow by improving the disambiguation algorithms, working on the efficiency of the algorithms and adding the category information to the result to allow the client to suppress unwanted categories.

UC2 Content Summarisation through Figures and Tables

**Description:** Researchers uses figures and tables for article navigation and summarization.

**Workflow:** The user opens a scientific article in the Mendeley client. In the background the Enrichment Service is invoked to extract the information from the article. Alternatively the scientific article has already been processed by a batch process and thus only the result of the batch processing needs to be retrieved.

**Status:** This use case is fully implemented.

**Work done in Phase II:** Refinement of the Enrichment Service algorithm, in particular in regard to the extraction of tables.

UC3 Article Navigation and Summarisation through Structure

**Description:** Researcher uses article structure for article navigation and summarisation.

**Workflow:** The user opens a scientific article in the Mendeley client. In the background the extracted structure of the scientific article is read out from a list of precomputed articles, or alternatively the information is extracted on the fly.

**Status:** This use case is fully implemented.

**Work done in Phase II:** Improvements of the extraction of heading lead to improved quality of the structure information.

UC4 Article Navigation and Summarisation through Entities

**Description:** Researcher uses entities for article navigation and summarisation.

**Status:** Has not been followed further as it has been found that from a user experience point of view little benefit can be achieved over the already supported by UC3.

UC5 Entity Search

**Description:** Researcher wants to search for papers with particular entities (e.g. search for an algorithm to see how it performs on a dataset).
**Workflow:** A user reads a scientific article, finds an algorithm's name (or any other named entity) and wants to read another paper using this algorithm. This workflow is based upon the output of the Enrichment Service and depends on the search infrastructure of the client’s backend.

**Status:** This use case is fully implemented

**Work done in Phase II:** Much effort has been put into improving the entities found by the Enrichment Service, in particular the Computer Science domain. Here we focused in particular on names of data sets and algorithms.

**UC7 New Entity Class Creation**

**Description:** Researcher wants to create a new class of entities and markup a set of articles with new entities.

**Workflow:** Allow the user to actively create own models and to annotate new instances.

**Status:** Implemented on a prototype state level

**Work done in Phase II:** Started working on a web-based show-case application to highlight the algorithms developed in work package 2 and to demonstrate the user case of the creation of new entity classes.

**UC8 Entity Marketplace**

**Description:** Researcher wants to sell their models to others

**Workflow:** Based on the use case “New Entity Class Creation” a user defines her own models and selects the distribution method

**Status:** Interviews conducted by Mendeley have indicated little interest to support this use case specifically within work package 2. Therefore this use case has not been followed further.

**UC9 Structured Data Extraction from Table**

**Description:** Researcher wants to visualize tabular data from an article in a graph.

**Workflow:** A user opens a scientific article, clicks on a table and in the background the table gets extracted and analysed and finally displayed to the user.

**Involved steps:** i) parse the PDF, ii) identify the table region, iii) decompose the structure of the table (e.g. columns, rows, cells, ...), iv) disambiguate columns, v) transform to data cubes, vi) visualise the data cubes

**Status:** i) – iii) are part of the Enrichment Service and the performance of the related algorithms is iteratively refined, iv) is part of the Disambiguation Service, has been started in year 2, v) is the responsibility of work package 3 and reported there and vi) is reported by work package 4.

**UC10 Create Mindmap Presentation of Article**

**Description:** Researcher wants to present the contents of a new article that they have published

**Workflow:** A user wants to create a Mindmap out of a scientific article, next she wants to use the presentation mode of Mindmeister.

