ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA

Second Cycle Degree Programme in

DIGITAL HUMANITIES AND DIGITAL KNOWLEDGE

Dissertation Title

Extending The OpenCitations Codebase For Creating Non-DOI Citation Indexes: The Ingestion Of The NIH Open Citation Collection

Final Dissertation in

Open Science

Supervisor: Silvio Peroni

Co-supervisor: Ivan Heibi

Presented by: Arianna Moretti
Matriculation number: 950520

Session III

Academic Year
2021-2022

Table of contents

Table of figures

List of tables

List of scripts

Abstract

This thesis introduces the OpenCitations Index codebase expansion implemented for the creation of NOCI, a new citation index based on the National Institute of Health Open Citation Collection (NIH-OCC). Since the entities in NIH-OCC are identified with PMIDs, the software infrastructure was adjusted to include and manage the new identifier typology into a DOI-based system. The adjustments allow the possibility of handling duplicated data, i.e., adding the same citation twice (both as a DOI-to-DOI and PMID-to-PMID form) into the OpenCitations dataset. To handle this issue another type of identifier for bibliographic entities was adopted, i.e., the METAID, to unambiguously identify a bibliographic entity in OpenCitations, tracing back all its identifiers to the same METAID. Both the data harvested for populating NOCI and the mapping information generated are stored as RDF triples and made available in a triplestore, queryable using SPARQL language.Simplified data-access alternatives are provided, such as an API service and data dumps. The development of the software expansion required programming in Python and using the Semantic Web technologies, and a theoretical background on the OpenCitations Data Model. The achieved software flexibility will be beneficial for OpenCitations not only for the NOCI integration, but also for future ingestions dealing with PMID and other types of identifiers. This thesis starts by reviewing the citation data and how these are stored and represented in OpenCitations, and moves toward the definition of a methodology for the creation of NOCI. This approach involves the development of software expansions on the current OpenCitations software infrastructure, which is discussed in a both theoretical and technical perspective. Finally, the new software is evaluated through an analysis of its computational costs.

1. Introduction

The aim of this thesis is extending the software handling the OpenCitations indexes[1] for the ingestion of the new index of citations: NOCI, a dataset holding a collection of PMID-to-PMID citations provided by the National Institute of Health.

The thesis project falls within OpenCitations infrastructure (OC), a non-profit organization advocating for open citations, aimed at publishing open bibliographic and citation data exploiting the Semantic Web technologies (Peroni e Shotton 2020). A citation index is a dataset of references between publications, derived from data provided by specific bibliographic databases. In OpenCitations, all the indexes store citations as first-class data entities with accompanying properties (Shotton 2018b), identified by an Open Citation Identifier (OCI). The metadata of each citation are expressed as RDF statements according to the OpenCitations Data Model (OCDM), and can be harvested by querying the Index SPARQL endpoint, as well as using the Index REST API, or downloading dumps of the full index[2].

Currently, two OpenCitations indexes are available: COCI, storing Crossref open DOI-to-DOI citations (Heibi, Peroni, and Shotton 2019b), and CROCI, the Crowdsourced Index (Heibi, Peroni, and Shotton 2019a). The incorporation of a new open subset of citation data from the National Institute of Health (i.e., the NIH Open Citation Collection[3]) into OpenCitations demands the need for an extension of the software infrastructure to handle the peculiarities and the diverse nature of the new citations data. Technically, this means developing codebase expansions following the model approach adopted in OpenCitations for the already available and ingested indexes, but with respect to the nature of the citation data of NIH. These extensions mainly concern the management of the new PMIDs used to identify the entities in the NIH Open Citation Collection. Indeed, the main peculiarity of NOCI - with respect to OpenCitations infrastructure - is that it gathers reference data where both the citing and the cited entities are identified by PMIDs – the NIH National Library of Medicine identifiers[4] assigned to documents indexed in PubMed[5] (Canese and Weis, 2013.).

[1] *index*, Python (2019; repr., OpenCitations, 2021), https://github.com/opencitations/index.
[2] 'OpenCitations - OpenCitations Indexes', accessed 21 February 2022, http://opencitations.net/index.
[3] ICite, Hutchins, B. Ian, e Santangelo, George, «iCite Database Snapshots (NIH Open Citation Collection)», 2022, https://doi.org/10.35092/YHJC.C.4586573.V26.
[4] 'PMID vs PMCID: What's the Difference? – NIH Extramural Nexus', accessed 24 January 2022, https://nexus.od.nih.gov/all/2015/08/31/pmid-vs-pmcid-whats-the-difference/.
[5] «PMID vs PMCID: What's the Difference? – NIH Extramural Nexus», consultato 24 gennaio 2022, https://nexus.od.nih.gov/all/2015/08/31/pmid-vs-pmcid-whats-the-difference/.

Currently, the open citation indexes provided by OpenCitations are based on DOI identifiers, therefore until now the software was developed to manage DOI-identified entities only. The ingestion of NOCI requires codebase additions to handle this new type of identifiers. This means a higher adaptation of the software for different identifiers (beside DOIs) never faced before for the other ingested indexes. This software flexibility will be beneficial for OpenCitations not only for the NOCI integration but also for future ingestions dealing with PMID and other types of identifiers.

One of the main issues that emerged following the ingestion of PMID-identified entities in OpenCitations was the possibility of having data duplication. Potentially, a citation could be gathered twice in the system with no prior knowledge to distinguish the two entities. Technically, this happens when having a citation expressed in both a DOI-to-DOI and a PMID-to-PMID form. To overcome this situation, a mapping system was introduced based on a new identifier, i.e. the METAID. The mapping is based on the association of each identifier of the same entity (DOI or PMID) to a new unique id (METAID). The METAID identifies a data block storing the metadata information of the entities (citing or cited) managed in the system.

From a technical point of view, the development of the project required mastering the codebase, which practically meant programming in Python Programming Language[6] and using the Semantic Web technologies, such as the Resource Description Framework (RDF) data model for metadata[7] and the internal OpenCitations Data Model (OCDM), used for the metadata management of all the OpenCitations' datasets (Daquino et al. 2020).

This thesis starts from a presentation of OpenCitations and a background to describe the current situation before moving the discussion toward the aforementioned development aspects from both the theoretical and technical perspectives. The second chapter is dedicated to the state of the art, including an overview on the OpenCitations Infrastructure and in particular on the available citation indexes, an introduction to other external citation datasets, and a detailed presentation of the NIH Open Citation Collection. Chapter three introduces the adopted methodology. It starts from a nontechnical introduction to the OpenCitations index structure and functioning, followed by an analysis of the main phases of the workflow which highlights the main issues and the implemented solutions. The workflow of the methodology is studied more in detail in the two following chapters,

[6] 'Welcome to Python.Org', Python.org, accessed 3 February 2022, https://www.python.org/.
[7] 'RDF - Semantic Web Standards', accessed 21 February 2022, https://www.w3.org/RDF/.

which present the Python programming implementation aspects at a base and advanced level, respectively. In particular, the fourth chapter introduces the OpenCitations specific concepts and structures used for the development of the NOCI index, as well as the coding standards and the file formats adopted. On the other hand, the fifth chapter discusses the technical details of the code. In the sixth chapter, we evaluate the process with respect to its computational costs. Finally, in the last chapter we summarize and give some final remarks regarding the work of this thesis.

2. State Of The Art

The aim of this chapter is to provide a general background of both the field of study and the OpenCitations infrastructure, within which the work of this thesis was conceived and developed. The first section of this chapter is dedicated to an introduction to citation data and, in particular, to open citation data. The second section is about OpenCitations, its origins, background field, principles, purposes, and perspectives. After the OpenCitations overview, our discussion focuses on the analysis of the two available indexes of OpenCitations: OpenCitations Index of Crossref open DOI-to-DOI citations, COCI (Heibi, Peroni, and Shotton 2019b), and Crowdsourced Open Citations Index, CROCI (Heibi, Peroni, and Shotton 2019a). Both the indexes provide the conceptual and the technical basics of the model for the creation of NOCI. The third section presents an introduction to other datasets external to OpenCitations, which manage citational data of similar type. The last section contains a detailed presentation of the NIH Open Citation Collection and the nature of the citation data stored in it. The description of such data will motivate our need to extend the existing infrastructure, so as to manage non-DOI identifiers (used in NIH) and foster the introduction of the new citation index NOCI.

2.1. Working With Citation Data

In scholarly domain, a citation is a link between a referencing work and another work referenced in it, generally established by the act of acknowledgement of the latter among the bibliographic references or in the footnotes of the other (Peroni and Shotton 2018, 2020). The informative content conveyed by the connections defined through citations is crucial from both qualitative and quantitative aspects. Citations have the power to tie publications to other publications, highlighting networks showing general patterns of a collective labor of knowledge building, sharing, and debate (Peroni and Shotton 2018, Peroni et al. 2015). Furthermore, other meaningful information is led by the directionality of the link between the citing and the cited work, since it is helpful in recognizing scholar credit and responsibility to the authors of the source entity. For what concerns the nature of the node-entities connected by the citations, they can be any kind of authored document which can be properly acknowledged by referencing (Shotton 2018a, Peroni et al. 2015).

The usefulness of the citation data increases proportionally with the amount of bibliographic details provided within the records of the two entities involved in the reference (i.e., the citing and cited entities). These metadata are not always exposed from a data source together with the citations

themselves, but if the bibliographic entities are identified by certified ids (e.g. DOI, PMID) it is both possible to go back to the original documents without ambiguities and to exploit external API services to retrieve missing information about the entities, so to recover the citation metadata (Peroni and Shotton 2018).

However, the most critical aspect about the state of citation data is without any doubt a system which still allows this type of information to be conceived as proprietary (Shotton 2018a, Peroni and Shotton 2018). Indeed, at the time of registering a publication at one of the major digital object identifier Registration Agencies, i.e.: CrossRef[8], it is currently up to the publishers whether to share or not the citation data for their work.

What should bring bibliographic citations out from payment copyright-related issues is their state of pure facts: as factual and not contenutistic information, they should be made freely accessible and shareable (Peroni et al. 2015, Peroni e Shotton 2020). This assumption, together with the awareness of the huge potential of citation data in scholar, scientific, and common use applications led to the foundation of what would become OpenCitations Infrastructure, and coherently still guides its ongoing evolution.

2.1.1 Open Citations and I40C

The current subsection introduces the specific criteria for a reference record to be considered as a proper open citation. Indeed, achieving openness for citations not only implies keeping both the citing and cited entity identifiable according to a certified system of persistent identification or an uniform resource locator, but also assuring the possibility to exploit the identifiers of both the publications to obtain the fundamental metadata associated to each of them (Peroni and Shotton 2018). In addition to these two requirements, other more specific conditions are declared by the Initiative for Open Citations (I4OC), setting a standard which requires the open citations to be:

1) Expressed in at least one format which can be interpreted by machines (i.e.: being "structured"). In particular, the practice of storing citation data in machine-readable format, possibly in open access repositories, enhances the automatized exploitation of the citation data and opens up new opportunities for digital applications (Peroni et al. 2015).
2) Individually accessible even when the associated bibliographic work is not (i.e.: being "separate"). This aspect is crucial since it splits the right to have access to the metadata from the possibility of having free access to the content of the publication, which is optional.

[8] «You are Crossref - Crossref», accessed 23 January 2022, https://www.crossref.org/.

3) Reachable and exploitable at no charge or restriction (i.e.: being "open"). To this end, a good practice is that of choosing a CC0 1.0 Universal waiver/license[9] for the publication of citation data (Peroni and Shotton 2018).

For what directly concerns OpenCitations - in addition to being compliant with the FAIR data principles (i.e: providing data which are findable, accessible, interoperable and reusable[10]) introduced by the scholars community Force11[11] - the infrastructure not only follows the I4OC directives, but it also figures among its founders.

The idea of giving birth to the aforementioned initiative dates back to 2016, on the occasion of the 8th Conference on Open Access Scholarly Publishing. At that time, the shared purpose of the six founding organizations (nominally: OpenCitations, Wikimedia Foundation[12], Plos[13], eLife[14], DataCite[15] and Center for Culture and Technology[16]) was that of gathering together all the relevant actors in the creation, management, reuse and distribution of citation data without any access restriction. The sensibilization activity performed by I4OC consists in asking scholarly publishers to make the reference lists they provide to CrossRef publicly available, and it took no time for the outcomes to be evident.

Indeed, the year following the foundation of the Initiative came with a crucial turnabout with respect to the previous trend: sixty and more publishers - including some of the majors - soon chose to make openly accessible their reference lists, and only a few months later around five hundred million references hosted by Crossref had been unlocked (Shotton 2018a). Further progresses in the citation data disclosure trend have been constantly made afterwards, so that the proportion of open citation publications in CrossRef grew up to 59% of the total, having started from one out of a hundred before the I4OC launch (Peroni e Shotton 2020).

For what concerns future developments, it is not only predictable but also desirable that also those publishers which are still reluctant to respond to the call for open citation data may be persuaded by the reasonableness of the recommendation. Accordingly, among the many and various reasons for following the virtuous example of the growing number of publishers who already agreed to the proposal, it is possible to enumerate: enhancement of the study activities performed by researchers,

[9] 'Creative Commons — CC0 1.0 Universal', accessed 22 January 2022, https://creativecommons.org/publicdomain/zero/1.0/.
[10] «FAIR Principles - GO FAIR», accessed 23 January 2022, https://www.go-fair.org/fair-principles/.
[11] «FORCE11 | The future of research communications and e-scholarship», accessed 23 January 2022, https://www.force11.org/.
[12] «Wikimedia Foundation», accessed 23 January 2022, https://wikimediafoundation.org/.
[13] «Home - PLOS», accessed 23 January 2022, https://plos.org/.
[14] «Latest research | eLife», accessed 23 January 2022, https://elifesciences.org/.
[15] «Welcome to DataCite», accessed 23 January 2022, https://datacite.org/.
[16] «Home - Centre for Culture and technology (CCAT) | Curtin University, Perth, Western Australia», accessed 23 January 2022, https://ccat.curtin.edu.au/.

authors, libraries, and their stakeholder communities; simplification of the assessments concerning the impact of publications and possible allocations of new funds; more efficient redirection to targeted online publications; and development of new software and visualizations for the management of the now available amount of data (Peroni e Shotton 2020).

2.2 OpenCitations Infrastructure: Background Field, Origins and Guiding Principles

The contribution of this thesis represents an implementation of a research project in the independent framework of OpenCitations: a not-for-profit organization for open scholarship. The aim of this infrastructure is exploiting Linked Data and Semantic Web tools and technologies for the publication of open data, both citation and bibliographical[17]. This section starts from a general introduction and description of the citation data and presents OpenCitations throughout a discussion toward its origins, its main contributions and how citation data are managed and represented in it.

2.2.1 OpenCitations Origins, Success and Evolutions

In a forerunner journal article published in Nature in 2013 (Shotton 2013), David Shotton introduced for the first time in a publication what would become the start-up project of OpenCitations: the OpenCitations Corpus (OCC), whose realization was made possible by Joint Information Systems Committee (JISC)[18] funds, aimed at sponsoring the implementation of groundbreaking technological proposals for UK scholarship and erudition (Peroni e Shotton 2020).
The intention leading to the creation of this repository hosting scholarly citation data under a CC0 license (i.e.: a Creative Commons public domain dedication[19]) was the will to provide the community with a freely accessible and exploitable resource, which would be also reliable for scientific purposes. Indeed, when the two founders and directors of OpenCitations Silvio Peroni and David Shotton[20] started working at OCC in 2010, the accessibility of scholarly citation data was critical, in particular due to the resistances to make public Elsevier's Scopus and Thomson Reuters Web of Science data, which in combination covered the majority of the available material of the field (Shotton 2013, Peroni e Shotton 2020).
Since the beginning, the information managed by OpenCitations was stored in machine-readable format and - in particular - encoded as Linked Open Data, leveraging the Semantic Publishing and

[17] «OpenCitations - Home», accessed 17 January 2022, https://opencitations.net/.
[18] «Jisc», Jisc, accessed 25 January 2022, https://www.jisc.ac.uk/.
[19] «Creative Commons — CC0 1.0 Universal», accessed 22 January 2022, https://creativecommons.org/publicdomain/zero/1.0/.
[20] «OpenCitations - About», accessed 22 January 2022, https://opencitations.net/about.

Referencing (SPAR) Ontologies and the Semantic Web guidelines, so as to maximize data exploitation, easy reuse, and further integrations from a variety of sources hosting data of different study areas of interest (Shotton 2013, Peroni et al. 2015).

In the following years, the organization kept growing with noticeable success, leading to the introduction of new datasets[21], which enhanced bibliometrics studies, with many advantages for the scientific community at large, but in particular for the researchers out of the system of academic libraries able to afford the price of the access to commercial citation data (Peroni et al. 2015). Indeed, despite the limited purposes of what was the original Open Citations Project, its resources are now more than ever exploited as such a reliable source of information that in recent years the infrastructure has arrived to provide data for several academic projects and official bibliometric studies (Peroni e Shotton 2020).

Further future evolutions of OpenCitations infrastructure will also be related to the envisioned acquisition of data from other open citation datasets, whose management requires the development of follow-up extensions of the nowadays available tools (Peroni e Shotton 2020).

2.2.3 OpenCitations Indexes

The data currently hosted by OpenCitations are stored in three dataset groups: the OpenCitations Corpus (OCC), the OpenCitations in Context Corpus (CCC), and the OpenCitations Indexes.

Both OCC and CCC are based on biomedical articles from the Open Access Subset of PubMed Central, and are released under a Creative Commons CC0 public domain license. The main difference between the two is that the CCC data is derived from the full text publications, exploiting contextual elements, such as in-text references and pointers or footnotes[22].

However, the main focus of this thesis is on the OpenCitations Indexes. The ingestion of our new set of data provided by the National Institute of Health (NIH) will produce a new index to be part of that group: NOCI.

In a citation index, the data can be represented in two ways: either as links between bibliographic publications (i.e.: referencing and referenced) or as entities of their own right. In the first case, the citations are conceived as properties of the entities, while in the second they are entities themselves, and thus can have properties (Shotton 2018b). OpenCitations chose to prioritize the representation of citations over publications (Peroni e Shotton 2018); therefore, one of the distinctive traits of the OC Indexes is that citations are considered as first-class data entities, to which are directly

[21] «OpenCitations - Datasets», accessed 22 January 2022, https://opencitations.net/datasets.
[22] «OpenCitations - Datasets», accessed 22 January 2022, https://opencitations.net/datasets.

associated specific properties, like the date, the timespan between the publication date of the referenced and the referencing publication, and additional information assessing whether or not the citing and the cited articles share the same publisher.

At the current state, the Digital Object Identifier is the only type of identifier for bibliographic entities managed by the OC Indexes, and they are exploited to retrieve metadata for both the parts involved in the citation data, through calls to external APIs[23].

The two available OC Indexes are: the OpenCitations Index of Crossref open DOI-to-DOI citations, COCI (Heibi, Peroni, and Shotton 2019b) and the Crowdsourced Open Citations Index,CROCI (Heibi, Peroni, and Shotton 2019b)[24]. Both store in RDF citation data where the involved publications are identified by Digital Object Identifiers and share common structural elements and development tools, however, they mainly differ the one from the other for the data gathering approach. In the next two subsections we discuss separately COCI and CROCI.

