Technical Report about EXCITE Data Conversion to OCC Ontology

1- Introduction:

Citation data are valuable since they show the linkage between efforts of different authors. Despite the fact, the open access to the data is still insufficient. Recently, some movements and projects focus on generating citation data openly. The aim of EXCITE project is the extraction of references out of PDFs, which will be done in three main steps:

1. Extraction of reference strings out of PDFs
2. Segmentation of each reference string
3. Match each bibliographical item against corresponding items in bibliographical databases

After this process, the output of EXCITE will be a set of reference strings and their Match information. The OC(open citations) project aimed to publish open bibliographic citation information in RDF and to make citation links as easy to traverse as Web links. The main deliverable of the project was the release of an open repository of scholarly citation data described using the SPAR Ontologies, and named the OpenCitations Corpus (OCC), which was initially populated with the citations from journal articles within the Open Access Subset of PubMed Central.

The code is part of EXCITE project and it is dedicated to the task of converting EXCITE Data to a JSON file with (OCC ontology). Besides that, these data will be enriched with metadata of records in Sowiport.org and SSOAR. Since a portion of references extracted has match information, we use this opportunity to extract metadata from corresponding records in these databases and add it to our data and then convert it to a JSON file.

The OCC includes information about six different kinds of bibliographic entities:

- Bibliographic resources (br): Cited/citing bibliographic resources
- Resource embodiments (re): Details about the bibliographic resources made (such as pages)
- Bibliographic entries (be): Reference strings
- Responsible agents (ra): names of agents having specific roles concerning bibliographic resources (i.e., names of authors, editors, publishers, etc.)
- Agent roles (ar): roles held by agents concerning bibliographic resources (e.g., author, editor, publisher)
- Identifiers (id): external identifiers (e.g. DOI, ORCID, PubMed ID) associated with the bibliographic entities.

2- Data corpora of EXCITE project:

2-1- Data corpora for processing:

The final product of EXCITE project would be a web service which user can submit a Pdf and receives the reference strings inside the Pdf and match information of them. But beside this web service, EXCITE aims to process some corpora that it owns already and publish all data produced. EXCITE contains 1- SSOAR corpus (~ 35 k papers), 2- Arxiv papers (more than 1

million), 3- SOJ – Springer Open journal corpus (~ 80 k papers), and 4- Sowiport papers (~ 116K)

2-2- Data corpora for enriching data:

2-2-1- The Social Science Open Access Repository (SSOAR):

The SSOAR repository provides full-text access to documents. It covers scholarly contributions related to different social science fields such as social psychology, communication sciences, and historical social research. Today, approx. 46,000 documents are available. This repository provides some metadata such as abstract and keywords for each article in both German and English. It is considered as a secondary publisher which publishes pre-prints, post-prints, and original publishers' versions of scholarly articles, but it also lets authors publish their work for the first time.

2-2-2- Sowiport - the online portal for the social sciences:

Sowiport is an online portal for social sciences. It contains bibliographical meta of more than 9 million references on publications and research projects. This corpus is used as the target set for EXCITE matching algorithms.

2-3- How do EXCITE data look like?

After processing an EXCITE corpus (e.g. SSOAR corpus), the data produced will be stored in Psql tables: 1- A table for storing reference strings extracted from PDFs, 2- A table for storing segmented reference strings, and finally 3- A table for storing match information of references.

For this task we join the first and the third table and only keep 4 columns:

1. ref_id: It is an auto-incremental number that by inserting a new reference string, it will be generated automatically and later on, for referring the reference strings, it will be used.
2. paper_id: It shows the information of the paper that contains a reference string (for example, if the paper is from SSOAR corpus, then it would be SSOAR ID)
3. ref_text: this column contains reference strings extracted from PDFs
4. Sowiport_match: This column shows the match information for each reference string. If there is a match for the reference string, then we have a sowiport id in the column for the reference string otherwise the value would be NaN (i.e. numpy.nan).

Afterward a new column would be added to the table, which is "ssoar_match". This column would be created based on metadata of Sowiport. If in sowiport match there is any sowiport id, then it will be checked that if based on the id, there is any ssoar id or not. If there is a ssoar id, then it will be inserted into "ssoar_match" column. Figure 1. shows EXCITE data example for this task from SSOAR corpus.