**Status:** The Enrichment Service provides API calls specifically to generate the base for a Mindmap.

**UC11 Article Navigation through References**

**Description:** Researcher uses reference list for article navigation.

**Workflow:** A user opens a scientific article, while in the background the references of the article gets extracted and they are then displayed to the user.
**Status:** The state-of-the-art in reference extraction has been implemented and improved by integrating feature from our PDF extraction pipeline. The work has been conducted in the phase II and will be further refined. The references are then exposed via the service API.
3 Architecture & Tasks

The main work on the architecture has been conducted in Phase I, but is included in this document to keep the document self-contained. There have been little need to adapt the basic architecture, just the definition of the inputs and outputs have been adapted. The motivation for these adoptions where not due to technical limitations or requirements, but due to i) feature requests and ii) feedback from the review.

The main changes were the integration of the JSON-LD syntax to the data provided by the web-services, which just affects the API of the Enrichment Service, while the API of Disambiguation Service remained compatible by just adding information, like the category information, which is now additionally provided by the service.

The figure gives a quick overview of the three main components of one of the defined workflows. It should be noted that this just depicts one way to use the services, as they have been developed to be used either in conjunction with each other or separately.

**Enrichment Service**

This section gives an overview of the architecture of the Enrichment Service. The architecture is presented from three different views, namely the conceptual view, the execution view and the implementation view.
Conceptual Software Architecture

The conceptual software architecture describes the domain model of the system and represents the initial software architecture. The conceptual architecture is directly derived from the requirements. The connections between the components indicate an information flow, which is usually directed. Therefore cycles between components need to be avoided.

Conceptual Software Architecture View

Domain model components:

- **PDF parsing and rendering library**: serves as a basis for all further processing steps
- **Layout extraction**: uses the output of the PDF parsing and transforms them into blocks, which represent the individual layout
- **Role extraction**: extracts table of contents, headings, captions, decoration, main text and metadata from the scientific article
- **Relation Extraction**: extracts the relationship between the different items within the article, for example the reading order
- **Table Extraction**: extracts the tables and their content in a structured way from a scientific article
- **Domain Classification**: uses the input as a basis to decide to which domain the scientific article belongs to, e.g. biomedical domain or computer science
- **Entity Extraction**: extracts entities (especially named entities), for example names of genomes, proteins or names of data-sets, from the main text of the article,
- **Fact Extraction**: puts the extracted entities in relation to each other to form triples for further processing
- **Model Storage**: is a component to manage the different models of domains or individual users

Execution Software Architecture

The execution software architecture focuses on the runtime of the components, especially on the sequence of the processing and on concurrency issues. Components in the execution software architecture are separate machines, processes and threads.

The execution software architecture of the Enrichment Service has been developed to cope with a high amount of data. Therefore the scalability of the software architecture has been given a high priority. In the runtime architecture, two main concepts have been followed to achieve the desired level of scalability:

- **Scale out**: Horizontal scaling by making it possible to dedicate more machines, in order to cope with a higher demand. Therefore the workload needs to be modelled in a way to allow parallel execution.
- **Asynchronous calls**: By following an asynchronous approach, the available resources can be utilized more efficiently. To allow for asynchronous processing, all incoming request are first stored in a queuing system

![Runtime Software Architecture View](image)

The main components in the runtime software architecture are:

- **Web Service Frontend**: Handles incoming web service requests and stores them in the message queue.
- **Message Queue**: Is responsible to manage the list of outstanding requests and for keeping the stack of requests consistent
- **Workers**: Read out requests from the message queue and process individual requests. Here multiple requests can either be handled within a single machine or across multiple machines, as the individual requests should have no side effects on other requests
• **PDF Document Storage**: Provides an external source for PDF documents. This is needed as the original request did only contain a URL to the PDF, but not the PDF itself. The enrichment service itself does not provide means to store or index scientific articles.