COCI

The OpenCitations Index of Crossref open DOI-to-DOI citations (Heibi, Peroni, and Shotton 2019b) was created in 2018 and populated with open references deposited at CrossRef[25], particularly benefitting from the effects of the data disclosure campaign promoted by I4OC since 2016 (Peroni e Shotton 2020). Indeed, at the time of the last update in late November 2021, the index hosted 1,235,170,583 citations and 69,897,400 bibliographic resources: a substantial amount of data which grants COCI a position among the major datasets with relevant citation data coverages, which could have been impossible to imagine before the boom of the unlocking trend for reference lists provided by publishers to CrossRef in 2017 (Martín-Martín et al. 2021, Shotton 2021).

COCI Data acquisition and management goes through a pipeline of four major phases.

In the first phasethe CrossRef database is queried in order to gather all the DOI-identified publications with their references lists and - where possible - the acquisition of some relevant information concerning the publication dates of the entities involved in the citations, the registered identifiers of the authors (ORCID), and the resources where the works were published (ISSN).

The second phase stores the citations in a CSV format. Each line represents a citation, where the first column contains the identifier of the citing entity and the second column the identifier of the

[23] 'The Unifying REST API for All the OpenCitations Indexes', accessed 20 February 2022, https://opencitations.net/index/api/v1.

[24] «OpenCitations - Datasets», accessed 22 January 2022, https://opencitations.net/datasets.

[25] 'OpenCitations - Download', accessed 20 February 2022, https://opencitations.net/download#coci.

cited entity (i.e., DOIs of the citing and the cited entities).

In the third phase, the data stored in the CSV file are converted to triples in RDF format. Finally, in the last phase, the RDF dataset is uploaded to a triplestore, which is updated periodically.

The data of COCI are made available to the public in several ways: an endpoint for SPARQL requests, a RESTful application programming interface for the web users which are not acquainted with Semantic Web tools and languages, the access to bulk downloads, some user-friendly and intuitive web interfaces, and a HTTP content negotiation for the direct access to the citations[26].

CROCI

Crowdsourced Open Citations Index (CROCI)[27] is similar in structure and purposes to COCI, yet it draws its data from other resources. CROCI was created to overcome COCI coverage limits by exploiting the voluntary participation of all those voluntary researchers who are willing to make publicly available the data they have access to, after some preliminary processing required to avoid waiver violations (Heibi, Peroni, and Shotton 2019a).

The proposed strategy is supported from the definition of open citation and on the assumption that "citations are statements of fact about relationships between publications" (Heibi, Peroni, and Shotton 2019a, 6), and for this reason the only thing that could be protected by copyright is the textual arrangement of a specific reference list. Accordingly, what is asked to the community is to submit the citation data together with the publication dates of both the citing and the cited entities involved in a reference. Besides the widespread CSV file format, OpenCitatons also accepted data in the Scholix format, proposed by the homonyme interoperability initiative (Cousijn et al. 2019) "to create an open global information ecosystem to collect and exchange links between research data and literature"[28].

Each volunteering contributor must be self-identified by providing their ORCID, upload the material either on Figshare[29] or Zenodo[30], and open a new issue on the CROCI GitHub repository[31], so as to make the OpenCitations team aware of the contribution.

[26] 'OpenCitations - COCI', accessed 23 January 2022, https://opencitations.net/index/coci.
[27] «OpenCitations - CROCI», accessed 23 January 2022, https://opencitations.net/index/croci.
[28] 'About - SCHOLIX', accessed 20 February 2022, http://www.scholix.org/about.
[29] «figshare - credit for all your research», accessed 24 January 2022, https://figshare.com/.
[30] «Zenodo - Research. Shared.», accessed 24 January 2022, https://zenodo.org/.
[31] «Issues · opencitations/croci · GitHub», accessed 24 January 2022, https://github.com/opencitations/croci/issues.

2.3 Field Analysis: Other Citation Datasets

In recent years, the cultural scene concerning the citational data sources has been changing considerably in a relatively short timespan. Coherently, the data coverage of many of the available datasets has greatly improved after the I4OC sensibilisation campaign which led to the public exposure of a large amount of data which had been proprietary until then.

This section provides an introductory overview of some of the main resources maintaining reference datasets or offering access to information about citations. However, the scene of citation indexes significantly changed after the launch of I4OC, and thus the exposed information should be considered provisional because of the fast mutability of the data in the field. A case in point of this trend is presented in a OpenCitations blog post published by David Shotton with a self-explanatory title[32], where Alberto Martín-Martín presented data proving that - after only three years from the moment of its creation - COCI had already reached such a wide citation data coverage that could be compared to the ones achieved by the two giants Web of Science and Scopus.

These two institutions held the unquestioned primacy in the bibliometric field for years before the I4OC advent, by detaining a huge amount of proprietary data which they were reluctant to make open. However, the following analysis reveals the strengths that let these two databases excel in the citation indexes scene.

Scopus was founded by Elsevier as a subscription-based database, with the aim of gathering a wide selection of abstracts and citations, setting up an authoritative source of information for scientific research. Its reliability is in part due to the owning notorious publishing company, but in particular to the amount of data maintained, which also covers a variety of research fields and disciplines. According to the statistics presented by Elseveir, Scopus currently includes more than 82 millions of documents and 1,7 billions of references (from the 1970s onward), from more than 7000 publishers[33].

The other leader subscription-based service is Thomson Reuters Web of Science[34] (WoS), owned and managed by Clarivate Analytics. As for Scopus, also WoS hosts scholarly citatiotion data from different scientific areas, allowing extensive cross-disciplinary research. WoS is considered as one

[32] David Shotton, «Coverage of Open Citation Data Approaches Parity with Web of Science and Scopus», *OpenCitations Blog* (blog), 27 ottobre 2021, https://opencitations.wordpress.com/2021/10/27/coverage-of-open-citation-data-approaches-parity-with-web-of-science-and-scopus/.

[33] Elsevier, 'Content - How Scopus Works - Scopus - | Elsevier Solutions', Elsevier.com, accessed 21 February 2022, https://www.elsevier.com/solutions/scopus/how-scopus-works/content.

[34] «Clarivate», accessed 24 January 2022, https://access.clarivate.com/login?app=wos&alternative=true&shibShireURL=https:%2F%2Fwww.webofknowledge.com%2F%3Fauth%3DShibboleth&shibReturnURL=https:%2F%2Fwww.webofknowledge.com%2F&roaming=true.

of the most trustworthy research engines for citation data studies. Fifty years of indexing activities led to gathering 1.9 billion references from more than 171 million records[35].

Moving to non subscription-based services, a relevant source of citation data among the bibliometrics authorities is Google Scholar[36]. Indeed, as opposed to WoS and Scopus, this dataset is freely accessible to the public, although its sources have significant access restrictions, if compared with the fully-open competitors' resources. According to a study by Van Noorden (Van Noorden 2014), no official statistics are made available by Google Scholar, even if it was recently computed that its data should cover around 300 million records and probably more, including most of the material present in Scopus and WoS (López-Cózar, Orduna-Malea, and Martín-Martín 2018). For what concerns its drawbacks, the most significant weaknesses are the poor metadata and the complex extraction process to be performed in order to access them.

However, the monopoly of the established authorities on the field has been opening to new emerging forces which have become available in a relatively short timespan. In addition to indexes of OpenCitations, some of the new sources deserving mention are Microsoft Academic and Dimensions, whose data coverages can already be compared to the forerunners' ones.

Microsoft Academic[37] was founded in 2016 as a free scholar service exploiting Bing's preexisting framework. This search engine also provides API services to make bulk data retrieval easier[38], and - in spite of some reuse limitations (Martín-Martín et al. 2021, 873) - this platform must be counted among the breakthrough services which significantly improved the open access to data for bibliometrics research, maintaining data for 225 million publications.

Two years after the creation of Microsoft Academic, Digital Science founded the Dimensions[39] dataset. The access format proposed for this resource is blended, consisting in the possibility to exploit for free only the basic functionalities, while implying payment for the ones included in the premium package. Dimensions includes more than 124 million publications, from 87k journals, 49 preprint servers and more than 1m books[40]. Its coverage is particularly complete for items associated with a DOI identifier (Martín-Martín et al. 2021).

A recent comparative study held by Martín-Martín, Thelwall, Orduna-Malea and Delgado

---

[35] «Trusted publisher-independent citation database - Web of Science Group», accessed 25 January 2022, https://clarivate.com/webofsciencegroup/solutions/web-of-science/.
[36] «Google Scholar», accessed 24 January 2022, https://scholar.google.com/schhp?hl=en&as_sdt=0,5.
[37] «Microsoft Academic», *Microsoft Research* (blog), accessed 25 January 2022, https://www.microsoft.com/en-us/research/project/academic/.
[38] «Microsoft Academic Graph - Microsoft Research», accessed 25 January 2022, https://www.microsoft.com/en-us/research/project/microsoft-academic-graph/.
[39] «Dimensions - The next Evolution in Linked Scholarly Information», Dimensions, accessed 24 January 2022, https://www.dimensions.ai/.
[40] 'Dimensions AI | The Most Advanced Scientific Research Database', Dimensions, accessed 21 February 2022, https://www.dimensions.ai/.

López-Cózar (Martín-Martín et al. 2021) for assessing the current situation of some of the major bibliometric databases revealed the results presented in the Figure 1. The chart is represented as a table aimed at presenting the rates of references covered by several sources of citations with respect to each other and considering the total number of citations covered .

The study took into account 3,073,351 citations from all the aforementioned sources (including COCI of OpenCitations), involving a total of 2,515 documents written in English, covering 252 subject areas. The most evident result is the outstanding coverage reached by Google Scholar, which obtained a rate of 88% on the total number of considered citations and more than the 90% on almost the totality of the other datasets taken individually. All the other sources' ranged from 52% to 60% of the total, with the only exception of COCI, which reached a 28% in 2019, but almost doubled its previous result in only two years, reaching a 50% coverage of the total number of citations in 2021.

The obtained results, as previously mentioned, are the tangible proof of a groundbreaking turnabout concerning the proprietary sources of data monopoly on the bibliometrics studies. Indeed, since the increasing awareness of the various applications which can exploit citation data and the acknowledgement of the nature of the reference datum as a pure fact information which shouldn't be copyrighted, many formerly closed sources such as Scopus have been choosing to propose alternatives which allow the free access to their data, consequently leading other sources to increase their coverage.

To conclude the overview on the currently available bibliometrics sources, a number of other databases should be mentioned. For the sake of completeness, a subset of eight other sources have been selected, based on their similarity (conceptually and technically) to the data managed in the OpenCitations Indexes.

1) *WikiCite*[41]: a dataset which exploits the material harvested by Wikidata, with the aim of enhancing, augmenting and refining citations in Wikimedia. The principles guiding the initiative are the openness of data and the will of creating a collaborative environment to support the growth of the bibliometrics study field with the development of innovative tools. The most updated data officially exposed by WikiCite date back to the 10th of January 2022, and declare a total of more than 284 million citation data covered[42].
2) *Open Ukrainian Citation Index (OUCI)*[43]: an Ukrainian index meeting OpenCitations selection criteria. Despite being mainly focused on material published in Ukraine or at least

[41] «WikiCite», accessed 24 January 2022, http://wikicite.org/.
[42] «Statistics», accessed 25 January 2022, http://wikicite.org/statistics.html.
[43] «OUCI», accessed 25 January 2022, https://ouci.dntb.gov.ua/en/.

in Ukrainian language, this search engine manages information regarding around 131 million publications from all over the world, in addition to more than 375.000 publications in 1627 Ukrainian journals[44]. The citation database exploits the subset of Crossref's material which was provided with the Cited-by service, allowing full access to citations[45]. The primary aim of OUCI is enhancing access to scholarly citation data - in particular for Ukrainian publications - by exploiting DOI identifiers, in order to improve the state of the art for what concerns bibliometrics studies in general, and in particular at national level.

3) *ScholExplorer*[46]: a resource by OpenAIRE. It slightly differs from the previously mentioned indexes, since it specifically provides a service giving access to links between resources (e.g.: datasets, literature objects), which are not necessarily citation data. The service exploits the information received from certified resources to create a graph depicting their connections, which is openly accessible (published under a CC0 waiver). The links are represented in Scholix format and the output information can be retrieved through REST APIs services.
4) *Europe PubMed Central*[47]: among the other authoritative sources for bibliometric studies, Europe PubMed Central is a renowned pillar in the support of open access to academic literature. In particular, it makes freely available to any user more than 39.500.000 million documents of a wide variety of types, with all the links to the metadata and the related materials[48].
5) *Scholar Archive*[49]: a new full-text search engine in its launch phase, which is improving the quality of the exposed metadata and finalizing the provided functionalities. Currently, it includes up to 25 million articles and academic documents, ranging from digitized copies of eighteenth-century publication to web preprints. The metadata provided are harvested from the collaborative scholarly catalog fatcat.wiki[50], and the possibility to recover provenance and metadata for the retrieved publications is always guaranteed.
6) *OpenAlex*[51]: this catalog is named after the ancient Library of Alexandria, and aims at providing a reliable free and open source of data. This index includes hundreds of millions of entities linked by more than a billion connections, which can be accessed either via web interface (in its launching phase in February 2022), API, or database snapshot. The system is

[44] Ibidem
[45] «Cited-by - Crossref», accessed 25 January 2022, https://www.crossref.org/services/cited-by/.
[46] «scholexplorer», accessed 24 January 2022, https://www.openaire.eu/scholexplorer.
[47] «Home - Europe PMC», accessed 24 January 2022, https://europepmc.org/.
[48] «About - Europe PMC», accessed 25 January 2022, https://europepmc.org/About.
[49] 'Internet Archive Scholar', accessed 21 February 2022, https://scholar.archive.org/.
[50] 'Perpetual Access To The Scholarly Record | Fatcat!', accessed 21 February 2022, https://fatcat.wiki/.
[51] 'OpenAlex: The Open Catalog to the Global Research System', accessed 21 February 2022, https://openalex.org/.

based on the description of five types of entities, i.e., works, authors, venues, institutions, and concepts, woven together in a huge directed graph.

7) *Semantic Scholar*[52]: a free research service for scientific literature based on Artificial Intelligence, containing more than 205 million papers, covering most of the scientific fields. The aim of the project is exploiting AI to overcome information overload and provide an open and free reliable service to the community. The access to data is guaranteed via semanticscholar.org website, the Semantic Scholar API, and the Open Research Corpus.
8) *iCite*[53] ; a project which is currently hosting more than 27,6 million publications, dated back to the 1980s onward[54]. This web-based tool is provided by the National Institute of Health[55] to allow the users to access a database of citation data for publications identified by PMIDs. Among all the other presented resources, this latter has a crucial role in the work of this thesis , as - besides other services - it also "disseminates link-level, public-domain citation data from the NIH Open Citation Collection"[56], namely the source of the data which motivated the extension of OpenCitations index software for the management of PMIDs in addition to DOIs, and the creation of the new index NOCI.

[52] 'Semantic Scholar | AI-Powered Research Tool', accessed 21 February 2022, https://www.semanticscholar.org/.
[53] «iCite | NIH Office of Portfolio Analysis», accessed 19 January 2022, https://icite.od.nih.gov/.
[54] «iCite | Statistics | NIH Office of Portfolio Analysis», accessed 25 January 2022, https://icite.od.nih.gov/stats.
[55] «National Institutes of Health (NIH) | Turning Discovery Into Health», accessed 25 January 2022, https://www.nih.gov/.
[56] «iCite | NIH Office of Portfolio Analysis», accessed 19 January 2022, https://icite.od.nih.gov/.

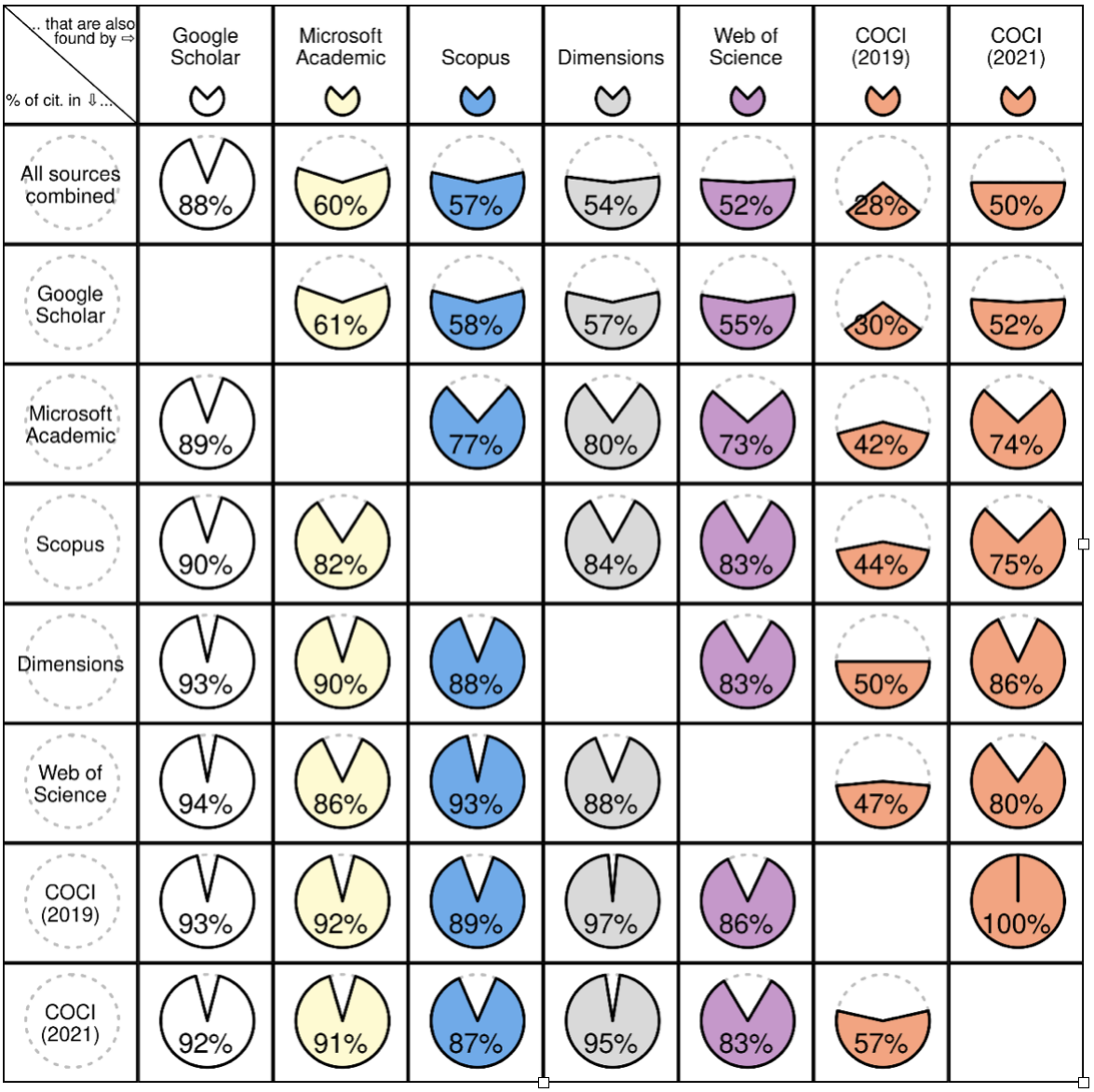

Figure 1. Coverage percentages of citation datasets. *A table showing coverage percentages for each database, both calculated on the total of the considered data and on the coverage of each of the other sources*[57].