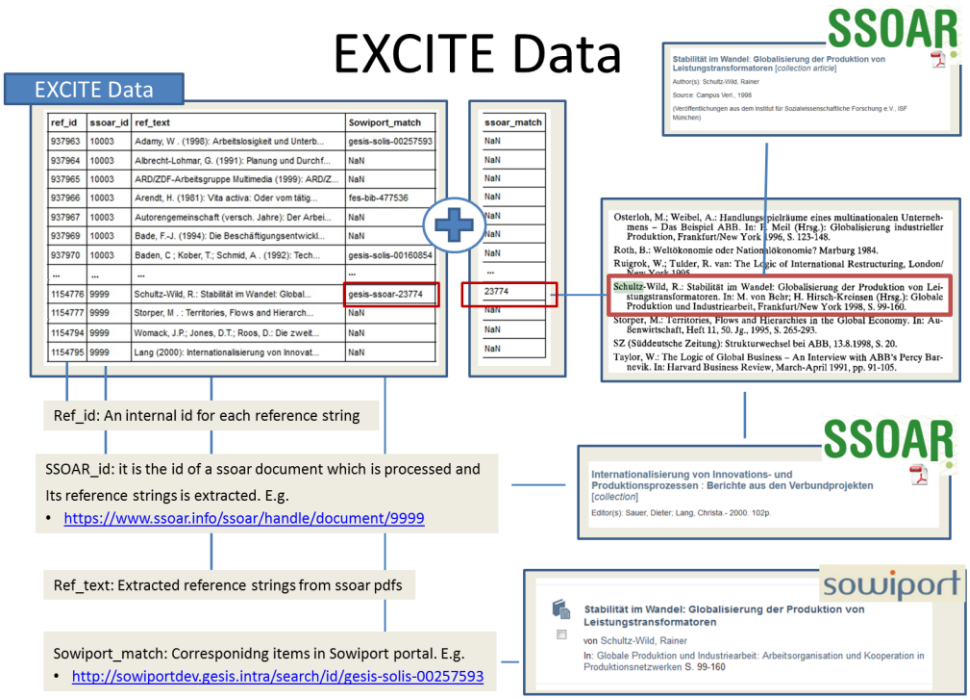

Figure 1. EXCITE data example for Data conversion task from SSOAR corpus.

3- Data Preprocessing:

Before starting and analyzing EXCITE data, we should apply some steps to pre-process data clean them:

1- For each reference string, we should compare “paper_id” column with “sowiport_match” and “ssoar_match”. If the comparison shows that the reference is referring to the paper which contains the reference, we should drop this record.
2- If a reference string repeats more than one time in a same paper, then we need to keep one and drop the rest.

4- EXCITE Data Conversion:

4-1- Create Output for Each Corpus:

EXCITE data before conversion into OCC ontology will be enriched by sowiport and ssoar metadata. The metadata comes from SSOAR has a higher priority than sowiport since they have a better quality. Therefore, we decided to have a separate module for data conversion of ssoar papers than other corpus. The module includes metadata of ssoar for the ssoar papers and also their reference strings will be enriched by ssoar metadata and sowiport metadata, if our algorithm could match them. In process of SSOAR data, we use:

```
from SSOAR_full_Corpus_new import Generate_Flatten_json_OCC_For_SSOAR
```

The function gets three inputs:

1- REF_SSOAR_file: It is the address of the CSV format of the Psql table (like Figure 1.) after preprocessing for the corpus, which we want to convert to OCC ontology.
2- Target_dir: The path for saving the output in JSON format (the default value is “./data“)

3- ssoar_harvest: If it is not zero, then it tries to harvest metadata of whole SSOAR and convert them into a python dictionary type to use it later on in the code for enriching data. We also need to set it not zero for the first time to generate "data_harvest1.json" in "support_data" directory.