• **Callback Web Service**: Provides the response for the individual requests and reports to another web service, which is specified by the original request

**Implementation Software Architecture**

The implementation software architecture focuses on how the system is build. Components are either domain components or components provided by the underlying system. The main components of this software architecture view are:

• **Web Application Server**: Runtime context of the Enrichment Service

• **RabbitMQ**: Messaging system, which provides a high-performance, persistent message queue

• **ZooKeeper**: Component to allow a centralised configuration for a whole cluster of machines and additionally provides synchronisation mechanisms

• **Enrichment Worker**: Runtime context of the individual workers, which run on separate machines within a cluster

---

**Implementation Software Architecture View**
Disambiguation Service

The semantic integration task focuses on integrating ontological concepts, in particular concepts available in the LOD cloud, into research papers or structures extracted from research papers. In order to do so we focus on two subtasks:

1. Disambiguation of Named Entities provided by either the Enrichment Service or the Mendeley/Meisterlabs clients
2. Disambiguation and type estimation of table columns

In order to estimate the accuracy and complexity, we started with evaluating the complexity and baseline accuracy of disambiguation tasks given the heterogeneous nature of the Linked Open Data cloud and the domain of research papers. After that, we investigated the influence of machine learning approaches (more specific Learning To Rank) on disambiguation algorithms by a set of experiments on special purpose knowledge bases from the biomedical domain:

- We show that Learning to Rank significantly improves accuracy with an entity-centric knowledge base but has no effect on document-centric knowledge bases
- We show that by appropriately combining these two database types can improve the results
- We show that Learning to Rank does not prevent accuracy degeneration after increasing heterogeneity and size of a knowledge base
- We demonstrate the scalability by integrating our Learning to Rank system in Mendeley Desktop

Our results indicate that machine learning approaches may significantly improve disambiguation with entity-centric knowledge bases. However, due to a missing relation between the ranking task and correct entities in the algorithm, results could not be improved with a document-centric knowledge base. Additionally, our results show that by appropriately combining entity-centric and document-centric knowledge base results can be improved significantly. This includes not only the results of the measurements, but also the robustness against a strong increase of available documents and entities in the knowledge bases. In this context, we show that LTR does not prevent accuracy degeneration after increasing heterogeneity and size of an entity-centric knowledge base.

Our second task comprises the annotation of table headers with semantic type information based on the content of column’s cells. We achieved similar accuracy as previous work with more complex methods. Additionally, from our experiments it seems reasonable to use only a small number of cells for annotating the header (20 cells lead to 94% of the total achievable accuracy) if performance is an issue.

Technically all of our services have been tested and integrated on an API level by Task 2.1, Task 2.2 and WP 3.
3 APIs and Interfaces

Enrichment Service

In order to keep this document self-contained, some information already described in D2.1.2, while some information in regard to the API specifications are omitted here, as they did not change. The main API calls themselves did undergo only small, mostly cosmetic changes. The main change to the Enrichment Service in particular has been the integration of the JSON-LD format, as suggested in the review meeting. Additionally we make use of the NLP Interchange Format (NIF) to semantically expose the extracted information. Endpoint: Asynchronous PDF Enrichment

URL: http://external.know-center.tugraz.at:7070/code-server-1.0-SNAPSHOT/enrichment/upload

POST Request, Content-Type: application/json

Input: Document Metadata

Description: unique document URI, URL to fetch the PDF from, callback URL, priority flag, optionally other metadata like authors, publication date/venue.

Example:

```json
{
  "uri": "your-documents-uri-here",
  "pdfUrl": "http://url-to-pdf-on-s3",
  "callbackUrl": "http://url-to-receiving-end",
  "priority": "true",
  "metadata": {
    "authors": [
      "Max Muster",
      "William T. Kirk"
    ],
    "publication-date": "19850412T232050"
  }
}
```

Process: Given the document metadata, the service will push the request into a queue ("isPriority" = "false", "isPriority" omitted) or process the document immediately ("isPriority" = "true"). Once processed, the service will push the output to the provided callback URL. If the callback endpoint isn’t available the results are discarded.