2.3.1 NIH Open Citation Collection

This section introduces the dataset provided by NIH to be ingested in a new OC Index. The ingestion of this dataset can take place only after the implementation of the software extension

[57] Alberto Martín-Martín et al., «Google Scholar, Microsoft Academic, Scopus, Dimensions, Web of Science, and OpenCitations' COCI: a multidisciplinary comparison of coverage via citations», *Scientometrics* 126, n. 1 (1 January 2021): 871–906, https://doi.org/10.1007/s11192-020-03690-4.

which allows receiving information from a datasource where the identifiers of the citing and cited entities are not DOIs.

The provider of this dataset is the National Institutes of Health (NIH) Office of Portfolio Analysis (OPA)[58], which has been gathering, managing, and making available to the public the amount of citation data in exam – the NIH Open Citation Collection (NIH-OCC)[59].

The NIH-OCC database stores citation information for publications from the biomedical field of study. These publications are mainly gathered from some sources involved in this domain, such as PubMed Central[60], National Library of Medicine Entrez[61], or MedLine[62], together with some other more general-purpose sources, such as CrossRef. Further citation data were derived from the full-text of free open-access scientific publications available online (Hutchins et al. 2019).

As mentioned above, the novelty regarding this collection - with respect to the other datasets already ingested in OC - is the type of identifiers, since the entities involved in the references are primarily identified by PubMed Identifiers (PMIDs), which are codes attributed by the NIH National Library of Medicine to publications indexed in PubMed[63]. NIH-OCC is now freely available to the public on iCite, where an update snapshot is uploaded every month.

The integration of this collection in OpenCitations offers various opportunities: indeed, not only it implies the occasion for an extension of the OpenCitations index software to manage citations where the publications are identified with new types of identifiers, but it also implies the inclusion of specific data from the biomedical domain in a system that already manages data from a variety of sources, covering different interest areas. This latter aspect can help overcome the sectorality nature of the NIH Open Citations Collection.

[58] «The Office of Portfolio Analysis (OPA)», accessed 24 January 2022, https://dpcpsi.nih.gov/opa.
[59] «iCite Database Snapshot 2021-07», accessed 20 January 2022, https://nih.figshare.com/articles/dataset/iCite_Database_Snapshot_2021-07/15148737.
[60] «Home - PMC - NCBI», accessed 24 January 2022, https://www.ncbi.nlm.nih.gov/pmc/.
[61] Entrez Programming Utilities Help (National Center for Biotechnology Information (US), 2010).
[62] «MEDLINE Home», Product, Program, and Project Descriptions (U.S. National Library of Medicine), accessed 24 January 2022, https://www.nlm.nih.gov/medline/index.html.
[63] «PMID vs PMCID: What's the Difference? – NIH Extramural Nexus», accessed 24 January 2022, https://nexus.od.nih.gov/all/2015/08/31/pmid-vs-pmcid-whats-the-difference/.

3. Methodology

The present chapter introduces from a not-technical point of view the main aspects concerning the methodology adopted for the ingestion of the NIH Open Citations Corpus in the OpenCitations, and provides some further information about the issues encountered in the procedure, i.e. the introduction of PubMed Identifiers in a DOI-based infrastructure and the resulting potential data duplication in OpenCitations.

In particular, the adopted methodology will be presented with respect to the two main conceptual and procedural moments of the work, which nominally are the preparatory activities for the creation of a new OpenCitations Index, and the development of a mapping strategy for unequivocally identifying a citation.

More in detail, the former aspect will be addressed with a particular focus both on the general steps which are common to the implementation of every OC Index and on the specificities derived from the integration of data from the National Institute of Health. The latter aspect, which is the mapping issue, will be presented in the conclusive phase of the data ingestion, but also as a brand-new process by itself. The possibility of having the same citation expressed by two different types of identifiers is a novelty in OpenCitations infrastructure, and consequently required the ex-novo development of an adequate methodology.

In conclusion, a final paragraph introduces the approach adopted to facilitate the access to the harvested citation data, highlighting the querying possibilities made available to the potential users.

3.1 Working in a Framework: Developing a Software for an OC Index

In principle, the creation of a software to populate an index is an almost free and creative process, since the needed data can be extracted from a given source by exploiting a variety of development strategies. However, working in a framework implies following some good coding conventions, such as adopting solutions which are coherent with the rest of the system where necessary, and reusing existing code with the needed generalizations where possible.

In the specific case, the methodology followed for the implementation of NOCI software was largely inspired by the one developed and adopted for the other OC Indexes, and in particular to COCI's one. Indeed, as anticipated, CROCI data gathering strategy was conceived to overcome the impossibility to directly access some of the reference data deposited at Crossref. Accordingly, in

contrast with the full process implemented for the other two indexes, in this case a preprocessing phase aimed at retrieving additional information from the datasource is missing.

Nonetheless, for what concerns the full pipeline of an OC Index data process, the main phases are represented in Figure 2. More in detail, the procedure can be summarized as follow, in four main phases:

1) *Data Acquisition*. The ingestion workflow starts with the acquisition of the citation data from the data source. These data are the input material for the whole process, and the variety of formats and structures they come with determines the necessity of a different and specific information extraction strategy for each index. In particular, the source generally exposes its data dump either in JSON or CSV format, uses its own naming conventions to refer to the addressing and addressed entities in the citation, and provides a specific set of additional metadata together with the main information required by the OpenCitations Infrastructure, which are the identifiers of the citing and cited entities only. Accordingly, the peculiarities of the material gathered in this phase imply the necessity of case-specific adaptations for each index in the following steps of the workflow.
2) *Data Preprocess*. The data preprocessing consists of global data generation from the input material, which means exploiting the available information to create support files storing useful extra information about the citing and the cited entity involved in each citation, so to reuse it in the core step of the workflow. In particular, a fully informative citation data in OpenCitations Infrastructure is meant to include the following six elements: identifier of the addressing entity, identifier of the addressed entity, date of creation of the citation datum (which means date of publication of the work where the reference appears), timespan between the date of publication of the referenced entity and the date of publication of the referencing entity, and whether or not the two entities share the same author and/or journal of publication. Coherently, in order to properly gather these elements in the next step of the process, this preliminary phase is aimed at associating to each encountered entity its date of publication, the identifier of the journal where it was published (expressed as the journal ISSN[64]), and the identifiers of its authors (expressed as ORCID IDs[65]).
3) *Data Elaboration*. The core step of the workflow is the proper data process, which is aimed at elaborating the gained information and eventually output it in the format required for the data upload in the data store. More in detail, for each citation datum to be ingested, the

[64] «What is an ISSN? | ISSN», consultato 30 gennaio 2022, https://www.issn.org/understanding-the-issn/what-is-an-issn/.
[65] «ORCID», consultato 30 gennaio 2022, https://orcid.org/.

process starts with an internal identification, in order to determine whether or not the information is already stored in the system. Once assessed that the datum is not present in the index data store, the identifiers of both the citing and the cited entities are validated, in order to determine the resulting validity of the citation. At this point, the procedure involves the collection of the aforementioned extra information in order to assemble the full citation in the extended format required for OC Indexes. In the case one or more of the extra elements were not collected in the previous phases of the process, before storing the citation datum with missing information, the workflow implies the attempt to overcome the internal lack by resorting to external services for retrieving metadata associated with the searched identifier. At the end of this phase, the new citations are presented in the shape required for the population of an OpenCitations Index, and the harvested data are submitted in three alternative file formats, nominally CSV, Scholix and RDF (Turtle).

4) *Processed Data Upload*. The final step of the OpenCitations Indexes workflow is the population of the data store. The first data upload establishes the new index creation, and it is meant to be followed by periodical follow-ups, in order to keep the index updated.

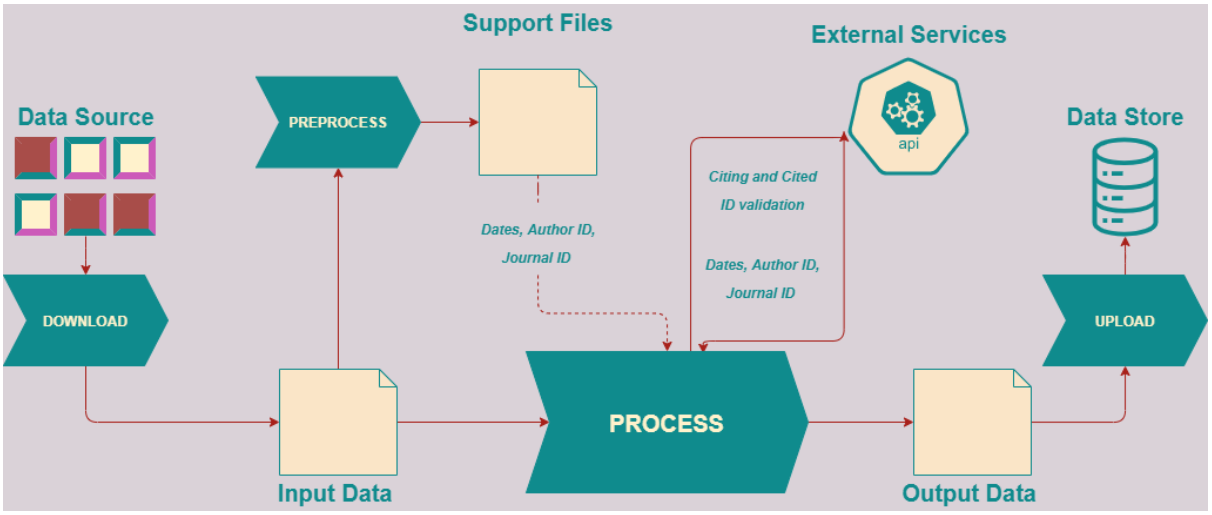

Figure 2. OC Index citation data processing. *This diagram illustrates the main phases of an OpenCitations Index data processing, from the data acquisition from the source to the harvested data upload in the data store. The intermediate phases represent the core moments of the procedure itself, including the data preprocess, where the ancillary data are extracted from the source files and stored to be reused as support material in the main process, and the main data process itself, where the information is elaborated so to be properly integrated in OpenCitations infrastructure.*

3.2 Data Acquisition Management

As briefly mentioned, the data acquisition phase is strictly related to the specific data source of the index to be populated, and therefore implies a deep analysis of the available resources aimed at preliminarily identifying the required information, its collocation in the resources and the most efficient retrieval strategies. Accordingly, the present paragraph is aimed at introducing all the research aspects related to the individuation of the available sources for the population of NOCI and the selection of relevant data with respect to the OpenCitations Indexes requirements.

3.2.1 NIH Open Citation Collection Data Source

The data source of the index NOCI is the Open Citations Collection by the National Institute of Health, whose data are provided on FigShare[66], an online open access platform serving as a research material repository.

The NIH-OCC updated data are periodically made available with the iCite Database Snapshots[67]. From October 4th, 2019 onwards, twenty-six versions of the collections have been published. At the moment of writing, the latest update was posted on January 13th, 2021, and the datasets of all the previous versions are still available[68].

The dataset consists of two main subsets of data, which nominally are the Open Citation Collection itself, exposed as CSV file, and the iCite Metadata, both exposed as CSV file and compressed JSON files. More in detail, as it is represented in Figure 3, the Collection only contains citation data expressed as pairs of citing and cited entities, represented by the couple of PMIDs identifying them. On the other hand, iCite Metadata is a collection of bibliographic records representing publications. Indeed, even if this subset of data indirectly provides the same information about citations as the Open Citation Collection, in this latter case the focus is not on the references themselves, but on the citing entities in their own right, presented with a set of annexed metadata, also including their list of references. Coherently, it is possible to trace it back to the citation data from iCite Metadata by simply associating the citing entity to each element of its reference list individually; still, iCite Metadata is not a collection of reference data, but a collection of bibliographic records.

---

[66] «figshare - credit for all your research», consultato 24 gennaio 2022, https://figshare.com/.
[67] «iCite Database Snapshot 2021-07», consultato 20 gennaio 2022, https://nih.figshare.com/articles/dataset/iCite_Database_Snapshot_2021-07/15148737.
[68] ICite, Hutchins, B. Ian, e Santangelo, George, «iCite Database Snapshots (NIH Open Citation Collection)», 2022, https://doi.org/10.35092/YHJC.C.4586573.V26.

For the purposes of the present work, only a limited number of the metadata provided for each record was necessary, and even a narrower selection was properly exploitable. Specifically, with respect to the data collected for each reference in OpenCitations Indexes, the potentially useful elements are stored in the CSV fields `"authors"`, `"year"`, `"journal"`, `"cited_by"` and `"references"`. Nonetheless, neither the authors nor the journals are identified by official identifiers, but by name. Indeed, even if it is reasonable to assume that each journal has a unique name, the same premise can not be granted for the names of people. Thus, the metadata provided about the authors of the publications were considered unreliable, and thus discarded. Further, the information about the entities citing the publication was considered redundant; therefore, only the data about the referenced entities were considered, assuming that each citation data which could be traced back exploiting the "cited_by" information is elaborated in the moment of processing the respective bibliographic records and their lists of referenced entities.

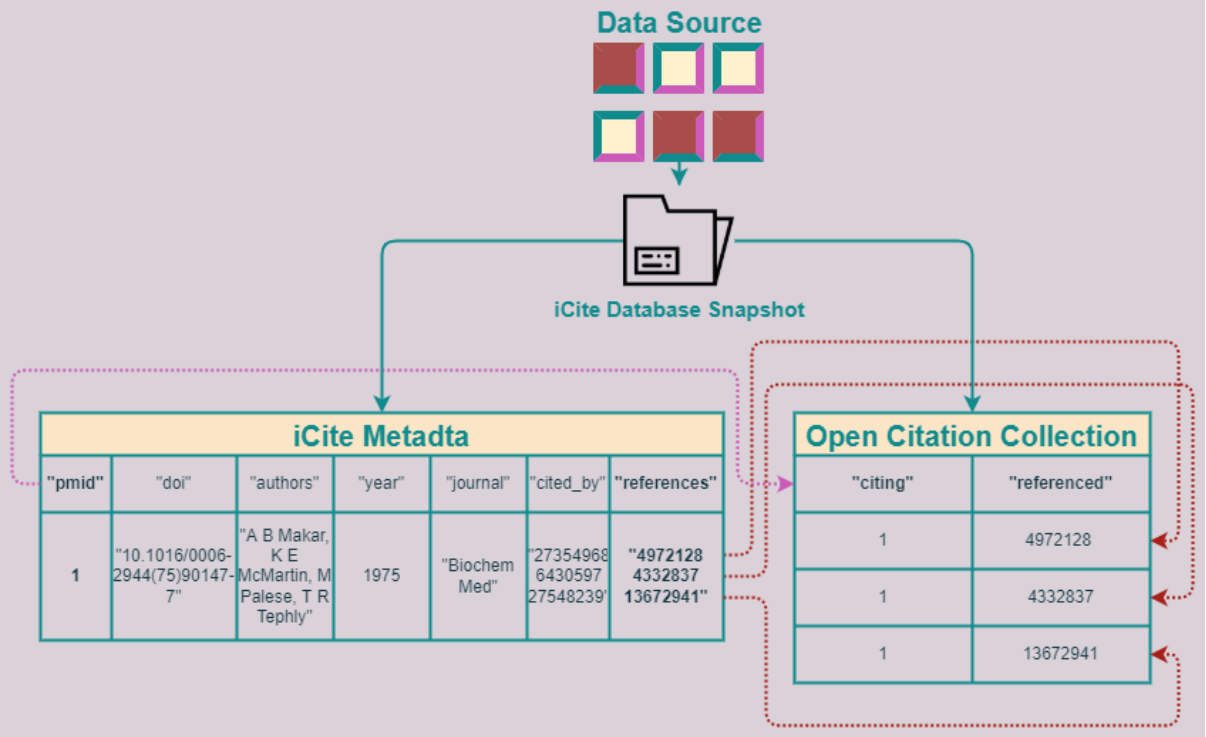

Figure 3. iCite Database. *The periodically uploaded iCite Database Snapshots are updated versions of the Open Citation Collection dataset. In particular, the two main resources provided are the Open Citation Collection itself, containing citational data represented by the couple of PMIDs of the citing and the cited entities, and iCite Metadata, collecting bibliographic records. For readability purposes, the image does not show the JSON version of iCite Metadata. Among the metadata provided for each publication represented in iCite Metadata, the crucial one for the purposes of the present work is the list of referenced entities, identified with the PMIDs of all the publications cited by each represented entity. Indeed, this information*

allows to connect the two subset of data, and to infer additional metadata for the citation links represented in Open Citation Collection from iCite Metadata. Note that the fields represented in iCite Metadata table are only the ones which provide consistent information for the present work, and also the list of referenced and referencing entities included in the fields "references" and "cited_by" is not complete. Thus, the image has illustrative purposes only. For a complete representation of the data, see the data source[69] for consulting all the versions of the data set, or check the last Snapshot[70] to analyze in detail the current structure of both iCite Metadata and Open Citation Collection.

Further, as it will be explained in the further paragraphs, another crucial detail provided among the metadata for the publications is the Digital Object Identifier. Indeed, even if this information is not available for each entity, it allowed the extraction of a relevant amount of reliable mapping data, which helped start the internal disambiguation process in OpenCitations, aimed at managing the potential duplication of data expressed both as DOI-to-DOI and PMID-to-PMID citations.

3.2.2 Retrieving PMID Metadata: NOCI External Services

For what concerns the external service exploited for PMID validation and further metadata retrieval, NOCI software relies on the utilities provided by PubMed[71]. In addition to the undisputed reliability of PubMed, one of the main strengths of this external service is the amount of data provided, accessible by the use of the three displaying modalities offered for each identifier, which are:

1) *Abstract*. This is the default result retrieved when a PMID is searched on PubMed service. This display option allows the user to have an overview on the core elements of the bibliographic entity in exam, among which the name of the author(s), the title of the publication, the name of the journal where it was published, its PMID, PMCID and DOI identifiers, and - where possible - the abstract. Further information is also provided, such as links to related articles or external resources and the lists of substances and medical subjects headings (see Figure 4).
2) *PubMed*. The PubMed display option provides a list of metadata for the searched publication, divided per field (see Figure 5). The set of official fields is defined by MEDLINE/PubMed, and a detailed description of the data expected for each field and a

[69] ICite, Hutchins, B. Ian, e Santangelo, George, «iCite Database Snapshots (NIH Open Citation Collection)», 2022, https://doi.org/10.35092/YHJC.C.4586573.V26.
[70] «iCite Database Snapshot 2021-07», consultato 20 gennaio 2022, https://nih.figshare.com/articles/dataset/iCite_Database_Snapshot_2021-07/15148737.
[71] «PubMed», consultato 31 gennaio 2022, https://pubmed.ncbi.nlm.nih.gov/.

guide to the abbreviations are provided in the official documentation[72]. This display option - among the three - was the most exploited one for retrieving metadata for the citations which are aimed at populating NOCI.

3) *PMID*. This display modality is the less informative of the three, since the response only consists of a page containing the searched identifier, in case it exists (see Figure 6), or of a failure message displayed in a redirection page, in case it doesn't. However, it successfully serves the purpose of PMID validation: indeed, in the present work this service is exploited to assess whether or not the identifiers of the two entities involved in a citation are valid.