In this function, we also call the following function:

```
from ssoar_sowi import sowi_ssoar_match
```

The function adds "ssoar_match" columns to the EXCITE Data as it is shown in the figure 1. Also, even if the self citations are not checked in preprocessing task, then in this code, it is considered and they will be cleaned by comparing "ssoar_id" (PDF) and "ssoar_match"(Reference). The code generates all brs for ssoar papers and their citation and enrich them by metadata from sowiport and SSOAR. The OCC format looks like:

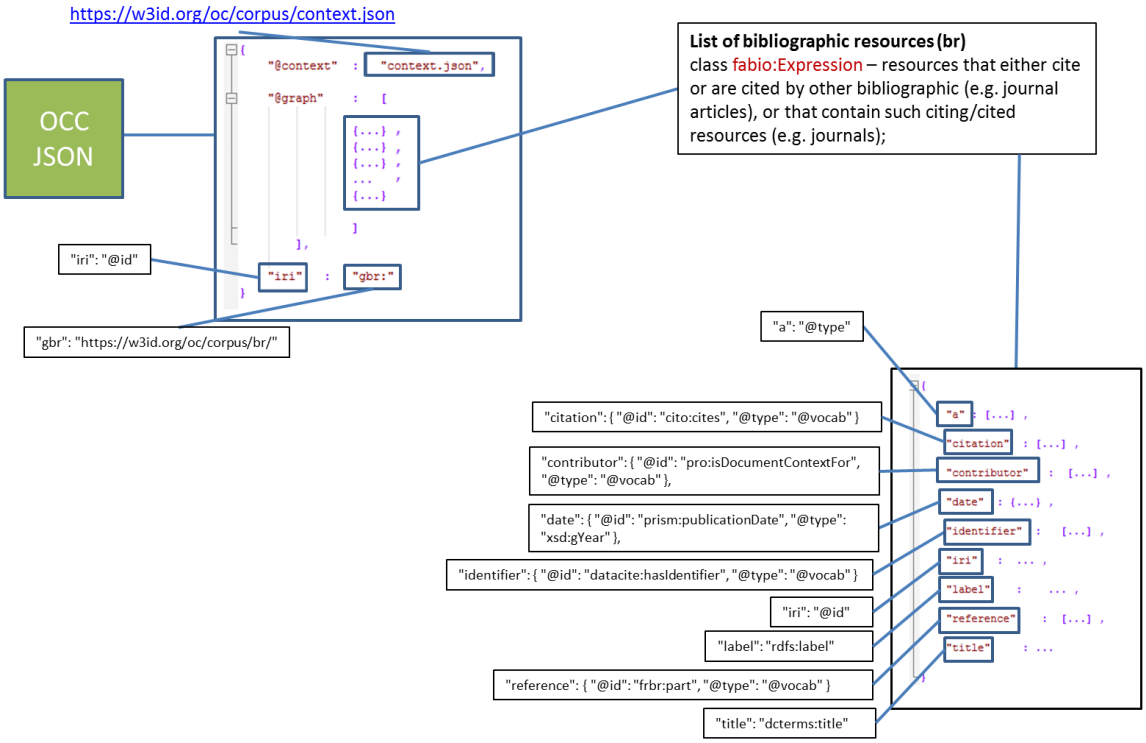

Figure 2. The structure of JSON file with OCC ontology, and definition of each field.

"iri" field is "@id" in context.json file. We use meaningful structures for creating the iri in the first step and then convert them to the format that OCC like to have the data. "iri" in OCC format should start with an identifier for the data producer (e.g. EXCITE project has 0110 as the prefix) and after this prefix there is an auto incremental digit. For presenting each corpus, we use a symbol: 1- SSOAR: 0000, 2- Arxiv: 0005, and 3- Sowiport: 0001.
There are only three categories. The reason is that first, for arxiv papers we don't have metadata in sowiport and ssoar, so it is categorized in one group. For SSOAR papers we have higher quality metadata in ssoar portal and for the rest of corpora in EXCITE, we have metadata in

sowiport. Also for matching, we use sowiport and ssoar as target dataset. We use these symbols of corpora in our structures for iris. The structures are shown in table 1.