Output: JSON Object, status code 204

Description: a JSON Object containing named entities, facts and the table of content of the document.
Example: the root Object contains parts of the metadata (document ID etc.) and two additional members storing the named entities and facts as an array, and table of content as a nested JSON object.

```json
{
    "annotations" : [ /* annotations */ ],
    "@type" : [ "nif:RFC5147String", "nif:Context" ],
    "nif:isString" : "Fulltext of the article",
    "nif:beginIndex" : 0,
    "nif:endIndex" : 24947,
    "documentUri" : null,
    "toc" : { /* table of contents */ }
}
```

- There are two types of annotations: named entities and facts.
- A named entity looks like this:

```json
{
    "type" : "NamedEntity",
    "id" : "0",
    "@type" : [ "nif:RFC5147String" ],
    "text" : "formalin",
    "end" : 7277,
    "uri" : "http://dbpedia.org/resource/Formality",
    "positions" : [ {
        "type" : "PdfPosition",
        "offsets" : [ 7270, 7271, 7272, 7273, 7274, 7275, 7276, 7277 ],
        "pageId" : 2
    }, ],
    "description" : "A formality is an [...].",
    "begin" : 7270,
    "categories" : [ { "url" : "http://dbpedia.org/resource/Category:Cultural_conventions", "label" : "Cultural conventions" } ],
    "normalized" : "Formality",
    "confidence" : 1.0,
    "classLabel" : "Chemical Compound",
    "nif:referenceContext" : null
}
```

Each entity has the following attributes:
- text: the characters as found in the document
• normalized: the normalized label for the entity, may be null
• uri: the (disambiguated) instance uri for the entity, may be null
• confidence: confidence in classification and disambiguation
• classLabel: the name of the entity class
• positions: the positions and bounding boxes of individual parts of the annotated text. An entity can consist of multiple parts if it extends over multiple lines. Each position object specifies
  o pageld: the page the part of the named entity was found on, starting at page 0.
  o offsets: the character offset relative to the document start.
  o boundingBox: the 2D bounding box within the pdf of that part of the entity.
  measurements are given in percentages relative to the page width and height.

A fact looks like this:

```json
{
  "type": "fact",
  "uri": "uri-of-fact",
  "confidence": 0.83,
  "subject": {
    "text": "SVM",
    "normalized": "Support Vector Machine",
    "confidence": 0.83,
    "classLabel": "algo",
    "positions": [...]
  },
  "predicate": {
    "text": "is a",
    "uri": "http://somerdfvocabulary.org/isA",
    "confidence": 0.83,
    "classLabel": "relation",
    "positions": [...]
  },
  "object": {
    "text": "supervised classifier",
    "normalized": "Supervised Classification",
    "uri": "http://en.dbpedia.org/page/Supervised_Classification",
    "confidence": 0.83,
    "classLabel": "algo",
    "positions": [...]
  }
}
```

Each fact is a subject/predicate/object triple plus additional information:
• confidence: the confidence that this fact is correct
• subject: a named entity, same fields as an isolated named entity.
• predicate: predicate linking to concept in a defined vocabulary defining the relation.
• object: a named entity or a literal. In the latter case no URI will be provided.

A table of contents looks like this:
“elements”: [ {
  "type": ".Title",
  "id": 0,
  "label": "Nestin and CD133: valuable stem cell-specific markers for determining clinical outcome of glioma patients",
  "positions": [ {
    "type": ".PdfPosition",
    "boundingBox": { 
      "minx": 54.14,
      "miny": 676.32,
      "maxx": 485.09036,
      "maxy": 687.492
    },
    "pageId": 1
  },
  {
    "type": ".PdfPosition",
    "offsets": [ 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 149, 150, 151, 152, 153, 154, 155, 156, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 170, 171, 172, 173, 174, 176, 177, 178, 179, 180, 181, 182, 183 ],
    "boundingBox": { 
      "minx": 54.140015,
      "miny": 658.3203,
      "maxx": 410.83643,
      "maxy": 669.4923
    },
    "pageId": 1
  } ],
  "annotationIds": [ ]
}, {
  "type": ".Section",
  "children": [ ],
  "id": 1,
  "label": "Background",
  "positions": [ {
    "type": ".PdfPosition",
    "offsets": [ 2675, 2676, 2677, 2678, 2679, 2680, 2681, 2682, 2683, 2684 ],
    "boundingBox": { 
      "minx": 54.14,
      "miny": 95.34009,
      "maxx": 113.81788,
      "maxy": 102.85809
    },
    "pageId": 1
  } ]}
The response specifies the URI of the input document as well as the table of contents, encoded in the "toc" field. The "toc" field is an array of objects, each having a type specified via a "type" attribute. The following types are available:

- "Title": the title of the publication.
- "Abstract": the abstract of the publication.
- "Section": a generic layouting element. Sections can be nested and contain other types, such as other sections, figures or tables. Nested elements are given in the "children" array attribute.
- "Figure": a figure, consisting of a label (=caption) and the binary data of the image, in whatever format it is found to be in in the PDF.
- "Table": a table, consisting of a label (=caption) as well as the actual data.

**Side-effects**

None

**Errors**

4XX Bad Request: metadata invalid, returns JSON object with an error message:

Example:

```
{
    "error": "The error message"
}
```

5XX Internal Server Error: for unforeseen errors while executing the request
Disambiguation Service
The following section describes the API of the developed prototypes including lookup, summarization of results and user feedback. The APIs are fully functional and already integrated with corresponding end-user facing interfaces.

Disambiguation Lookup Service
Figure 1 shows an example JSON Request

```
{
  "documentId": "unique document id",
  "surfaceFormsToDisambiguate": [ ...
    {
      "selectedText": "influenza",
      "context": "Typically, influenza is transmitted through the air by coughs or sneezes, creating aerosols containing the virus.",
      "position": {
        "pageId": 0,
        "offsets": [0, 1, 2, 3, 4, 5, 6, 7],
        "boundingBox": [0.1, 0.2, 0.3, 0.4]
      }
    },
    { ...
  }
  "alreadyDisambiguatedEntities": [ ...
    {
      "text": "illness",
      "entities": [{
        "uri": "http://en.wikipedia.org/page/Illness",
        "confidence": 0.90
      }]
    },
    { ...
  }
}
```

Figure 1: JSON Disambiguation Lookup Service Overall

**Input: URI (required)**
- Description: The URI of the PDF document from Mendeley

**Input: Surface forms to disambiguate**
- Description: An array of multiple surface forms within one document which should be disambiguated
  - Selected Text (required): The user is able to select some text in the document. The selected text should be disambiguated.
  - Context (required): The context of the user selected text described above. This might be the surrounding sentences.
  - Position (required): An array of positions defining the exact positions of this surface form in a document. A surface form may consist of multiple parts if it extends over multiple lines. Each position object specifies:
    - pageld: the page the part of the named entity was found on, starting at page 0
    - offsets: the character offset relative to the document start
    - boundingBox: the 2D bounding box within the pdf of that part of the entity. Measurements are given in percentages relative to the page width and height

**Input: Already disambiguated Entities (optional)**
• Description: A list of other disambiguated and user approved entities will be delivered. Distance indicates the distance in characters to the location of the current entity/annotation.
  • Text - Surface form
  • EntityUri - The URI of the disambiguated entity
  • Confidence - The probability of the entity being correct
  • Distance - Distance of this surface form to the current surface form which should be disambiguated

Process: The disambiguation service tries to disambiguate the selected text and creates a ranked entity list with most probable entities first. Therefore we take a look at our data store to check existing data about the selected text. If not a WP3 callup will crawl for information.