Figure 4. NIH Pubmed, "abstract" display option. *Partial view of the Abstract default web page displayed when a PMID is searched on the PubMed platform. Abstract display option offers an overview of the selected publication, including its title, the name of the author(s), the name of the journal of publication, the date of publication, the PMID, PMCID and DOI identifiers and - where available - the abstract. Further, other additional data are provided (which are not displayed in this image), such as links to similar articles, the list*

[72] «MEDLINE®/PubMed® Data Element (Field) Descriptions», Technical Documentation (U.S. National Library of Medicine), consultato 31 gennaio 2022, https://www.nlm.nih.gov/bsd/mms/medlineelements.html.

of medical subjects headings and links to external resources, and the list of publications citing the entity under consideration. The displayed data concern the sample PMID 970516. [73]


```
PMID- 970516
OWN - NLM
STAT- MEDLINE
DCOM- 19761201
LR  - 20190514
IS  - 0090-0036 (Print)
IS  - 1541-0048 (Electronic)
IS  - 0090-0036 (Linking)
VI  - 66
IP  - 10
DP  - 1976 Oct
TI  - Quality of care given to first time birth control patients at a free clinic.
PG  - 986-7
AB  - In an empirical study of the quality of care at a free clinic, criteria for optimal
      care for female first visits for birth control were established and 100 charts were
      reviewed, 50 in April 1974 and 50 in April 1975 with an interval in between of
      in-service training accompanied by new medical forms and procedures. An encouraging
      improvement in record keeping was observed. The authors feel it is important that
      free clinics concentrate on quality as well as quantity and accessibility of care.
FAU - Grover, M
AU  - Grover M
FAU - Greenberg, T
AU  - Greenberg T
LA  - eng
PT  - Journal Article
TA  - Am J Public Health
JT  - American journal of public health
JID - 1254074
RN  - 0 (Contraceptives, Oral, Hormonal)
SB  - IM
MH  - Adult
MH  - California
MH  - Clinical Laboratory Techniques/standards
MH  - Contraceptives, Oral, Hormonal
MH  - *Family Planning Services
MH  - Female
MH  - Humans
MH  - Male
MH  - Medical History Taking
MH  - Physical Examination/standards
MH  - *Quality of Health Care
PMC - PMC1653443
EDAT- 1976/10/01 00:00
MHDA- 1976/10/01 00:01
CRDT- 1976/10/01 00:00
PHST- 1976/10/01 00:00 [pubmed]
PHST- 1976/10/01 00:01 [medline]
PHST- 1976/10/01 00:00 [entrez]
AID - 10.2105/ajph.66.10.986 [doi]
PST - ppublish
SO  - Am J Public Health. 1976 Oct;66(10):986-7. doi: 10.2105/ajph.66.10.986.
```

[73] «Quality of care given to first time birth control patients at a free clinic - PubMed», consultato 31 gennaio 2022, https://pubmed.ncbi.nlm.nih.gov/970516/.

Figure 5. NIH Pubmed, "pubmed" display option. *This web page screenshot displays the result retrieved for the sample PMID 970516, exploiting the PubMed display option.*[74] *The metadata provided concerns the searched publication, divided per field. A guide to the meaning of the fields and to the abbreviation adopted for each of them is available at the MEDLINE®/PubMed® Data Element (Field) Descriptions page.*[75]

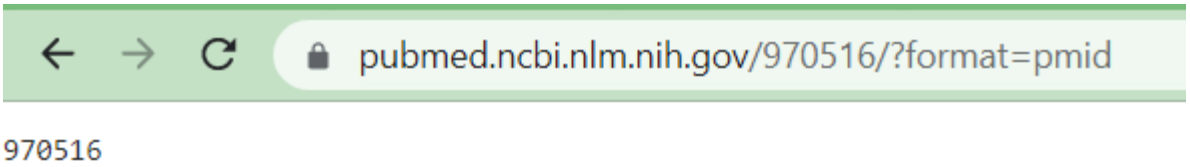

Figure 6. NIH Pubmed, "pmid" display option. *This web page screenshot shows the positive response to a request for the sample PMID 970516, obtained with the display modality PMID.*[76]

3.3 Data Processing and Integration in an Existing Infrastructure

As already discussed in the previous sections, the NOCI Index peculiarity - with respect to OpenCitations previous infrastructure - is that the entities involved in the gathered citations are identified by PMIDs.

Besides the potential data duplication issue, which will be analyzed in the next paragraphs, the introduction of a new identifier requires additional methodological adroitness.

First of all, different types of identifiers are meant to have different structures and constitutive peculiarities: as a consequence, separate strategies are needed to detect the identifier among other potential contents provided in a resource and to determine whether or not a sequence of characters can be assessed to be a either a PMID or a DOI. Coherently, also different approaches are required to correct the identifiers in case they contain evident structural inconsistencies, and specific external services are needed to assess their validity.

Indeed, having an identifier whose structure is compliant with the constraints defined by the standards for its ID type does not directly imply that the identifier is valid; which means that a virtually correct identifier could be assigned to no real entity, and thus be invalid.

Therefore, the analysis below introduces the most significant differences between PMIDs and DOIs, providing an overview of the relevant implications for OpenCitations infrastructure.

---

[74] «970516», consultato 31 gennaio 2022, https://pubmed.ncbi.nlm.nih.gov/970516/?format=pubmed.
[75] «MEDLINE®/PubMed® Data Element (Field) Descriptions», Technical Documentation (U.S. National Library of Medicine), consultato 31 gennaio 2022, https://www.nlm.nih.gov/bsd/mms/medlineelements.html.
[76] «970516», consultato 31 gennaio 2022, https://pubmed.ncbi.nlm.nih.gov/970516/?format=pmid.

3.3.1 PMID: a New ID Type in a DOI-based Infrastructure

The causes for funding OpenCitations as a DOI-based system are reasonable and largely related to the widespread use of this type of identifier. Indeed, the DOI system assures a set of guarantees which makes it particularly convenient. Specifically, among additional features, the system assures a high level of interoperability with data of different provenance, an easy retrieval of the associated entity, and the persistence of the identifier through any possible evolution of the state of the signified material. Indeed, the use of DOIs is also suggested in citation styles guidelines in order to ensure the consistency of the provided bibliographic references[77].

In spite of having emerged from the editorial field, it was projected to provide an architecture to identify materials of the digital world from the beginning, in 1997. Three years later, the system had its first official extensive use in the connection of digital articles by Crossref and the DOI identifier was standardized, although it was accepted as ISO standard only ten years later, in 2010[78].

The standard syntax of a compliant DOI name specifies the composition of a sequence of characters consisting of two main elements, nominally the prefix and the suffix, separated by a forward slash. The former element is assigned to a registering organization, and thus uniquely identifies a naming authority, while the latter is assigned by the registrant and specifically identifies the entity itself. The resulting DOI name is a string of any possible length aimed at persistently identifying the digital object it was assigned to, and whose composition bestows upon it a high informative content[79] (see Figure 7).

[77] «DOI Handbook Introduction», consultato 1 febbraio 2022, https://www.doi.org/doi_handbook/1_Introduction.html#1.6.4.
[78] Ibidem
[79] Ibidem

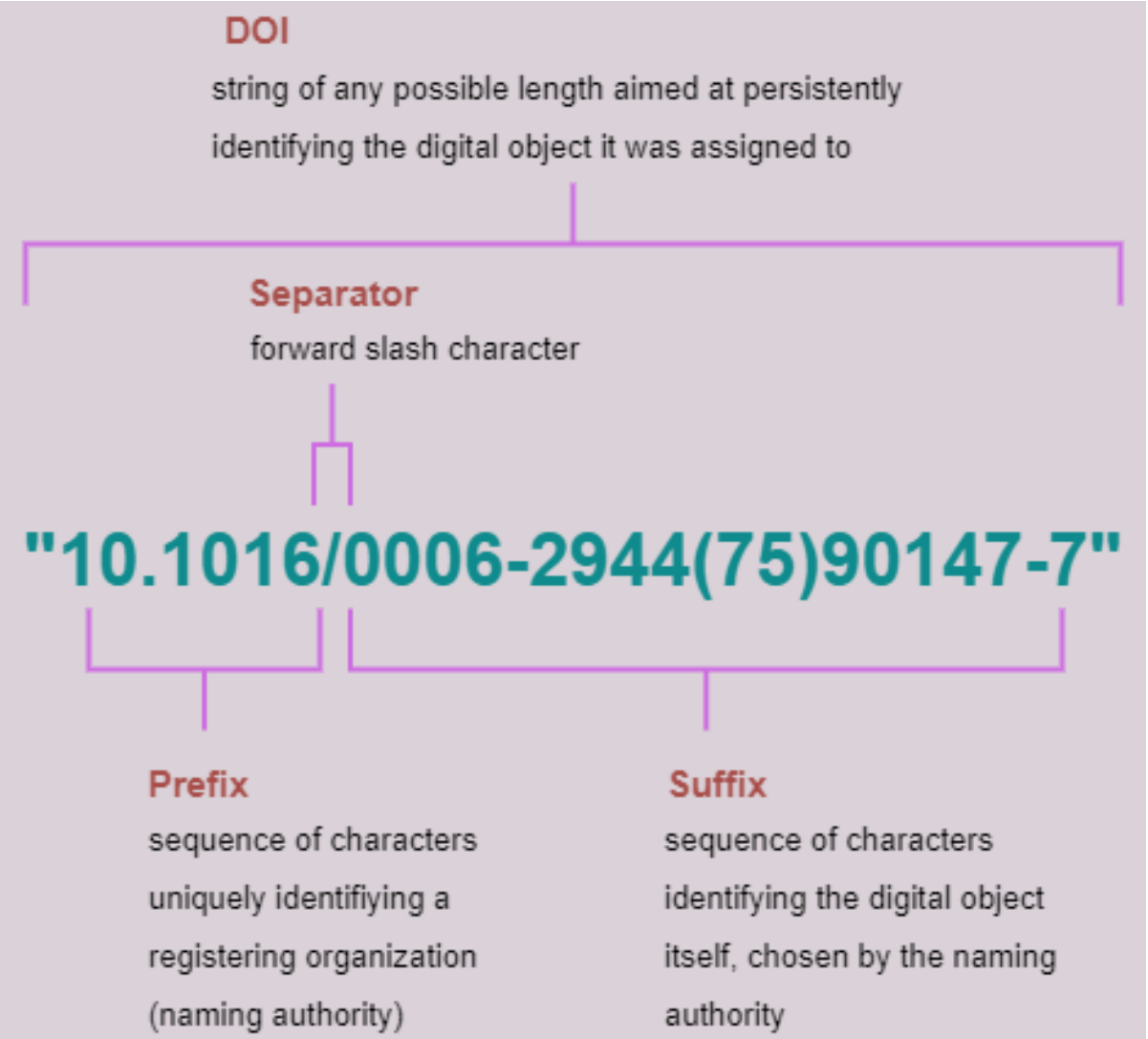

Figure 7. DOI structure. *Structure of a sample DOI with description of its components. The DOI "10.1016/0006-2944(75)90147-7" is mapped to the PMID 970516, used for illustrative purposes in the previous paragraphs of this chapter.*

The structure of the PMID, in contrast, makes this type of identifier intrinsically less informative than a DOI name. Indeed, a PMID is a NIH National Library of Medicine identifier assigned to a document indexed in PubMed[80], and since the identification approach just consists of associating the indexed entities to sequential numbers, the ratio behind the assignment does not provide further information about the document itself.

Compared with DOI names, the requirements for well-formed PMIDs are much easier to meet: unless the sequence of ciphers starts with one or more zeros or extra non-digit characters are present in the identifier, it can be assumed that the PMID form is compliant with the standard (see Figure

[80] «PMID vs PMCID: What's the Difference? – NIH Extramural Nexus», consultato 24 gennaio 2022, https://nexus.od.nih.gov/all/2015/08/31/pmid-vs-pmcid-whats-the-difference/.

8). However, just like for the DOIs, the validity of the identifier needs to be ultimately asserted by an official external service, and - in addition to that - a potential mistake in the sequence of digits is technically impossible to detect.

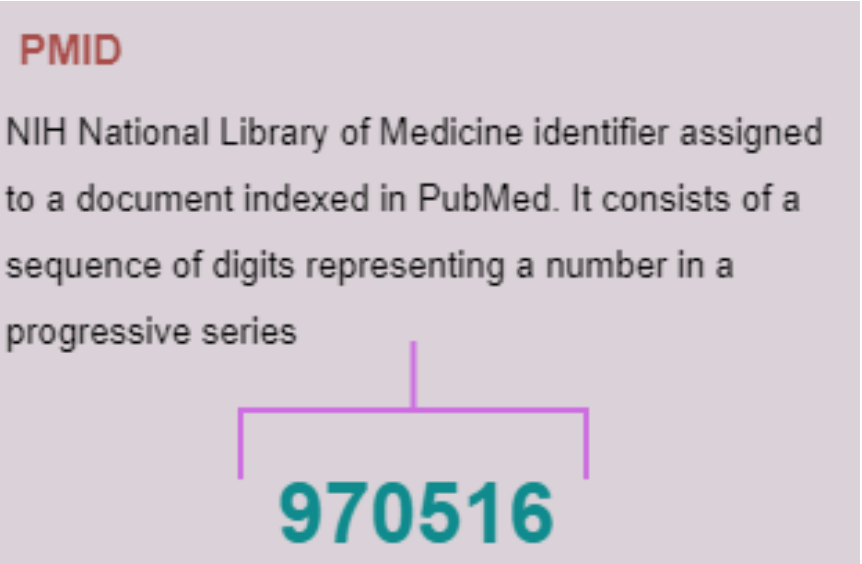

Figure 8. PMID structure. *Structure of the sample PMID 970516: the identifier consists of just a series of digits forming a number, uniquely identifying the 970516th entity registered in the PMID system.*

3.4 Overlapping Data Issue

As anticipated, the ingestion in a DOI-based system of citation data expressed with the PMID identifiers of the two entities involved in the reference implies the possibility that the same citation datum is unknowingly received twice. The solution adopted in OpenCitations to overcome the issue consists of a mapping system based on the new METAID: an identifier with an assignment ratio based on progressive numbers, aimed at associating all the identifiers of a same entity to a unique id. The resulting information is ultimately stored as RDF triples.

In contrast with the imitative approach adopted for creating NOCI Index following a methodology as coherent as possible with the pre-existent infrastructure, the elaboration of a mapping system answered to a previously unseen issue in OpenCitations, and thus resulted in the development of a brand-new process. The main passages of the implemented methodology are illustrated below under a procedural and non-technical perspective.

3.4.1 METAID Mapping System

The focus of the present mapping strategy is to avoid uncertainties about the absolute number of citation data included in OC; and this objective was pursued with a strategy for stating when multiple identifiers refer to the same entity. Indeed, the METAID assignment approach relies on the association of the new METAID to each encountered identifier, instead of just coupling the mapped ids with each other. As a result, this strategy has the advantage of implementing an easily updatable system, based on the entities themselves instead of their identifiers, thus setting a stable framework which can also rearrange its data at different times, in light of successively received mapping information.

Even if up to now the potential overlappings in OpenCitations may involve a maximum of two id types (nominally DOIs and PMIDs), the approach developed was conceived as a general-purpose methodology, in view of the integration of other new types of identifiers in future.

The diagram below (see Figure 9) has the scope of representing the main steps of the mapping workflow, which can be summarized as follows.

- *Prerequisites*. Preliminarily, the process requires some support materials, the main of which are: citation data harvested in the main index population process, the number of the last assigned METAID, id-to-METAID already generated associations, and id-to-id mapping information. For what concerns the last point, up to now the process was tested with PMID-to-DOI mapping information only, but it is implemented to be easily extended in future so as to receive mapping material expressing associations between identifiers of different types. At the first run of the process, no id-to-METAID previous associations are already stored, and - coherently - the METAID to be assigned is the first one: so, the last assigned METAID is initialized to 0. The only strictly required materials are the citation data harvested in OpenCitations. Indeed, the absence of id-to-id mapping information does not imply per se an obstacle to the process, since in this case new METAIDs are just assigned to the new encountered entities. However, the id-to-id mappings conceptually represent the core element of the methodology and thus should be provided.
- *Citing Entity Process*. For each citation to be processed, the first identifier taken into account is the one of the citing entity. This phase of the process consists of three main steps, which are: 1) the analysis of the id-to-id mapping material, in order to assess whether or not the citing id refers to the same entity as one or more other identifiers of different type; 2) a

check on the already generated id-to-METAID associations, so as to potentially retrieve the METAID(s) (if any) already paired with at least one of the ids associated to the same entity; 3) the assignment of the (same) new METAID to (all) the id(s) referring to an entity, or the re-assignment to all the ids referring to a same entity of a METAID already associated with one of the grouped identifiers. This phase has three possible outcomes, which nominally are:
    - One METAID was already assigned to one of the potentially grouped ids referring to the same entity: the METAID is maintained for the already paired id (and assigned to all the other mapped ids, if any);
    - No already assigned METAID was retrieved: a new METAID is generated ad associated to (all) the (mapped) ID(s);
    - Multiple METAIDs assigned to mapped IDs were retrieved: the METAID with the lowest value is maintained for the already paired id and (re)assigned to all other mapped ids, while the other METAIDs are invalidated. If during the process the state of the input material changed, the files are updated with the new information, such as an increment of the last assigned METAID.
- *Cited Entity Process*. Once the citing entity id and all the possible ids it is associated with have been paired with a METAID, the same process is repeated for the cited entity identifier.
- *Partial Output*. The id-to-METAID mapping data retrieved during the process are stored in output CSV files. More in detail, the harvested data are divided in two main groups: the one containing information concerning the just generated associations, and the other storing information about the pre-existing matches which were invalidated during the process.
- *Final Output*. Since OpenCitations system is based on data stored as triples, the material gathered in the previous step of the process is converted to RDF format. Indeed, at the end of the whole workflow, two RDF files are generated: the one containing the new mappings created during the process, and the other with the just invalidated mappings, i.e. the triples to be removed from the data store.
- *Upload and Deletion*. The new id-to-METAID mapping data are uploaded in the data store, while the invalidated ones are deleted from it.