Format of iris for "id"s entities
• gid: match_corpus(0001) sowi_id(csa-pais-1994-0302653) idtype • gid: match_corpus(0000) ssoar_id(43864) idtype
Format of iris for "br"s entities
• gbr: corpus(0000) ssoarid • gbr: corpus(2000) ref_id ← not match • gbr: corpus(0001) sowi_id • gbr: corpus(2005) ref_id • gbr: corpus(0005) paper_id
Format of iris for "ar"s and "ra"s entities
• gar: author(00) sowi_match(0001) (indexaut) (sowiportid) • gar: publisher(01) ssoar_match(0000) (indexaut) (ssoarid) • gra: author(00) sowi_match(0001) (indexaut) (sowiportid) • gra: publisher(01) ssoar_match(0000) (indexaut) (ssoarid)
Format of iris for "be"s entities
• gbe: corpus(0000) ref_id • gbe: corpus(0005) ref_id

Table 1. The structures used for "iri"s before the conversion to OCC "iri"s' format

for processing other corpuses than SSOAR papers we use the following function:

from Othercorpus_full_OCC **import** Generate_data_other_corpus

The function gets the following inputs:

1- Target_dir: A directory for saving the JSON output. (the default value is ./data directory)
2- ID_courpus: The symbol of the corpus, that we want to process.
3- SourceDir: where csv tabel of the corpus is stored there.

4-2- "br"s from different corpora into a PSQL table:

After creating each outoup json files for each corpora, we need to load all brs from each of these files in a table for further analysis. We do this job by using the following function:

from the load_df_into **import** loaddata_from_csv_to_table

This function gets three inputs:

1- Corpus_id_name: Id of the corpus that you need to upload the output json of that into the Psql table.
2- flag_source_dir: If the source directory is ./data, then it should be zero since only we need to use "Corpus_id_name" parameter for referring to the specific file that you need to upload.

3- file_fulldirectory: if "flag_source_dir" is not zero, then you should also specify the exact address of the file to be read and loaded into the Psql table.

if you saved the JSON output for each corpus in the default directory, then it is only needed to give the id of the corpus to load data from that file to the table. In other cases, for example, when we want to some additional papers for a corpus and we don't want to process the whole corpus, we can save the result of the extra papers in another file and directly from that file we load data into the database.
We read data of each JSON file and focus on the list of "br"s of them. The Psql table has the following columns:

1- Corpus: the symbol for corpuses (e.g. 0001 for ssoar)
2- Iri: the generated iri for each br
3- br_value: each br value for each record
4- a: the type of each br
5- contributor: the authors' and publishers' information
6- date: the information about the date of publishing
7- identifier: identifier information such as Doi, URL, and etc.
8- label: the label for the record
9- title: title of the record
10- citation: information about records cited in the work
11- reference: information about the list of reference strings
12- inserttoken: an auxiliary value to show that when a record is inserted into the table.
13- ex_mat_cor: extracted information about the corpus of br from iri value
14- auxnr: An auto incremental value for referring each br in the table. iri are not enough since we may have same br from some different sources. (e.g different reference strings which refer to a same bibliographic record).

4-3- Remove Dublicates from the Psql Table:

The next step would be removing duplicates from list of brs, which we remove them by applying:

```
from remove_dublicate import remove_dub_record
```

At first, the function checks the whole br records, if it could find brs with a same iri, then it tries to find which one has reference strings to keep the record. If all of them don't have any reference strings, then it checks which br contains some metadata, and then remove those without metadata. If none of two above checks find a candidate to be kept then one item randomly will be picked and afterward, all others will be removed.

4-4- Convert "iri"s from EXCITE Format to OCC Format:

The iris and labels should be replaced by the new format which is ok for OCC. The only it is needed to run the following function:

```
from replaceid_gen import main_replace_id
```

If it is the first time that you run the code on data, then after the process gets finished completely, 4 new json files will be generated beside the final json file:

1- be_id.json, 2- br_id.json, 3- id_id.json, and 4- ra_ar.json

These files contain map information between the old ids and the new ids. For example, from br_id. Json:

"0001ubk-opac-HL000301369": "011018012"

Next time that we want to process another corpora or new records for the corpora that we checked them before, the information will be used to avoid generating different new ids for a same old id. In this way we try to keep the consistency of ids.