Output: Ranked entity list
• Description: All surface forms which should be disambiguated are retrieved. Every disambiguated surface form contains a ranked entity list which is sorted by descending order. The position property is used to provide a correct assignment of the results to the surface forms in a PDF document.
• An example is shown in figure 2.

```json
{
  "documentUri": "unique document id",
  "disambiguatedSurfaceforms": [
    {
      "selectedText": "influenza",
      "position": {
        "pageId": 8,
        "offsets": [1, 2, 3, 4, 5, 6, 7],
        "boundingBox": {"minx": 0.1, "miny": 0.3, "maxx": 0.61, "maxy": 0.83}
      },
      "disEntities": [
        {
          "text": "Influenza (Illness)"
          "entityUri": "http://en.dbpedia.org/pages/..."
          "confidence": 0.80
          "description": "some additional description"
        }
      ]
    }
  ]
}
```

Figure 2: JSON Disambiguation Response

Side-affects
The disambiguation process returns a ranked entity list. Depending on the amount of results which are delivered by WP3, the list might be empty.

**External Dependencies**
- The number of returned hits depends on the API configuration. Adding an additional execution parameter would be useful. The default value should be between 10 to 20 items.

Endpoint: Disambiguation User Feedback Service
The user might choose an item of the ranked entity list which is correct (but is not needed to be correct). After evaluating at least one entity the user is able to send an evaluation request to our disambiguation service which uses these information for “learn to rank” process.

**Input: URI (required)**
- Description: The URI of the PDF document from Mendeley

**Input: Selected Text (required)**
- Description: The user is able to select some text in the document. The selected text should be disambiguated
- Example: SVM

**Input: Feedback items (required)**
- Description: A bunch of feedback items which are assigned to different surface form in the pdf document
  - Entity Url: The URL of the disambiguated entity
  - Selected Text: The surface form which was disambiguated
  - Position: The position is required due to the unknown entry in the disambiguation database. It is utilized as a primary key. More details can be found above. Due to possible multiple parts if the surface forms extends over multiple lines a position array is used.
  - Type of feedback: For each entity in the list, the feedback from the user belongs to one of the four types: (i) accept (correct suggestion), (ii) reject (wrong suggestion), (iii) no feedback (either no viewed or not able to decide) or (iv) a new concept is added. Thus, the feedback type will be represented by integer values. Value 1 for correct, value -1 for wrong. A new concept is represented by the value 2. Entities with no decision do not occur in the returned list.
Process: The user feedback is required to adapt the learn to rank process to improve the results in the future lookup services.

Output: None

External Dependencies
- The user is not required to give a feedback. Additionally the user feedback might be wrong.

Endpoint: RDF Summarization Service

Input: Some URI's of RDF resources which should be summarized
- Description: A List of URI's of an arbitrary RDF resources, which should be summarized

Process: The RDF Summarization Service tries to create the best summarization of a RDF resource

Output: List of summarizations Figure 4 shows a RDF Summarization Response in JSON format.
- Description: Each response item contains a human readable summarization of the RDF resource together with its representing label and its types.
  - URI: The uri of the respective resource
  - Label: The label of the respective resource
  - Types: Array of different types
  - Summary: Summary of the RDF resource

Figure : JSON Disambiguation Feedback
Endpoint: Table Disambiguation Service

The table disambiguation service provides a rest interface for disambiguating table headers.

**Input:** Document ID (required)  
Figure 5 shows an example of a table disambiguation request.
- Description: The unambiguous ID of a document, which contains the table

**Input:** Name of the table (optional)
- Description: The headline which describes the content of a table

**Input:** List of columns (required)
- Description: A list of columns which should be disambiguated
  - ColumnHeader (optional): The header of a column
  - CellContent: List of all Cells (required)

**Process:** The table disambiguation services disambiguates each cell separately and searches its relevant types in the Linked Open Data cloud. The algorithms constitutes a knn-classification with majority vote.

**Output:** Ranked type list  
Figure 6 shows an example of a table disambiguation response.
- Description: The results of the disambiguated columns are retrieved. Please note that the length of the resultlist of each column is constrained to 10.
Figure 6: JSON RDF Table Disambiguation Response

**Endpoint:** Category Suggestion

This endpoint suggests DBPedia categories after entering an incomplete string.