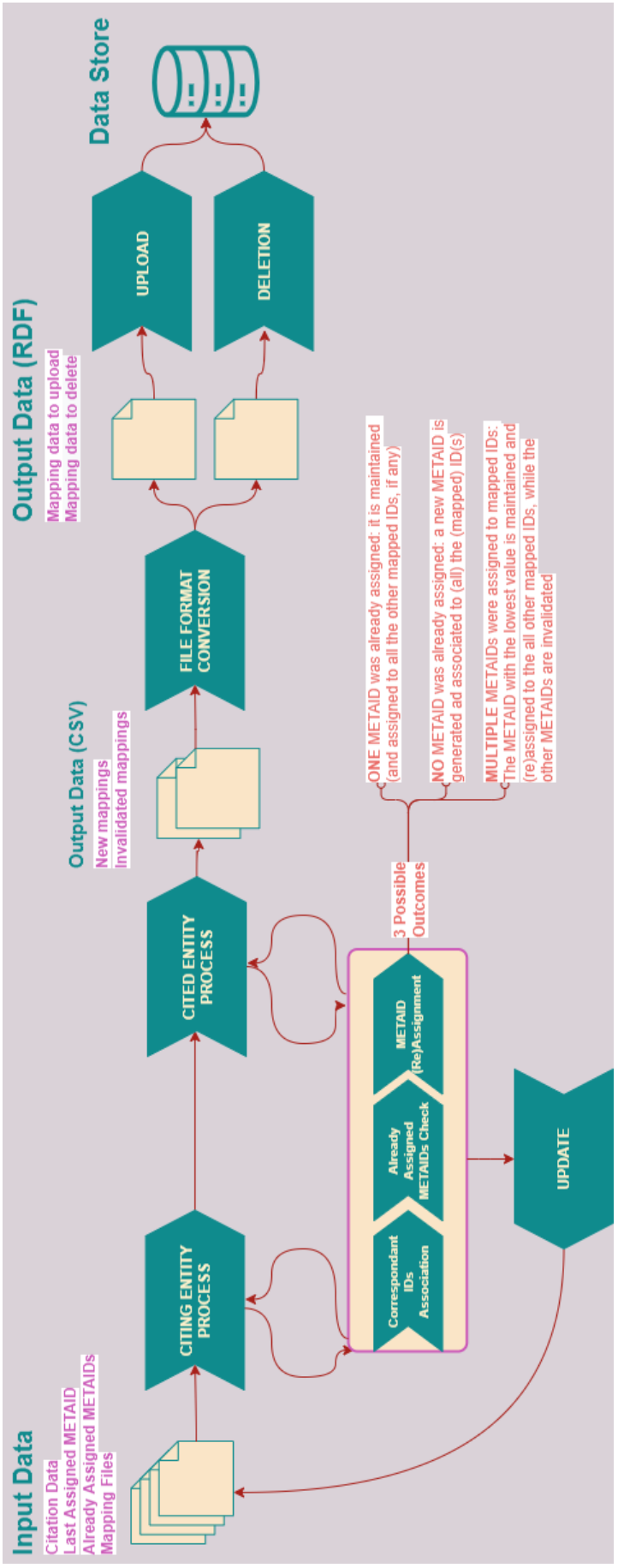
Input Data
Citation Data
Last Assigned METAID
Already Assigned METAIDs
Mapping Files
CITING ENTITY PROCESS
Output Data (CSV)
New mappings
invalidated mappings
CITED ENTITY PROCESS
Correspondant IDs Association
Already Assigned METAIDs Check
METAID (Re)Assignment
3 Possible Outcomes
ONE METAID was already assigned: it is maintained (and assigned to all the other mapped IDs, if any)
NO METAID was already assigned: a new METAID is generated ad associated to (all) the (mapped) ID(s)
MULTIPLE METAIDs were assigned to mapped IDs: The METAID with the lowest value is maintained and (re)assigned to the all other mapped IDs, while the other METAIDs are invalidated
UPDATE
Output Data (RDF)
Mapping data to upload
Mapping data to delete
FILE FORMAT CONVERSION
UPLOAD
DELETION
Data Store

Figure 9. METAID mapping process. *This diagram shows the main workflow steps of the OpenCitations mapping process, with its principal output and input materials. In particular, starting from mapping information and citation data generated in the main OpenCitations process implemented for the population of the indexes, the mapping system associates to each of the potentially multiple ids of different types referring to a specific entity a unique METAID. The same approach is adopted to map the id of the citing entity first, and then is repeated for the id of the cited one. Since new id-to-id mapping information may imply the necessity to remap an id to a different METAID from the one it was associated with in a previous phase, the process produces two output files: one with the new mappings and the other with the invalidated mappings to be removed from the data store.*

3.5 Facilitation of Data Querying

The conclusion of the present chapter is dedicated to the methodology adopted to facilitate the access to the harvested data to potentially inexperienced users. Indeed, the objectives of populating an open access resource can be met only partially if the data are accessible to the limited part of all the potential users which are acquainted with the Semantic Web technologies. Accordingly, an Application Programming Interface is configured to allow an easy access to NOCI data, on which can be performed a set of predesigned operations.

As for the process for populating NOCI, also in this case the methodology adopted was as consistent as possible with the one developed to create the APIs to access the other OC Indexes data (see Figure 10).

More in detail, the process was managed with the use of the OpenCitations application RAMOSE[81], nominally a Restful API Manager Over SPARQL Endpoints, which conveyed NOCI API a consistent appearance with all the other services generated exploiting the same application.

[81] *Restful API Manager Over SPARQL Endpoints (RAMOSE)*, Python (2018; repr., OpenCitations, 2022), https://github.com/opencitations/ramose.Restful API Manager Over SPARQL Endpoints (RAMOSE), Python (2018; repr., OpenCitations, 2022), https://github.com/opencitations/ramose.

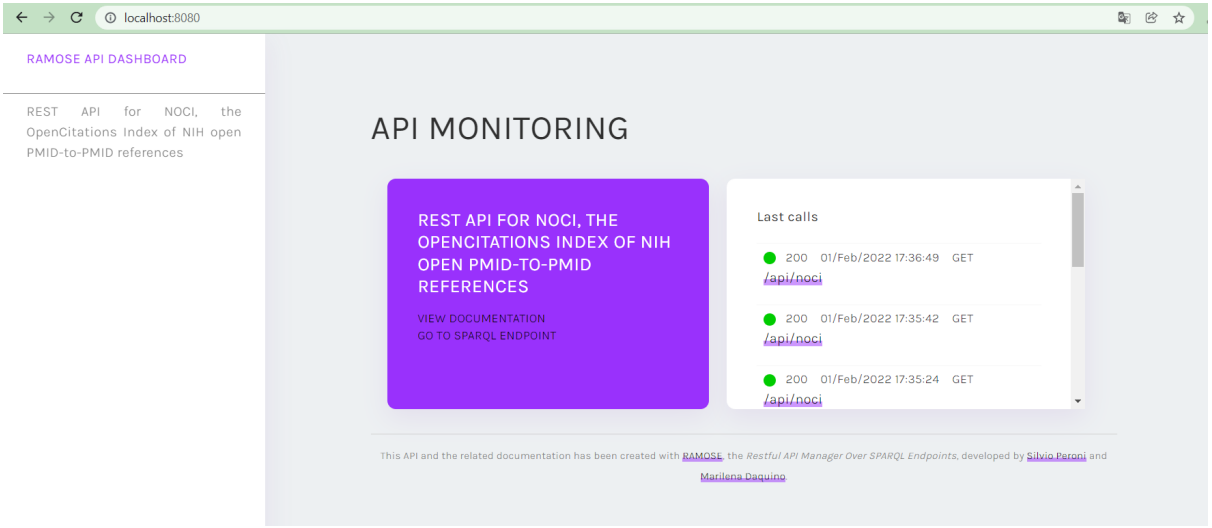

Figure 10. NOCI API service screenshot. *Partial view of the local-hosted version of NOCI API homepage. More in detail, besides a short introduction to the service and an overview on the last calls, the API gives direct access to both the documentation and the SPARQL endpoint.*

The API provides a clear description of the service, direct access to its documentation, a section dedicated to the parameters setting and the aforementioned set of pre-implemented operations. As default, all the executable operations return a response in JSON format, but it is possible to rearrange the request specifications to have it in CSV format.

Following the example methodology implemented for COCI, the operations made available also for NOCI are:

1) `/references/{pmid}`, giving access to the data for the citations addressed to publications in the reference list of the entity identified by the given PMID.
2) `/citations/{pmid}`, returning data about the citations received by the entity identified by the PMID. This is the reciprocal operation of the aforementioned `/references/{pmid}`.
3) `/citation/{oci}`, exposing metadata of the citation identified by an Open Citation Identifier (OCI), which will be presented more in detail in the next chapters.
4) `/metadata/{pmids}`, retrieving metadata concerning all the entities whose PMIDs are given in input.
5) `/citation-count/{pmid}`, which exposes the number of references received by the entity whose identifier is specified in input.

6) `/reference-count/{pmid}`, which exposes the number of references addressed by the entity whose identifier is specified in input. This is the reciprocal operation of the aforementioned `/citation-count/{pmid}`.

Among all the aforementioned operations, `/metadata/{pmids}` is the most specific one, which therefore required larger adjustments with respect to the COCI model. Indeed, this operation requires id-type-specific tasks, which needed to be implemented for PMIDs for the first time. Coherently, similar operations will be required when further types of ids will be introduced in OpenCitations infrastructure.

4. Implementation: General Overview

This chapter is dedicated to the base implementation, and thus introduces some technical aspects concerning the software structure and the development approach.
In particular, this chapter discusses how the work of this thesis has contributed to the software of the OpenCitations Index Software Library, through a presentation of the adaptations performed. A preliminary overview of the whole system with a specific focus on its core elements and concepts is part of such introductive background. Then, a particular focus on the classes' apparatus, with its dependencies and relations, is also discussed. The analysis of the base implementation is accompanied by instructions for the program launch, information on the type of input and output expected, and the development techniques adopted.

4.1 OpenCitations Index Software Library

The OpenCitations Index Software Library[82] has been implemented by OpenCitations to the creation, management and update of all the OC Indexes, i.e. COCI and CROCI. This software will be used also for the integration of our new index – NOCI.
The software is entirely developed in the Python programming language[83]. The initial commit of the code dates back to October 2019 and it has been publicly available as an OpenCitations GitHub repository ever since, at https://github.com/opencitations/index .

4.1.1 Internet Systems Consortium License

Following the principles guiding the OpenCitations infrastructure, all the scripts are released under the Open Source Initiative ISC License (ISC), which verbatim states what follows:

> Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.
>
> THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE

[82] *index*, Python (2019; repr., OpenCitations, 2021), https://github.com/opencitations/index.
[83] «Welcome to Python.Org», Python.org, consultato 3 febbraio 2022, https://www.python.org/.

AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.[84]

Coherently with the Open Source Initiative recommendation, the script files present the license accompanied by the copyright notice, including the year of creation and the name of the author of the code.

4.1.2 OpenCitations Core Concepts and Data Model

Before presenting the structural aspects concerning the OpenCitations Index Software Library, a preliminary introduction to some of the main concepts and features which are at the basis of the system is needed. More in detail, the next sections explain what an OCI Identifier is, the citation concept in Open Citation Data Model, the structure of the six-element tuple for representing reference data, and how the provenance information is generated.

OCI Identifier

The Open Citation Identifier (OCI) is the globally unique and persistent identifier for references introduced by OpenCitations (Peroni e Shotton 2019). OCIs are aimed at deciphering the couple of identifiers involved in a citation datum, thus providing homogeneity in the representation of references regardless of their provenance and id type[85].

The OCI structure is highly informative: the citation is identified by two numerical sequences separated by a dash, representing the two entities involved in the citation, each of which starts with a brief series of ciphers delimited by two zeros identifying the source of the bibliographic reference. Each OCI identifier is preceded by the lowercase prefix “oci”, followed by a colon, which introduces the proper OCI identifier (see Figure 11).

These bibliographic resource identifiers are unequivocal, and thus all OpenCitations recorded references have an OCI associated.

[84] «ISC License (ISC) | Open Source Initiative», consultato 3 febbraio 2022, https://opensource.org/licenses/ISC.
[85] «Identifiers.org». Consultato 3 febbraio 2022. https://registry.identifiers.org/registry/oci.

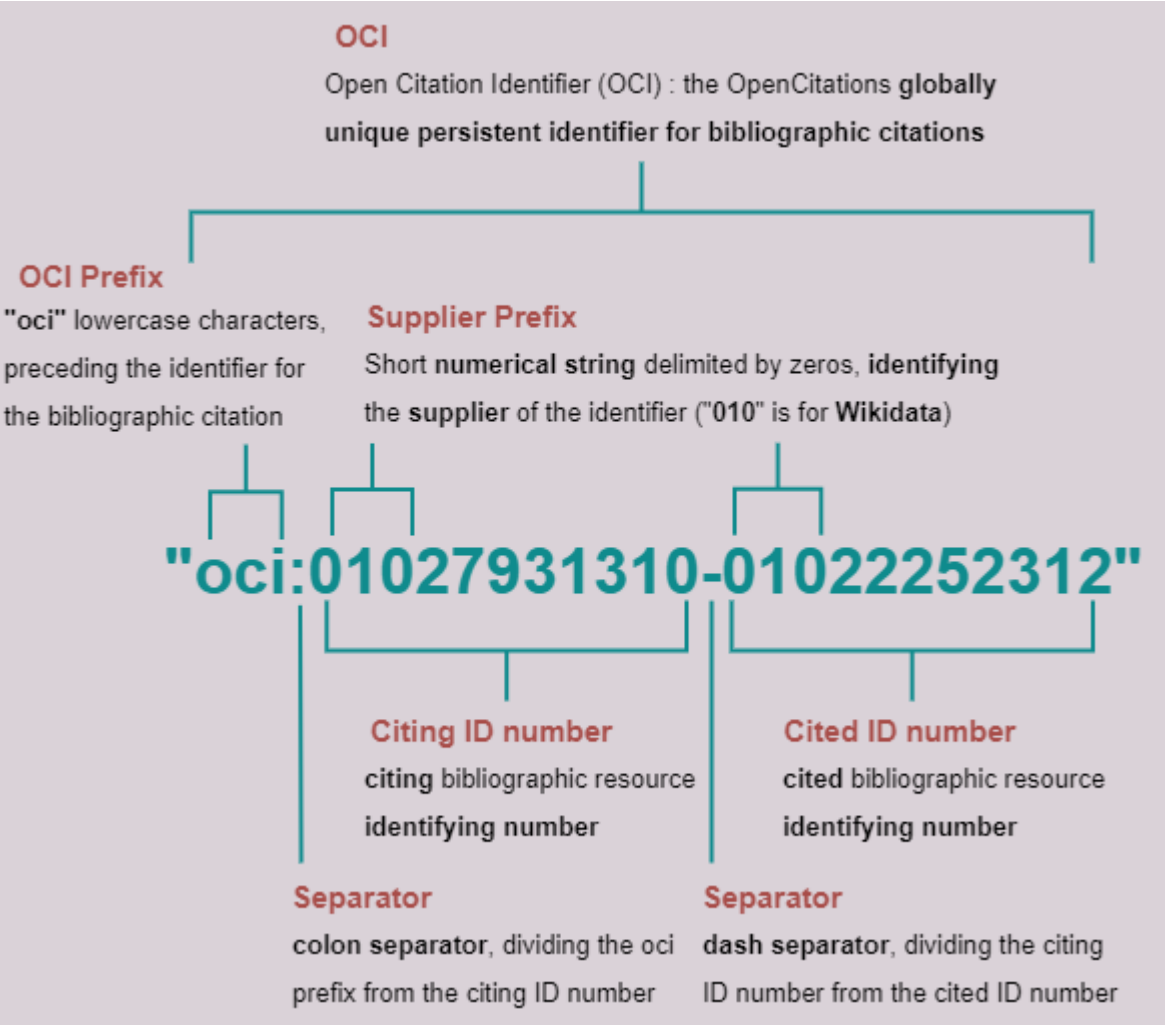

Figure 11. OCI structure. *OpenCitations Identifier (OCI). The image explains the OCI structure, exploiting the sample identifier "oci:01027931310-01022252312". More in detail, the lowercase "oci" prefix is followed by a colon, which precedes the two numerical sequences separated by a dash, identifying the citing and the cited entity. In each of the two numerical sequences the first group of ciphers delimited by zeros identifies the supplier prefix: in this case "010", which identifies Wikidata.*

The Python code file `oci.py`, is both included in the Index Software Library[86] and in the OCI Software Library[87]. The script primarily allows the OCI reconstruction for a given citation data based on the couple of identifiers of the entities involved in the reference, the validation of an OCI given as input, and the retrieval of the data concerning a given citation.
The script can be called from terminal by filling out and running the command below (see Script 1)

[86] *index*, Python (2019; repr., OpenCitations, 2022), https://github.com/opencitations/index/blob/738a6a1244731a812b1bec15646c6bbba26dedcb/index/citation/oci.py.
[87] *Open Citation Identifier*, Python (2018; repr., OpenCitations, 2022), https://github.com/opencitations/oci/blob/dca43c2976d1b96e23100b4dcdd98ffed4bcca14/oci.py.

```
python oci.py -o <OCI> -f <FORMAT>
```

Script 1. OCI launch command. *Command for calling oci.py from the terminal, specifying an input OCI identifier and the desired format of the response. To run the command, <OCI> needs to be replaced with an OCI identifier, while <FORMAT> can receive a value among the following: 'csv', 'json', 'json-ld', 'turtle', 'xml', 'n-triples', which will determine the file format of the output.*

Further, the same query can be performed through an online service available at the OCI page[88] in the OpenCiations website, specifying an OCI in the search box and selecting one of the possible file formats for the response, among the ones listed in the scrollbar on the right (see Figure 12). The response for the sample query for the OCI "oci:01027931310-01022252312" is shown in the Script 2 below.

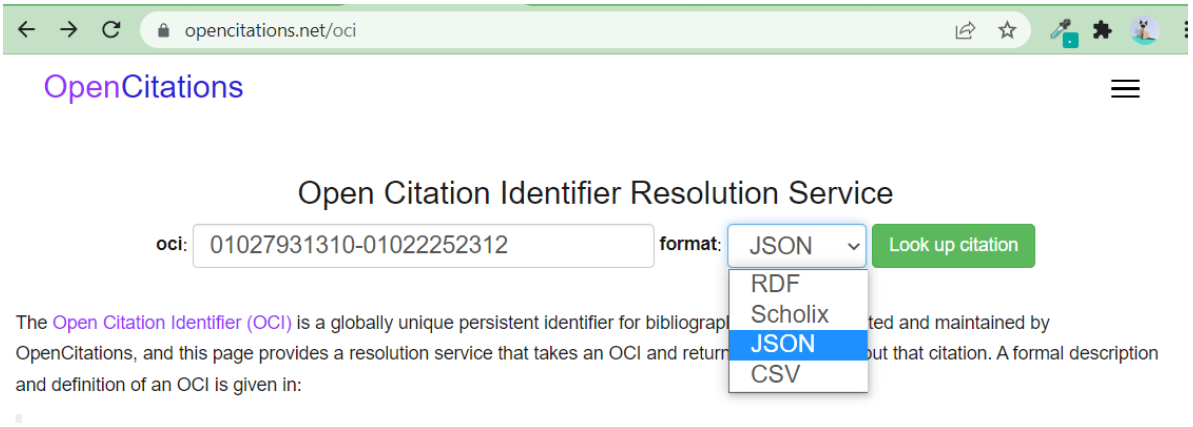

Figure 12. OCI metadata web service. *OpenCitations tool for OCI metadata retrieval, available at https://opencitations.net/oci . In the OCI online resolution service, the user specifies an OCI in input by typing it in the search box and selects a file format from the scrollbar on the right.*


```
{
    "timespan": "P1Y4M5D",
    "creation": "2012-07-06",
    "oci": "01027931310-01022252312",
    "journal_sc": "no",
    "cited": "Q22252312",
    "citing": "Q27931310",
    "author_sc": "no"
}
```


[88] «OpenCitations - OCI», consultato 3 febbraio 2022, https://opencitations.net/oci.