The following function is for decoding old "iri"s to returns different meaningful parts of them.

from decode_encode_id **import** decode_id

Here is an example 7ort he function:

- Input: decode_id(„0001gesis-solis-0022305720","gid")
- Output: {‚corpus': ‚0001', ‚id': ‚gesis-solis-00223057', ‚type': ‚isbn'}

5- Big Corpus but Small Output:

In EXCITE we have a huge number of PDFS, which after converting their related data to a JSON file with OCC ontology, a big file will be generated. If you are interested in a small file, there is a solution in the code set. When you load a JSON file into the Psql table, there is a "inserttoken" so we can use this column for filtering data and then replace ids and generate a smaller chunk of data. Please consider that we should generate the final JSON file only after applying the deduplication code in Psql table.

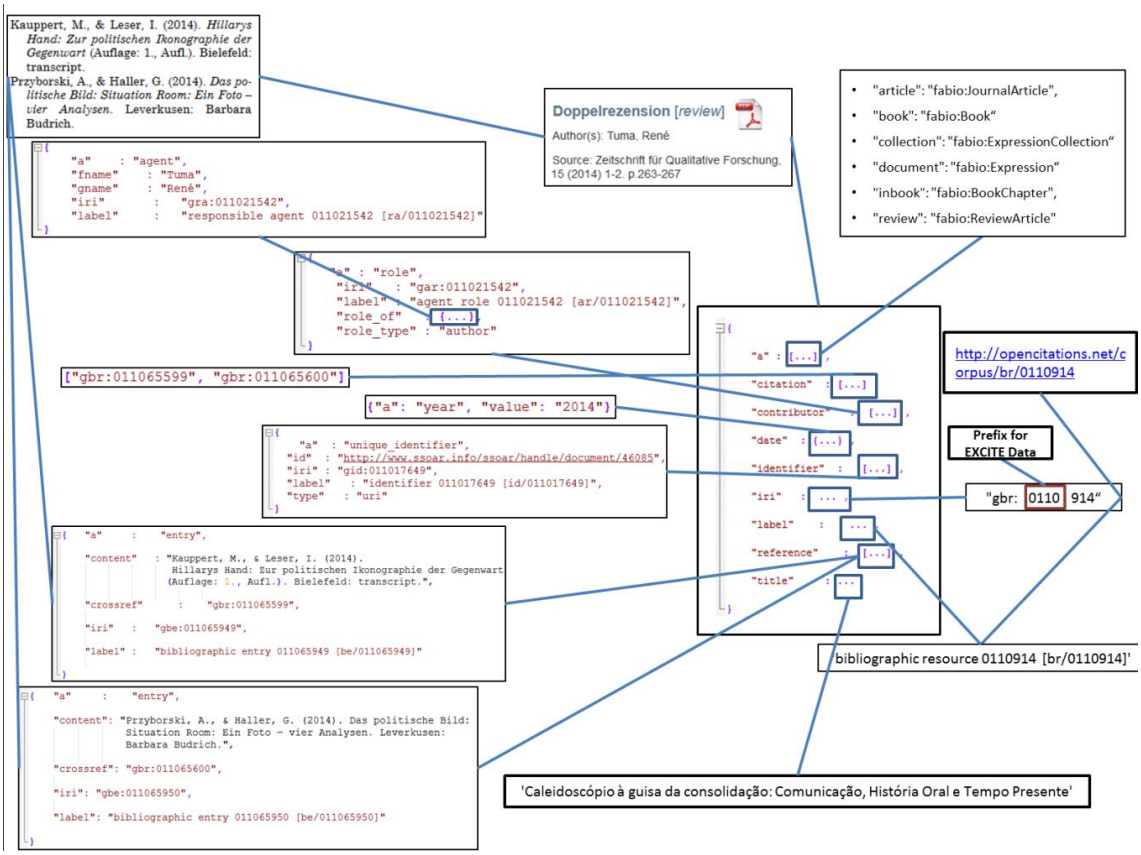

Figure 3. an example of a "br" with values