**Input:** Incomplete String (required)  Figure 7 shows an example of a request.

- Description: This may be any incomplete string. The engine tries to suggest relevant categories.

**Input:** Language (optional)

- Description: The language of the string. According to this language, appropriate categories will be shown.

```json
{
  "input": "Machine 1",
  "language": "en"
}
```

**Process:** The Category Suggestion Service searches relevant categories of the entered string. The optional language field is used for other languages than English language.

**Output:** Ranked category list  Figure 8 shows an example of a table disambiguation response.

- Description: Each category contains its Uri and Label
  - Uri: The uri of the category
  - Label: The label of the category

```json
{
  "categories": [
    {
      "label": "Machine Learning"
    },
    {
      "uri": "http://dbpedia.org/resource/Category:Machine_learning_algorithms",
      "label": "Machine Learning Algorithms"
    }
  ]
}
```

**Figure 8: JSON RDF Category Suggestion Response**
4 Developed Methods and Algorithms

Enrichment Service
In the deliverable D2.1.2 there has been given an outlook to the next development steps, of which the status is reported here:

- **State-of-the-art analysis**: have been conducted in all relevant areas, including PDF extraction techniques, as well as Named Entity Recognition approaches
- **Basic development dataset**: we have developed a number of data-sets, specifically in the domain of computer science as there does not exist sufficient data-sets, in contrast to the Biomed domain.
- **Table Extraction**: We improved the table extract algorithms, where we already published the main results
- **Hearst-like Patterns**: We employed a pattern based approach for extraction of grant information and investigated methods to manually create such pattern, where initial evaluations indicate, that their usefulness highly depends on the quality of the annotations
- **Evaluation & Dissemination**: We published all of our results at workshops, conferences and journals.
- **Seed-based Extraction**: This approach has been implemented in the show-case application, which also contains ways to automatically evaluate their performance.

In phase II of the project we focused on the aforementioned steps and on improving and refining the developed algorithms. We were able to successfully publish most of our work. Therefore the algorithms are not described in detail here, instead just the relevant publications are mentioned here and added as appendix to this document:


In regard to the reference extraction:

  
  http://www.dlib.org/dlib/september13/kern/09kern.html

Disambiguation Service

Improvements of the Disambiguation by Learn to Rank
A crucial factor when creating a disambiguation system is the type of the underlying knowledge base. The authors of [Mihalcea2007] compared disambiguation results achieved by using context information about already disambiguated entities in documents and encyclopaedic definitions of entities. In our work we refer to the former as document-centric and to the latter as entity-centric knowledge base. The results of Mihalcea et al. [MC07] show that the document-centric approach outperforms the entity-centric approach on general knowledge like Wikipedia. Similar results were attained on special purpose knowledge bases by Zwicklbauer et al. [ZSG13]. However, the question remains open
whether machine learning approaches (i.e. Learning to Rank) can overcome this deficit. In this context, it is unknown whether machine learning algorithms may prevent a significant decrease of accuracy after increasing the size and/or heterogeneity of a knowledge base. Additionally, it has not been investigated how results emerge after exploiting data of entity-centric and document-centric knowledge bases. In context of machine learning, it must be mentioned that supervised learning algorithms learn their model out of a ground truth annotation of a dataset. However, the results of such models also should be compared with a model created of feedback data from different users.

We tackled those problems and present a set of experiments on special purpose knowledge bases from the biomedical domain where we investigate disambiguation performance with the aid of Learning to Rank.

In terms of limitations, our LTR system is constrained by standard information retrieval features that exclude domain-specific features due to needed domain experts. Additionally, we did not consider semantic relationships like taxonomies or part-of-relationships. Another constraint in our work is the non-consideration of other entities in the context area of the surface form to be disambiguated. However, our algorithms shows that disambiguation with LTR using standard features already attains suitable measured values.