Script 2. OCI JSON response. *Response in JSON format to the sample query for the OCI "oci:01027931310-01022252312". The key-value pairs of the dictionary besides the OCI itself are the elements of the six-value tuple used in OpenCitations to represent reference data with the related metadata.*

Six-element Tuple

As mentioned in the script description above (see Script 2), the OpenCitations Index Software process is based on six elements describing a citation, derived from the citation source and - where necessary - integrated with the recourse to external API services. The six elements are:

1) `citing`: the identifier of the citing entity, expressed in the normalized format and validated through the API specific services for the id type in question.
2) `cited`: the identifier of the cited entity, expressed in the normalized format and validated through the API specific services for the id type in question.
3) `creation`: the date of the creation of the citation. In particular, since the citation is generated within the citing entity itself, it can be generalized that the creation of the citation corresponds with the date of publication of the referencing entity. This information is expressed in the format YYYY-MM-DD, which means the four digits representing the year, a dash separator, the two-digit (ranging from 01 to 12) representing the month, another dash separator, and the two-digit representing the day (which range from 01 to 31).
4) `timespan`: the time elapsed between the date of publication of the cited entity and the date of publication of the citing entity.
5) `journal_sc`: additional information stating whether or not the citing and the cited entity share the same journal of publication, i.e. if the citation is a self-citation, with respect to the journal of publication.
6) `author_sc`: additional information stating whether or not the citing entity and the cited entity share the same author, i.e. if the author of the citing entity referenced one of its previous works.

At the code level, the extraction of the six-element tuple is implemented by the various source-specific instantiations of the `CitationSource` class of the OpenCitations Index Software, providing the method `get_next_citation_data`, whose documentation states that:

> This method returns the next citation data available in the source specified. The citation data returned is a tuple of six elements: citing id (string), citingdate (string, or None if unknown),

cited id (string), cited date (string or None if unknown), if it is a journal self-citation (True = yes, False = no, None = do not know), and if it is an author self-citation (True = yes, False = no, None = do not know). If no more citation data are available, it returns None.[89]

For a correct functioning of the process, the only two strictly required elements are the identifiers of both the citing and the cited entity. Indeed, in case the data concerning either the date of creation of the citation or the timespan between the publications of the two entities is missing, these fields can stay empty. In the two remaining features, i.e. the self-citation fields, if an element is missing the values used in these fields, then the corresponding value is `None`, which means the absence of the needed data to state whether it is a self-citation or not. Also in this latter case, the generated data is still compliant with the OC format.

However, when the original data source does not provide all the required data, the missing information may be retrieved from pre-compiled support files - if present - , or from using external API services.

Citations in OpenCitations Data Model

In 2016, OpenCitations officially defined for the first time its own Data Model – the OpeCitations Data Model (OCDM).The aim was formalizing a description of the data contained in the OpenCitations Corpus. In the following years, the adoption of the OCDM for the description of new datasets led to its expansion: indeed, the 2019 release of the data model included new types of entities for defining also in-text reference pointers, the correspondent citations and their function, which means a description of the reason of the citation («OpenCitations - Data Model» s.d., Daquino et al. 2020, 449, Peroni e Shotton 2020, 434-435).

The most relevant aspect in OpenCitations Data Model - which mainly concerns the work of this thesis - is that the bibliographic references are conceived as first-class data entities accompanied by the related metadata (such as their textual context, the role of the involved agents, the provenance information), which can be released under a CC0 license. In particular, the references are expressed as instances of the Citation Typing Ontology[90] `class:Citation`. Among the related subclasses, `cito:AuthorSelfCitation` and `cito:JournalSelfCitation` should be mentioned

[89] *index*, Python (2019; repr., OpenCitations, 2022), https://github.com/opencitations/index/blob/738a6a1244731a812b1bec15646c6bbba26dedcb/index/citation/citationsource.py.
[90] «CiTO, the Citation Typing Ontology», consultato 6 febbraio 2022, https://sparontologies.github.io/cito/current/cito.html.

because of their crucial value with respect to the citation metadata required for the OpenCitations Indexes (Daquino et al. 2020, 449 - 452).

Further, the terms of the OCDM are also gathered in the OpenCitations Ontology (OCO)[91], which exploits terms from external ontologies, such as the Semantic Publishing and Referencing (SPAR) Ontologies.

As it is represented in Figure 13 , the OCDM provides the possibility to store data about: the published bibliographic resources (class `fabio:Expression`), the format of their embodiment (class `fabio:Manifestation`), the citations appearing in their reference list (class `biro:BibliographicReference`), the agents related to the resources (class `foaf:Agent`), the roles of the aforementioned agents with respect to the resources (class `pro:RoleInTime`), the citations (class `cito:Citation`) linking the involved resources, and the various associated identifiers (class `datacite:Identifier`) («OpenCitations - Data Model» s.d.).

More in detail, the table below (see Table 1) describes the OCDM representational requirements, i.e. OpenCitations Data Model citation properties, which are identified by sequential codes from P1 to P8.

ID	Description
P1	Definition of the type of citation, which provides some information about the citation nature, such as whether or not it is a self citation.
P2	The bibliographic entities annex metadata, such as the type of the entity, the associated identifiers, the authors, and the dates of publication.
P3	The bibliographic reference information, generally retrieved from the reference list of the citing entity.
P4	The identifiers of the in-text reference pointers detected in the text of the citing entity.
P5	The combined occurrence of the aforementioned in-text reference pointers in each in-text reference pointer list.

[91] «OCO, the OpenCitations Ontology», consultato 6 febbraio 2022, https://opencitations.github.io/ontology/current/ontology.html.

P6	The identifiers of structural elements, such as paragraphs, to define the location of the reference pointer in the text.
P7	The reference function of the citation, i.e. a description of the reason why the cited entity is referenced by the citing one.
P8	The provenance information concerning the procedure of citation extraction, which means who are the agents involved, which are the data sources and the dates of extraction.

Table 1. OCDM representational requirements. *OCDM table of representational requirements, i.e. the OpenCitations Data Model citation properties, numbered from P1 to P8 (Daquino et al. 2020, 450).*

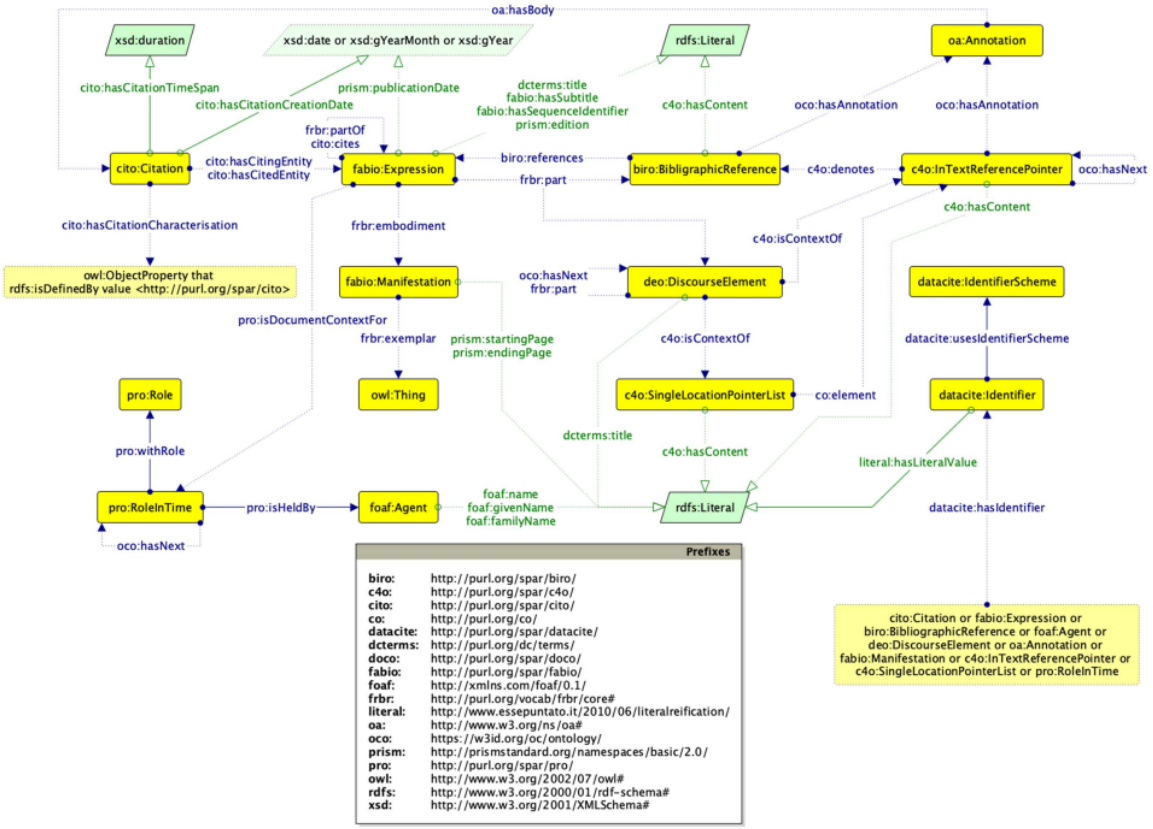

Figure 13. OCDM diagram. *The image contains a diagram describing the core aspects of the OCDM represented with the main classes and properties of the OpenCitations Ontology* (Daquino et al. 2020, 451; Peroni e Shotton 2020, 434).

Information of Provenance

As already mentioned, the information of provenance is the eighth OpenCitations representational requirement (see Table 1), and it includes data about the procedure of citation extraction, such as the identity of the involved agents, the exploited data sources, and the dates of extraction.

More in detail, the provenance of a resource is instantiated by all the agents (including people as well as institutions and other types of entities) and the actions which can generate, distribute or have an impact on the data. The importance of this information is strictly connected with the trustability of the data in question, since it permits the determination of responsibilities and the credits attribution for all the aspects concerning the data lifecycle (Peroni e Shotton 2020, 436).

The provenance information is one of the OpenCitations core citation properties, and thus it is provided for all the entities of OC datasets. This type of information is particularly precious, since it is exploited to track data evolution over time by generating snapshots, which consist of descriptions of the status of an entity at a specific moment. The strategy adopted with respect to the generation of this description required an extension of the Provenance Ontology[92] by Lebo, Sahoo and McGuinnes with SPARQL facilities (Peroni e Shotton 2020, 436), in particular by adopting an approach for tracking evolutions in RDF data with RDF provenance statements (Peroni, Shotton, e Vitali 2016, 1).

4.1.3 Software Library Functions and Structure

This section is dedicated to an overview on the global organization of the OpenCitations Index Software Library and the main functionalities implemented in the software, with a particular focus on the integrations required by NOCI data inclusion in the system.
At the moment of writing, the official repository of the software is available on GitHub at https://github.com/opencitations/index , while the extension implemented in this thesis is provided as a fork of the repository[93].
The OC Index Software Library is organized in packages (see Figure 14): the directory containing the whole software is named `index`, and includes both the file containing the code for launching the whole process for each of the three indexes (i.e., `cnc.py`) and a subfolder which is also named `index`, containing all the rest of the software material. Inside the `index` subfolder, all the scripts are organized in directories, with respect to their functionalities, among which the core ones are:

[92] «PROV-O: The PROV Ontology», consultato 7 febbraio 2022, https://www.w3.org/TR/prov-o/.
[93] Moretti Arianna, *index*, Python, 2021, https://github.com/ariannamorettj/index.

- *Citation management*: the management of all those aspects concerning the OpenCitations specificities in the citation data handling, such as the six element tuple for expressing the reference data, and the generation and validation of OCI identifiers.
- *Data storage and general management*: to create and fill out the output files in all the required formats, opening and updating input files, accessing the information already stored in the system, as well as uploading the final processed data in the triplestore.
- *Identifiers management*: handling the part of the workflow concerning the normalization, the validation and the API-specific requests to retrieve additional information for each type of identifier managed in OC Infrastructure, including both the ones for bibliographic entities (nominally PMID and DOI) and the ones for journals and authors (nominally ISSN and ORCID).
- *Source-specific data retrieval*: which implies the implementation of source-specific resource-finder tools to collect metadata for the processed entities (such as the ISSN of the journal of publication, the ORCID of the authors and the date of publication), exploiting different APIs, with respect to the type of identifier. The data sources currently managed by the OpenCitations Index Software Library are the ones provided by: Crossref (API service exploited: https://api.crossref.org/works/), DataCite (API service exploited: https://api.datacite.org/dois/), ORCID (API service exploited: https://pub.orcid.org/v2.1/search?q=), and the recently integrated Pubmed by the National Institute of Health (API service exploited: https://pubmed.ncbi.nlm.nih.gov/).
- *Data source-specific management for each index*: which consists of retrieving information both from the specific citation source for each index to be populated and from the API external services, so as to store extra information in support files to be reused in the main process for indexes' population.
- *Mapping*: which implies exploiting already processed citations and support material mapping different types of identifiers, in order to unequivocally identify each citation datum in OpenCitations, even if it was expressed using different id types and thus ingested in two different indexes.

Additional support functionalities are also implemented for the generation of statistics data and the management of specific file formats or data structures, but they were not included in the overview because those are side utilities with respect to the main process workflow, which is the focus of this section.

In addition, testing tools are implemented in order to verify the correct functioning of each of the procedures provided by the software.

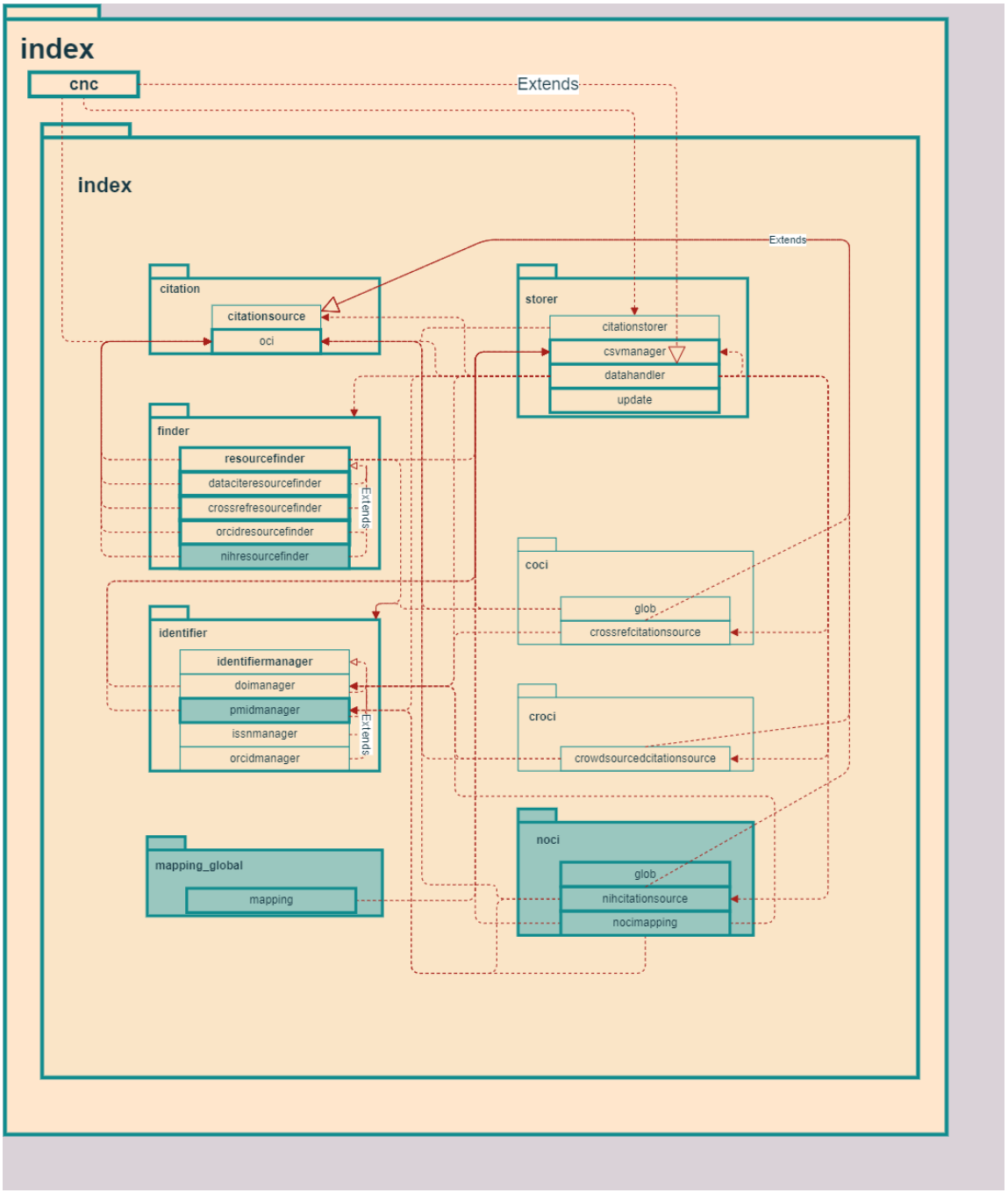

Figure 14. OC Index software UML packages diagram. *The image represents the packages diagram of the OpenCitations Index Software Library. brand-new packages and files created for the purposes of the present thesis work are colored in blue, while the material which was modified has a bolder border line. The arrows ending with a full pointer represent generic re-use of the material contained in the pointed element by the pointing one, while the arrows ending with an empty pointer represent a relation of extension. The main package is named index, and includes both the core file containing the script to launch the whole process (i.e. cnc.py) and a main subdirectory containing all the rest of the material, which is named index as well. Inside the index subfolder, the scripts are organized in packages with respect to their functions. In particular, each of the packages of the three indexes (i.e. coci, croci, and noci) contains the script for managing the*

specific citation source; further, coci and noci folders also contain a glob file each one, for the creation of support files for the main process. The identifier folder contains all the tools to manage the identifiers, which nominally are: ISSN, ORCID, DOI, and - after the software expansion - PMID. The finder directory groups all the resource-finder tools to collect metadata for the processed citations, exploiting different API-specific approaches. The directory citation stores the files whose scripts implement the main functionalities related to the OpenCitations specificities in the citation data handling, i.e.: the management of the extraction from a given source of the relevant information to complete the six element tuple, the creation of the tuple, and the generation and validation of OCI identifiers from the identifiers of the entities involved in a citation. The storer folder collects the scripts concerning the management of data storage at each phase of the workflow, which implies creating, opening and filling out output files in various formats, as well as uploading the processed data in the triplestore. In conclusion, the mapping_global package contains the code for processing the mapping information. The diagram is available both on draw.io and on GitHub[94]*. Note that the diagram is largely representative but not complete: for an exhaustive understanding of the OC Index Software Library visit the GitHub repository at* https://github.com/opencitations/index .

4.1.4 Code Launch and File Formats

The process launcher for the citation management of each OpenCitations index (i.e.: COCI, CROCI and NOCI) is called from outside of the `index` subfolder, by executing the file `cnc.py` using the specific parameters required with respect to the index to manage.
A sample call for running the NOCI process from the terminal is presented below (see Script 3).

```
python cnc.py -ib "https://pubmed.ncbi.nlm.nih.gov/" -b
"https://w3id.org/oc/index/noci/" -c "csv" -i
"index/test_data/citations_partial_pmid.csv" -doi
"index/noci_test/pmid.csv" -orcid "index/noci_test/orcid.csv" -date
"index/noci_test/date.csv" -issn "index/noci_test/issn.csv" -l
"index/test_data/lookup_full.csv" -d "index/noci_test" -px "0160" -a
"https://w3id.org/oc/index/prov/ra/1" -s
"https://doi.org/10.35092/yhjc.c.4586573.v16" -sv "OpenCitations Index:
NOCI" -type "pmid" -v
```

Script 3. NOCI call example. *Example of a call for NOCI process from terminal.*

[94] The diagram is published on draw.io and available at https://viewer.diagrams.net/?tags=%7B%7D&highlight=0000ff&edit=_blank&layers=1&nav=1#G15aA2RQV1bRVqDqu2i2kXqTT4koVUUAaD , where it can be accessed after logging in to a Google account. The PDF version of the diagram can be downloaded from GitHub, at https://github.com/ariannamorettj/OCIndexSoftwareLibraryExtension/blob/main/Packages%20Diagram.pdf .