Disambiguation of Web Tables
Annotating headers in (web-)tables has already been studied extensively whereby results mainly depend on the origin of tables [LCS10]. We propose an algorithm to annotate table headers with semantics based on the types of the column’s cells. In our experiments on 50 tables we achieved an $F_1$ value of 0.55, where the accuracy greatly varies depending on the used ontology. Moreover, we found that for 94% of maximal $F_1$ score only 20 cells (37%) need to be considered on average. Results suggest that for table disambiguation the choice of the ontology needs to be considered and the data input size can be reduced. We plan to exploit more relational knowledge (e.g. same-as) to further improve the annotations. The comprehensive evaluation of the experiments can be found in [BG13].

Problem Setting
Tables represent a large pool of factual data. However, for merging, analysing and combining tables from different sources we have to disambiguate the meaning of individual columns and cells. For example the column BMW, Mercedes and Renault could refer to car manufacturers or car types themselves. Given a column $C=c_1\ldots c_k$ with $k$ column values $c_i$ as string literals, we aim to

1. Identify the type of $C$, i.e. the class of all column values $c_i$. The type should be represented with a Semantic Web URI.
2. Given the identified type of $C$, disambiguate all corresponding column values (i.e. assign a Semantic Web URI)

Our approach is based on the assumption that a table contains entities of similar (super)-types.
Approach

Based on the developed disambiguation service we implemented the following workflow for identifying the types

1. For every column value $c_i \in C$, get all potential URL $U_i$ from the disambiguation service
2. For every candidate URI look up all type relationships (i.e. rdf:type) in the Semantic Web
3. Apply a ranking function on all types that returns a ranked list of types
4. Assign the highest-ranked $n$-types $T_{\text{selected}} = \text{argmax}(\text{rank}(C))$ with equal score as type to column $C$ and take the highest-ranked URI that has a type relationship to $T_{\text{selected}}$ as URI for $c_i$.

The algorithm is simple and can be conducted efficiently for small columns. For large columns, we aim to simply subsample the cell values to reduce the amount of lookups.

Evaluation

The evaluation of the service is currently in process, but first results seem promising (see [ZEGS13]). As ranking we took a simple counting function over types as well as a constrained based counting function, which means that only types are counted, that are assigned to all column values.

Our results show, that we can reliably identify column types and corresponding cell values. Moreover, the approach seems to be robust to failed disambiguation or type lookups. Then next step will be the integration of the table disambiguation service into the WP 3 data extractor.
5 Showcase Application

In order to demonstrate the algorithmic underpinning of the work conducted in work package 2 we decided to develop a demonstration application. This application should allow us to quickly integrate workflows and methods to illustrate our technologies.

Mockups

In the project meeting in London, on the 8th and 9th of July we presented a mock-up of the planned showcase application all the project partners to gather early feedback and to steer the development process. The mock-up is replicated here, where the basic workflow is:

- User logs into the system
- Selects one of the already installed data-sets or imports one from the individual Mendeley library
- Then the user may choose a model, or import one from Meisterlabs
- In the search interface the user may submit a query and select documents to annotate
- Finally the user may start to annotate the document with a selected model
Development

In Phase II we implemented the showcase application as planned, while refining our algorithmic underpinnings. During development we added a number of features which have not been included in the original mock-ups. Our development process has been accompanied by a Mindmap shared between the developers and a Scrum board, while employing techniques for agile software development.

The current state of the shared feature planning Mindmap on Mindmeister:

The current state of the showcase application is illustrated by the following screenshots that are picked as real-life counterparts to the mock up images:
Commercially Empowered Linked Open Data Ecosystems in Research

D2.2 – Refined Semantic Enrichment & Integration Workflow, Date: 2013-12-30
Annotate Document
Select one of the available modes for this data set. Once you have finished editing click the save button.

Save

© 2013 Know-Center GmbH
6 References