The command envisages the base URLs for the identifiers of the entities involved in the citations and of the dataset, which in this case are: "https://pubmed.ncbi.nlm.nih.gov/" and "https://w3id.org/oc/index/noci/". The class of data source for accessing specific information to create and process citations is defined in the parameter field "`-c`" : in this case it is "csv", which stands for `CSVFileCitationSource` (which will be presented with the whole classes system in the section *NOCI introduction in OpenCitation index*). The "`-i`" parameter defines the input file or directory for the Python code execution: in this case, the file is "`index/test_data/citations_partial_pmid.csv`". Support files with already stored data, if present, are provided in input as well, in order to avoid unnecessary calls to the APIs: "`-doi`", "`-orcid`", "`-date`", "`-issn`" respectively expose information about the already validated identifiers, the associations between an identifier of a publication and the ORCID identifier of its author, the date of publication, and the associations between an identifier of a publication and the ISSN identifier of the journal of its publication. Further, the lookup file (specified in the parameter field "`-l`") provides instructions for encoding and decoding the identifiers of the entities involved in the citations, while the parameter "`-d`" specifies the storage directory of the CSV files already added in the Index, concerning data and provenance. The prefix identifying the provider of the citation data is expressed in the parameter field "`-px`", which stands for "prefix": in this case we have "0160", associated with the National Institute of Health. Coherently, also the URLs of both the processing or providing agent for the citation data and the source of data extraction are exposed, which in this case are "https://w3id.org/oc/index/prov/pa/1" and "https://doi.org/10.35092/yhjc.c.4586573.v16". Moreover, the name of the service providing access to citation data is stated in the parameter field "`-sv`".

The implementation of the present thesis project led to the introduction of a new argument, which is "`-type`": the values expected are either "doi" or "pmid", and its presence is now necessary, in order to properly split the process of identifier management with respect to the id type.

Finally, if the optional "`-v`" parameter is included in the command, messages concerning the process states are printed on screen. Similarly, the optional inclusion of the "`-na`" argument, which stands for "no api", gives the user the possibility to choose if running the process without relying on API external services, just recovering the required data from the support files, if any.

The output material of the process is stored in two separate folders, the one containing the citation data expressed as OCIs with the related six elements required for the population of OC Indexes, and the other containing the provenance snapshots for the citations, providing information about the

agent, the source and and the date of creation, together with a possible description, the status update message (e.g: “Creation of the citation”), and the information about the possible invalidation.
The most used file format in OpenCitations software is the CSV, indeed both the outputted citation data and the provenance information are primarily provided in this format. Further, both groups of data are also expressed in RDF, which is the format required for the triplestore population, but only the processed citation data are also provided in Scholix format.

4.1.5 NOCI, a new OpenCitations index

The ingestion of new data from the National Institute of Health Open Citation Collection led to the necessity of both creating appropriate tools for the population of NOCI and managing PMIDs in a DOI-based system. At the implementation level, this meant intervening in the existing software scripts with the aim of generalizing the DOI-specific functionalities, but also developing new code from scratch.

Table 2 provides an overview on the main methodological aspects encountered, the main issues raised, and the adopted solutions.

Methodological Aspect	Implementation Issues	Implementation Solutions
Data acquisition from NIH OCC	The data source provides two CSV files and compressed JSONs. One of the CSVs contains PMID-to-PMID citation data, while the other and the compressed JSON store the same bibliographic data about the entities involved in the citations. However, the source files don’t provide all the information required for the population of OpenCitations Indexes.	Development of the citation source class `NIHCitationSource` as an instance of the superclass `CSVFileCitationSource` and creation of an id type-specific glob file to extract from the datasource additional support data to be reused in the main process.

Data integration in a DOI-based infrastructure	Data provided by the new source is not compliant with the id-type managed by the OpenCitations Index Software Library.	Development of software extensions and adaptation, so as to generalize the id type-specific passages of the process and manage the validation and the API data retrieval for PMIDs.
	The data retrieved from the new data source and from the API services for the new id type are not provided in the formats already managed by the software (nominally JSON or CSV).	Adaptation of the existing code in view of implementing the same functionalities, with respect to the different data formats and expositions.
Overlapping data issue	The ingestion of citation data expressed with a new type of identifier introduced the possibility of duplicating the same citation data as DOI-to-DOI and PMID-to-PMID reference.	Development of scripts for the implementation of a mapping system aimed at associating to a unique id the correspondent DOI and PMID. This implied the need for a new id type which is unequivocal in Open Citations: the METAID. The association ID-to-METAID is then expressed in rdf format, so that through a SPARQL query it is possible to find if two reference data expressed with dishomogeneous identifiers refer to the same citation.
	Presence of PMID-to-DOI mapping raw information for some of the bibliographic entities described in the iCite Metadata dataset	Development of a source-specific tool to exploit this initial set of data in order to create mapping files, to be exploited in the general mapping process.

Facilitation of data querying	Poor familiarity of the potential users with the SPARQL language, used to query the RDF triplestores of the OpenCitations Indexes.	Implementation by means of RAMOSE of an Index REST API, which facilitates the querying process, providing the possibility to download the responses in JSON and CSV formats.

Table 2. NIH-OCC data integration methodology. *This table provides a general overview on the main methodological aspects of the process of integration of the NIH-OCC set of data in OpenCitation Infrastructure. In particular, the structure of the table presents the methodological aspects together with the issues encountered in its implementation and the adopted solution.*

The four main methodological aspects which led to the necessity of either extending and modifying the pre-existing scripts or developing new code from scratch are the ones listed below:

- Data acquisition from NIH OCC;
- Data integration in a DOI-based infrastructure;
- Overlapping data issue;
- Facilitation of data querying.

The last point was addressed by extending the OpenCitations API repository[95], by providing the REST API specification for the OpenCitations datasets used by RAMOSE, with the NOCI configurations. In particular, the NOCI configuration settings are declared in the HF file script `noci_v1.hf`, which was developed following the approach adopted for the other indexes, in order to make available the same set of operations: `references`, `citations`, `citation`, `metadata`, `citation-count`, `reference-count` (see Chapter 3, *Methodology, Facilitation of data querying*). Further, the file `indexapi.py` was extended with a function for managing the metadata retrieval for PMIDs exploiting the PubMed format display option of the NIH service available at https://pubmed.ncbi.nlm.nih.gov .

For what concerns the NIH-OCC data acquisition, integration, and the overlapping data issue, the implementation solutions led to the generation of five new classes in the OC Index Software Library and to the adaptation of nine of the pre-existing ones (see Figure 15 and Figure 16).

In particular, the brand-new classes are:

1) `NIHCitationSource`: This class instantiates the superclass `CSVFileCitationSource`. Its aim is retrieving from the citation source the data

[95] *api*, Python (2018; repr., OpenCitations, 2021), https://github.com/opencitations/api.

required for completing the six-element tuple representing the citation datum. However, the main dataset provided by the National Institute of Health Open Citation Collection only consists of a two-columns CSV file, storing the PMID of the addressing entity (i.e., "`citing`") and the PMID of the cited entity (i.e., "`referenced`"). Thus, the only two elements retrieved from the source are the identifiers of the entities involved in the citation. `NIHCitationSource` plays the same role as other source-specific classes already defined in OpenCitations: `CrowdsourcedCitationSource` (for CROCI) and `CrossrefCitationSource` (for COCI).

2) `NIHResourceFinder`: This class instantiates the API resource finder superclass (i.e., `ApiIDResourceFinder`) for the PMIDs, exploiting the external service provided by the National Institute of Health available at https://pubmed.ncbi.nlm.nih.gov/ to retrieve data about the ISSN of the journal, the ORCID of the authors, and the date of publication associated with the processed identifier. The other classes performing the same functionality for other types of identifiers are: `DataCiteResourceFinder`, `ORCIDResourceFinder`, and `CrossrefResourceFinder`.
3) `PMIDManager`: The `PMIDManager` class is an instance of the `IdentifierManager` class, just like the `DOIManager`, the `ORCIDManager`, and the `ISSNManager`. The objective of the id manager classes is to normalize, validate and assess the existence of a given identifier. The `PMIDManager` was developed to perform the exact same functions as the `DOIManager`, while exploiting a PMID-specific validation service, available at https://pubmed.ncbi.nlm.nih.gov/ .
4) `Nocimapping`: This class was specifically developed for extracting and processing the PMID-to-DOI mapping raw information provided in the iCiteMetadata subset of Open Citations Collection, with the aim of creating CSV mapping files to be used as input material in the main mapping process.
5) `Mapping`: This class implements the whole process of id-to-METAID mapping, exploiting id-to-id mapping files and the output data elaborated for the population of the indexes. The outputs of `Mapping` class are two RDF files, the one storing id-to-METAID new association data to be uploaded in the triplestore, and the other containing the id-to-METAID associations invalidated during the process because of new mapping data, if any. The triples contained in the file storing the id-to-METAID invalidated data are to be removed from the triplestore.

In addition, the nine pre-existing classes which were modified are listed below:

1) `OCIManager`: The variables `doi_type = "doi"` and `pmid_type = "pmid"` were added to `OCIManager` as class variables.
2) `DataHandler`: This class gathers the data and creates citations. In this case, `"noci" : NIHCitationSource` was included among the other source classes, which nominally are: `"csv": CSVFileCitationSource`, `"crossref": CrossrefCitationSource`, and `"croci": CrowdsourcedCitationSource`.
3) `FileDataHandler`: This class extends the `DataHandler`, it was integrated with the `id_type` parameter to call the correct identifier manager with respect to the type of identifiers used in a specific data source, i.e., the PMID for the National Institute of Health, and the DOI in all the other cases.
4) `ResourceFinder`: the `id_type parameter` has been added to the class constructor. This modification allows the call of the correct identifier manager with respect to the type of the identifier.
5) `APIIDResourceFinder`: the name of the class was refactored from `APIDOIResourceFinder` to `APIIDResourceFinder`, and the methods implemented for the identifier normalization and validation were adapted to split the process with respect to the nature of the id.
6) `DataCiteResourceFinder`: this class extends `APIIDResourceFinder` in order to permit data retrieval using DataCite API service. The parameter `id_type=OCIManager.doi_type` was added to the arguments of the class constructor.
7) `ORCIDResourceFinder`.This class extends `APIIDResourceFinder` in order to permit data retrieval using the ORCID API service. The parameter `id_type=OCIManager.doi_type` was added to the arguments of the class constructor.
8) `CrossrefResourceFinder`. This class extends `APIIDResourceFinder` in order to permit data retrieval using the Crossref API service. The parameter `id_type=OCIManager.doi_type` was added to the arguments of the class constructor.
9) `CSVManager`. This class provides methods for easily managing data access and storage in CSV files. It was extended with a function to substitute the values of specified fields (given

as parameters). The developed method implements a functionality to correct id-to-METAID pairings on CSV files, when the METAID associated with an id is invalidated and a new one is assigned, because of new mapping information.

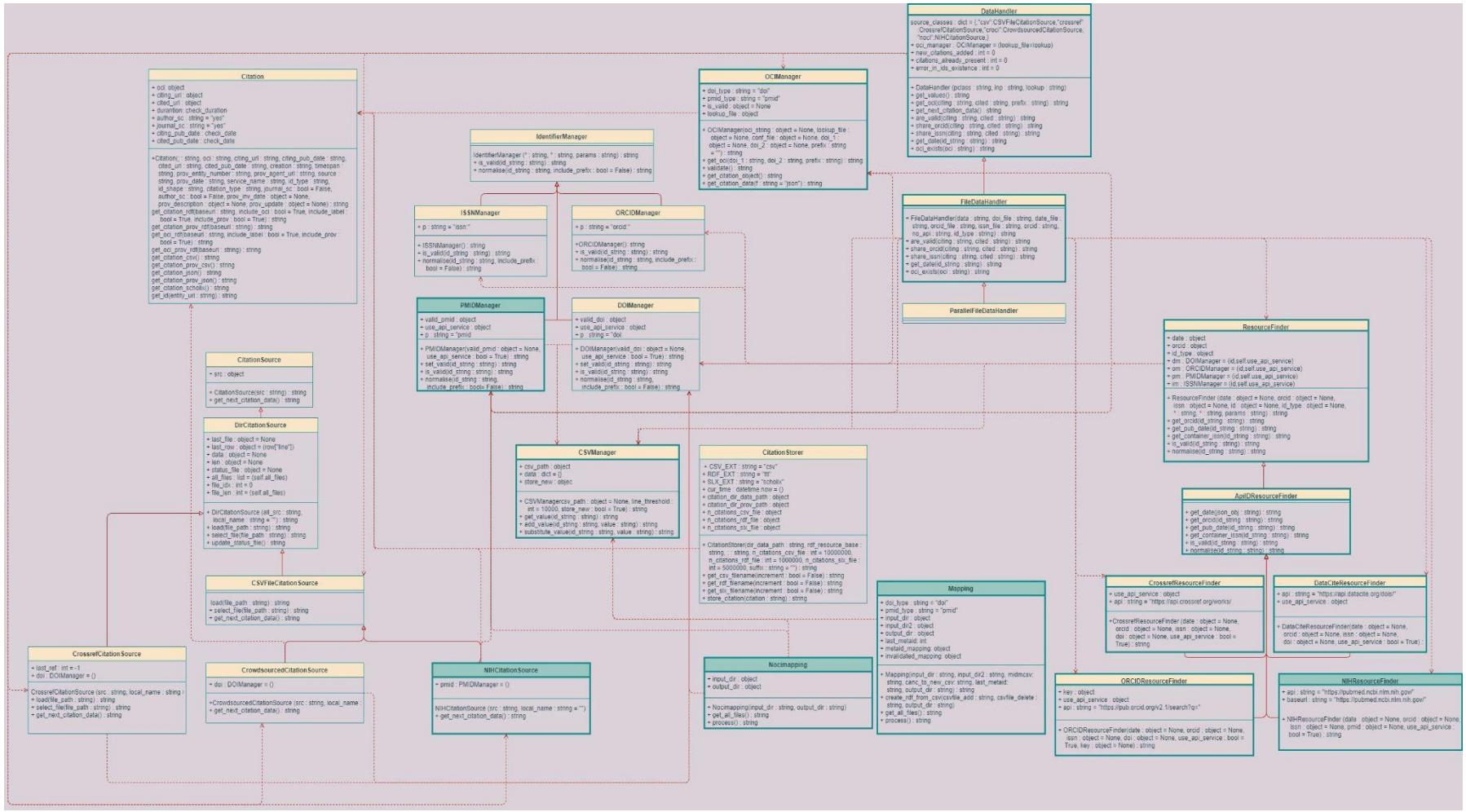
DataHandler
source_classes : dict = {"csv":CSVFileCitationSource,"crossref":CrossrefCitationSource,"croci":CrowdsourcedCitationSource,"nioci":NIHCitationSource,}
+ oci_manager : OCIManager = (lookup_file=lookup)
+ new_citations_added : int = 0
+ citations_already_present : int = 0
+ error_in_ids_existence : int = 0
+ DataHandler (pclass : string, inp : string, lookup : string)
+ get_values() : string
+ get_oci(citing : string, cited : string, prefix : string) : string
+ get_next_citation_data() : string
+ are_valid(citing : string, cited : string) : string
+ share_orcid(citing : string, cited : string) : string
+ share_issn(citing : string, cited : string) : string
+ get_date(id_string : string) : string
+ oci_exists(oci : string) : string
Citation
+ oci: object
+ citing_url : object
+ cited_url : object
+ duration: check_duration
+ author_sc : string = "yes"
+ journal_sc : string = "yes"
+ citing_pub_date : check_date
+ cited_pub_date : check_date
+Citation(: string, oci : string, citing_url : string, citing_pub_date : string, cited_url : string, cited_pub_date : string, creation : string, timespan : string, prov_entity_number : string, prov_agent_url : string, source : string, prov_date : string, service_name : string, id_type : string, id_shape : string, citation_type : string, journal_sc : bool = False, author_sc : bool = False, prov_inv_date : object = None, prov_description : object = None, prov_update : object = None) : string
get_citation_rdf(baseurl : string, include_oci : bool = True, include_label : bool = True, include_prov : bool = True) : string
get_citation_prov_rdf(baseurl : string) : string
get_oci_rdf(baseurl : string, include_label : bool = True, include_prov : bool = True) : string
get_oci_prov_rdf(baseurl : string) : string
get_citation_csv() : string
get_citation_prov_csv() : string
get_citation_json() : string
get_citation_prov_json() : string
get_citation_scholix() : string
get_id(entity_url : string) : string
OCIManager
+ doi_type : string = "doi"
+ pmid_type : string = "pmid"
+ is_valid : object = None
+ lookup_file : object
+ OCIManager(oci_string : object = None, lookup_file : object = None, conf_file : object = None, doi_1 : object = None, doi_2 : object = None, prefix : string = "") : string
+ get_oci(doi_1 : string, doi_2 : string, prefix : string) : string
+ validate() : string
+ get_citation_object() : string
+ get_citation_data(f : string = "json") : string
IdentifierManager
IdentifierManager (* : string, * : string, params : string) : string
+ is_valid(id_string : string) : string
+ normalise(id_string : string, include_prefix : bool = False) : string
ISSNManager
+ p : string = "issn:"
+ ISSNManager() : string
+ is_valid(id_string : string) : string
+ normalise(id_string : string, include_prefix : bool = False) : string
ORCIDManager
+ p : string = "orcid:"
+ORCIDManager(): string
+ is_valid(id_string : string) : string
+ normalise(id_string : string, include_prefix : bool = False) : string
PMIDManager
+ valid_pmid : object
+ use_api_service : object
+ p : string = "pmid
+ PMIDManager(valid_pmid : object = None, use_api_service : bool = True) : string
+ set_valid(id_string : string) : string
+ is_valid(id_string : string) : string
+ normalise(id_string : string, include_prefix : bool= False) : string
DOIManager
+ valid_doi : object
+ use_api_service : object
+ p : string = "doi
+ DOIManager(valid_doi : object = None, use_api_service : bool = True) : string
+ set_valid(id_string : string) : string
+ is_valid(id_string : string) : string
+ normalise(id_string : string, include_prefix : bool = False) : string
FileDataHandler
- FileDataHandler(data : string, doi_file : string, date_file : string, orcid_file : string, issn_file : string, orcid : string, no_api : string, id_type : string) : string
- are_valid(citing : string, cited : string) : string
+ share_orcid(citing : string, cited : string) : string
- share_issn(citing : string, cited : string) : string
- get_date(id_string : string) : string
+ oci_exists(oci : string) : string
ParallelFileDataHandler
ResourceFinder
+ date : object
+ orcid : object
+ id_type : object
+ dm : DOIManager = (id,self.use_api_service)
+ orm : ORCIDManager = (id,self.use_api_service)
+ prm : PMIDManager = (id,self.use_api_service)
+ irm : ISSNManager = (id,self.use_api_service)
+ ResourceFinder (date : object = None, orcid : object = None, issn : object = None, id : object = None, id_type : object = None, * : string, *_ : string, params : string) : string
+ get_orcid(id_string : string) : string
+ get_pub_date(id_string : string) : string
+ get_container_issn(id_string : string) : string
+ is_valid(id_string : string) : string
+ normalise(id_string : string) : string
CitationSource
+ src : object
+ CitationSource(src : string) : string
+ get_next_citation_data() : string
DirCitationSource
+ last_file : object = None
+ last_row : object = (row["line"])
+ data : object = None
+ len : object = None
+ status_file : object = None
+ all_files : list = (self.all_files)
+ file_idx : int = 0
+ file_len : int = (self.all_files)
+ DirCitationSource (all_src : string, local_name : string = "") : string
+ load(file_path : string) : string
+ select_file(file_path : string) : string
+ update_status_file() : string
CSVManager
+ csv_path : object
+ data : dict = {}
+ store_new : objec
+ CSVManager(csv_path : object = None, line_threshold : int = 10000, store_new : bool = True) : string
+ get_value(id_string : string) : string
+ add_value(id_string : string, value : string) : string
+ substitute_value(id_string : string, value : string) : string
CitationStorer
+ CSV_EXT : string = "csv"
+ RDF_EXT : string = "ttl"
+ SLX_EXT : string = "scholix"
+ cur_time : datetime.now = ()
+ citation_dir_data_path : object
+ citation_dir_prov_path : object
+ n_citations_csv_file : object
+ n_citations_rdf_file : object
+ n_citations_slx_file : object
+ CitationStorer(dir_data_path : string, rdf_resource_base : string, : string, n_citations_csv_file : int = 10000000, n_citations_rdf_file : int = 1000000, n_citations_slx_file : int = 5000000, suffix : string = "") : string
+ get_csv_filename(increment : bool = False) : string
+ get_rdf_filename(increment : bool = False) : string
+ get_slx_filename(increment : bool = False) : string
+ store_citation(citation : string) : string
AgentIDResourceFinder
+ get_date(json_obj : string) : string
+ get_orcid(xml_string : string) : string
+ get_pub_date(id_string : string) : string
+ get_container_issn(id_string : string) : string
+ is_valid(id_string : string) : string
+ normalise(id_string : string) : string
CSVFileCitationSource
load(file_path : string) : string
+ select_file(file_path : string) : string
+ get_next_citation_data() : string
Mapping
+ doi_type : string = "doi"
+ pmid_type : string = "pmid"
+ input_dir : object
+ input_dir2 : object
+ output_dir : object
+ last_metaid: int
+ metaid_mapping: object
+ invalidated_mapping: object
+ Mapping(input_dir : string, input_dir2 : string, inidrncsv : string, canc_to_new_csv: string, last_metaid: string, output_dir : string) : string
+ create_rdf_from_csv(csvfile_add : string, csvfile_delete : string, output_dir : string)
+ get_all_files() : string
+ process() : string
CrossrefResourceFinder
+ use_api_service : object
+ api : string = "https://api.crossref.org/works/
+CrossrefResourceFinder (date : object = None, orcid : object = None, issn : object = None, doi : object = None, use_api_service : bool = True) : string
DataCiteResourceFinder
+ api : string = "https://api.datacite.org/dois/"
+ use_api_service : object
+ DataCiteResourceFinder(date : object = None, orcid : object = None, issn : object = None, doi : object = None, use_api_service : bool = True) :
CrossrefCitationSource
+ last_ref : int = -1
+ doi : DOIManager = ()
CrossrefCitationSource (src : string, local_name : string = "") : string
+ load(file_path : string) : string
+ select_file(file_path : string) : string
+ get_next_citation_data() : string
CrowdsourcedCitationSource
+ doi : DOIManager = ()
+CrowdsourcedCitationSource (src : string, local_name : string = "")
+ get_next_citation_data() : string
NIHCitationSource
+ pmid : PMIDManager = ()
NIHCitationSource (src : string, local_name : string = "")
+ get_next_citation_data() : string
Nocimapping
+ input_dir : object
+ output_dir : object
+ Nocimapping(input_dir : string, output_dir : string)
+ get_all_files() : string
+ process() : string
ORCIDResourceFinder
+ key : object
+ use_api_service : object
+ api : string = "https://pub.orcid.org/v2.1/search?q="
+ ORCIDResourceFinder(date : object = None, orcid : object = None, issn : object = None, doi : object = None, use_api_service : bool = True, key : object = None) : string
NIHResourceFinder
+ api : string = "https://pubmed.ncbi.nlm.nih.gov/"
+ baseurl : string = "https://pubmed.ncbi.nlm.nih.gov/"
+ NIHResourceFinder (date : object = None, orcid : object = None, issn : object = None, pmid : object = None, use_api_service : bool = True) : string

Figure 15. OC Index software UML classes diagram. *Preview of the expanded version of the Unified Modeling Language (UML) Classes Diagram of OpenCitations Index Software Library. In accordance with the UML syntax, a class is represented by a box divided horizontally in three sectors: the first one from the top containing the name of the class, the central one exposing its attributes, and the last one on the bottom containing the main operations implemented by the class. However, for representational reasons, this diagram contains only a subset of core methods and attributes for each class. The relation between the instantiation of a class and its superclass is represented by an arrow with an empty triangular pointer, originating from the instance and pointing to the superclass. The reuse of a class' material by another class is represented by a dotted arrow, ending with a regular pointer. In order to improve the readability of the diagram, the links between classes were represented by continuous lines, instead of the standard dashed lines, which are also used to describe relations of code reuse. With the aim of visually describing the code extensions, the classes which were generalized or modified for the present thesis work are highlighted with a bolder border line, while the brand-new classes are coloured in blue. The diagram is available on draw.io and GitHub*[96].

[96] The diagram is published on draw.io and available at https://viewer.diagrams.net/?tags=%7B%7D&highlight=0000ff&edit=_blank&layers=1&nav=1#G1xcVqbXYomMvV3f7ni4Cv8pYZwmCzVCdP, where it can be accessed after logging in to a Google account. The PDF version of the diagram can be downloaded from GitHub, at https://github.com/ariannamorettj/OCIndexSoftwareLibraryExtension/blob/main/Classes%20diagram.pdf .

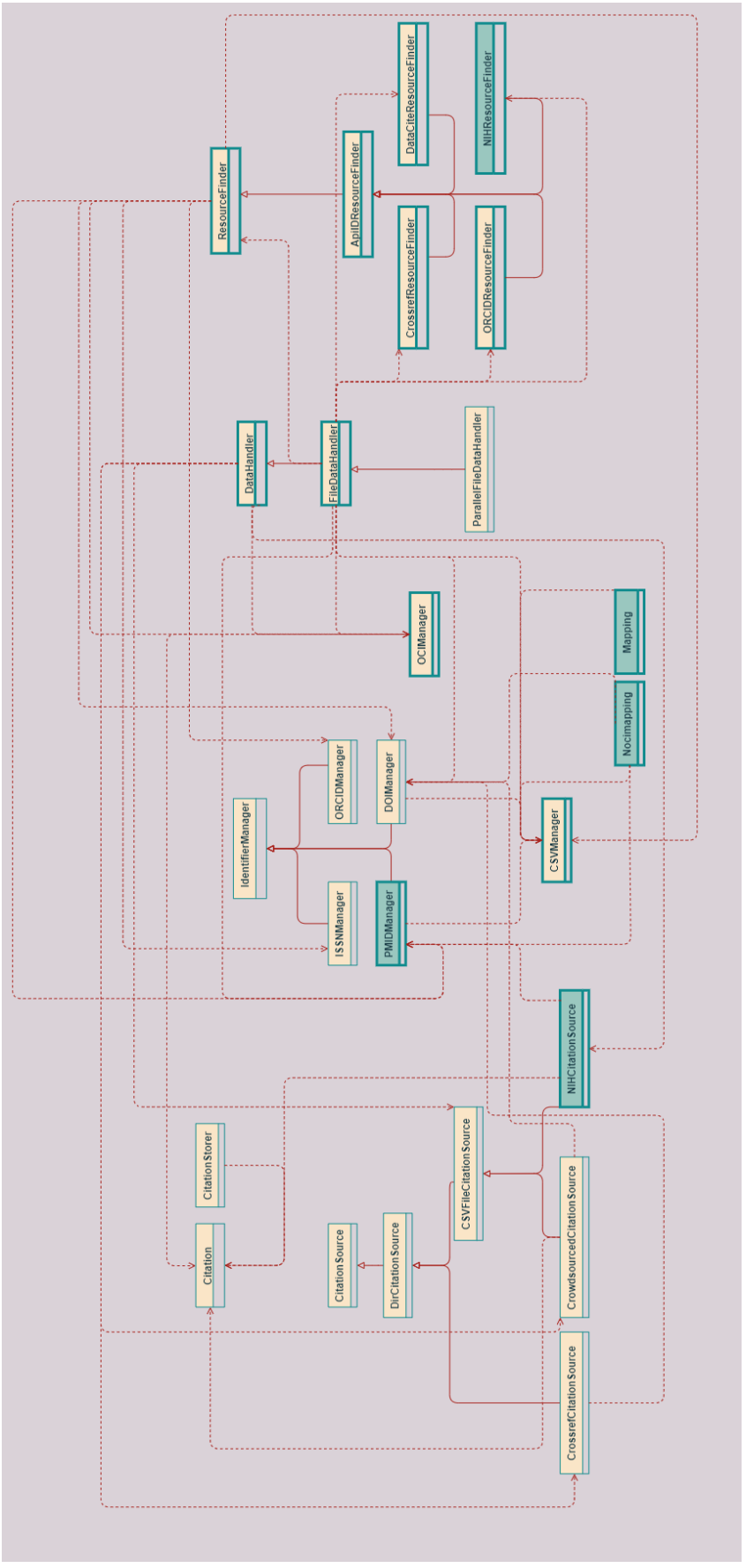
ResourceFinder
DataCiteResourceFinder
NIHResourceFinder
AnilDResourceFinder
CrossrefResourceFinder
ORCIDResourceFinder
DataHandler
FileDataHandler
ParallelFileDataHandler
OCIManager
Mapping
Nocimapping
ORCIDManager
DOIManager
IdentifierManager
CSVManager
ISSNManager
PMIDManager
NIHCitationSource
CitationStorer
CSVFileCitationSource
Citation
CitationSource
DirCitationSource
CrowdsourcedCitationSource
CrossrefCitationSource

Figure 16. OC Index software UML classes diagram (reduced). *Preview of the reduced version of the Unified Modeling Language (UML) Classes Diagram of OpenCitations Index Software Library. In accordance with the UML syntax, the relation between the instantiation of a class and its superclass is represented by an arrow with an empty triangular pointer, originating from the instance and pointing to the superclass. The reuse of a class' material by another class is represented by a dotted arrow, ending with a regular pointer. In order to improve the readability of the diagram, the links between classes were represented by continuous lines, instead of the standard dashed lines, which are also used to describe relations of code reuse. With the aim of visually describing the code extensions, the classes which were generalized or modified for the present thesis work are highlighted with a bolder border line, while the brand-new classes are coloured in blue. The diagram is available on draw.io and GitHub*[97].

4.2 Development Technique: Test Driven Development

With respect to the practices adopted for the development of the pre-existing OpenCitations Index Software Library, the extensions/adaptations implemented for the ingestion of the NOCI Index followed the Test Driven Development Technique[98].

The peculiarity of this technique is that the development of the test case precedes the development of the code to be tested, in order to guarantee that the implemented functionalities actually meet the development purposes defined before coding. In addition to that, the correctness of the code is repeatedly verified by running the code: indeed, the procedure is iterative, and requires further tests to be developed and passed each time the code is extended for implementing new functionalities. Coherently, even when the code is minimally modified, all the tests are run again, in order to check that they keep being passed successfully. The strength of this approach is that the focus is always maintained on the projectual purposes of the code development and on the obtainment of prefixed requisites for each functionality implemented[99].

Conceptually, the approach consists of five phases (see Figure 17). which nominally are:

[97] The diagram is published on draw.io and available at https://viewer.diagrams.net/?tags=%7B%7D&highlight=0000ff&edit=_blank&layers=1&nav=1#G1agfNw31rJYh115nPerFzCI2goo1VUFYz , where it can be accessed after logging in to a Google account. The PDF version of the diagram can be downloaded from GitHub, at https://github.com/ariannamorettj/OCIndexSoftwareLibraryExtension/blob/main/Reduced%20Classes%20Diagram.pdf .

[98] 'Breve Introduzione al Test Driven Development CompetIT - Sviluppo Software', accessed 5 February 2022, https://www.competit.eu/article/breve-introduzione-al-test-driven-development.

[99] 'Introduction to Test Driven Development (TDD)', accessed 14 February 2022, http://www.agiledata.org/essays/tdd.html#WhatIsTDD.

1. *Test Case Development*. A test case is developed in order to test a functionality which has not been implemented yet, in order to define what is supposed to be the output of the process to be developed, given a specific input.
2. *Test Run and Failure*. At this point of the development workflow, the test fails, since the code whose functionality it is supposed to be tested has not been developed yet.
3. *Code Implementation*. The actual functionality to be tested is implemented.
4. *Test Run and Passing*. If the functionality was developed correctly, the test implemented at the beginning of the process should be passed.
5. *Refactoring*. At the end of the process, the code is re-edited in order to improve the quality. This phase implies new test runs at each modification of the code.

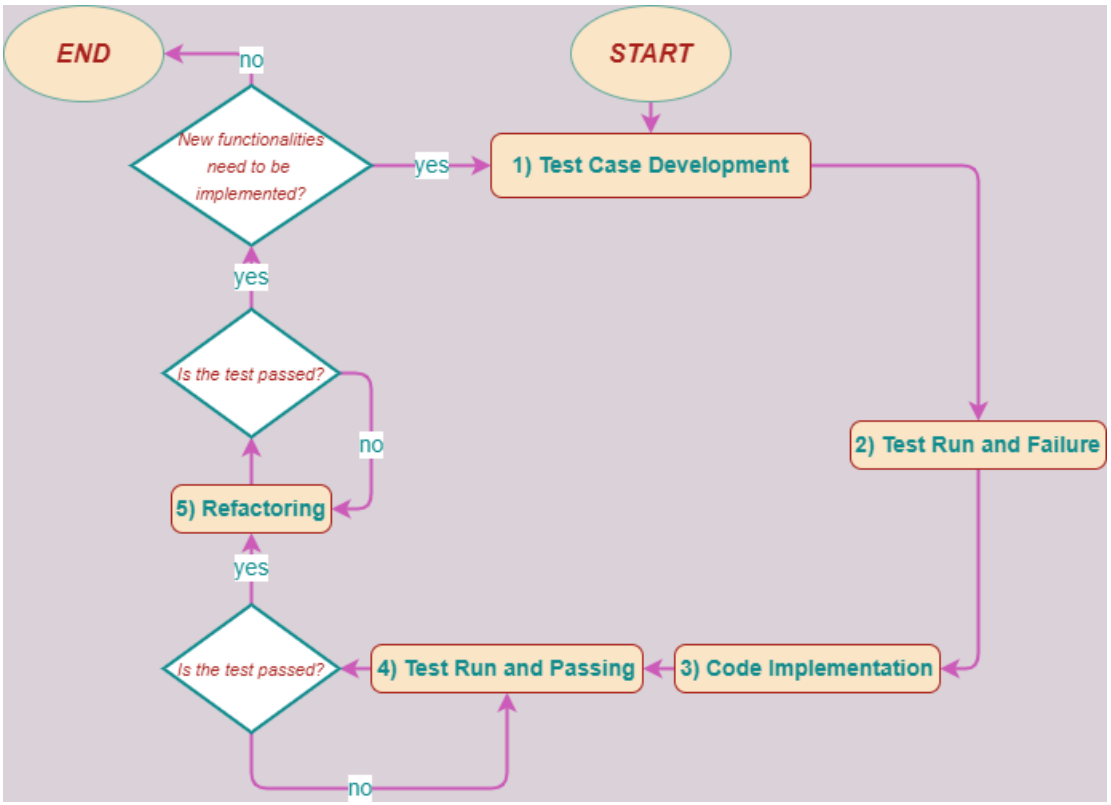

Figure 17. Test Driven Development flow diagram. *A flow diagram representing the five phases of the TDD methodology, which nominally are: test case development, test run and failure, code implementation, test run and passing, and refactoring. The process is iterative and needs to be repeated each time a new functionality is to be implemented.*

All the code developed for the NOCI index population, the PMID management, and the mapping system followed the approach described above. The new testing functionalities were either

integrated within the pre-existing test cases or developed from scratch. The latter option was the case of the tests concerning the brand-new functionalities, such as the id-to-METAID mapping, the creation of PMID-to-DOI mapping files, and the management of data from the National Institute of Health Open Citation Collection.

At the time of writing, all the either created or generated tests are available in the GitHub repository of the fork of the OpenCitations Index Software Library, available at https://github.com/ariannamorettj/index/tree/master/index/test .

4.2.1 Unittest Library

The practical implementation of the test-driven development approach was carried out by exploiting one of the most common modules for testing code developed in Python, i.e. Unittest[100]. More in detail, the Unittest is a widely-used testing framework providing a large selection of facilities for test development and running.

In spite of the number of sophisticated tools offered, only a few basic functionalities are enough for the purposes of most of the developers choosing Unittest. In the specific case of the present thesis project, most of the implemented functionalities were tested by exploiting three of the simplest methods offered, which nominally are assertEqual(), assertFalse() and assertTrue().

The subset of most commonly used methods in the Unittest framework is presented in the table below (see Table 3).

Method	**Checks that**
`assertEqual(a, b)`	`a == b`
`assertNotEqual(a, b)`	`a != b`
`assertTrue(x)`	`bool(x) is True`
`assertFalse(x)`	`bool(x) is False`
`assertIs(a, b)`	`a is b`
`assertIsNot(a, b)`	`a is not b`
`assertIsNone(x)`	`x is None`

[100] «unittest — Unit testing framework — Python 3.10.2 documentation», consultato 5 febbraio 2022, https://docs.python.org/3/library/unittest.html.

`assertIsNotNone(x)`	`x is not None`
`assertIn(a, b)`	`a in b`
`assertNotIn(a, b)`	`a not in b`
`assertIsInstance(a, b)`	`isinstance(a, b)`
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`

Table 3. Unittest methods. *This table presents the subset of the most commonly used methods in the Unittest framework, with the associated type of check performed on the tested data. The source of the table is the official Unittest documentation, and the extended version of the table is available at https://docs.python.org/3/library/unittest.html#assert-methods.*

The convention implies that the test case is created as a subclass of `unittest.TestCase`[101] and that all the methods implemented have names which start with the string "test". Ultimately, the conclusive block of a test contains the command-line program `unittest.main()`[102] , which loads and runs tests.

It is possible to launch the unittest module from the terminal either directly from modules, classes, specific methods or just specifying the file path. The scripts below are three commands launched from the terminal, respectively for running all tests included in the file "`11_nocimapping.py`" (see Script 4), all the test functions of the class NociMappingTest (see Script 5), and the specific test function `test_get_all_files` (see Script 6).

```
python -m unittest index.test.11_nocimapping
```

Script 4. NOCI test command-line call (file). *Command-line call for executing "11_nocimapping.py".*

```
python -m unittest index.test.11_nocimapping.NociMappingTest
```

Script 5. NOCI test command-line call (class). *Command-line call for executing all the testing functions included in the class NociMappingTest, contained in the file "11_nocimapping.py".*

```
python -m unittest
index.test.11_nocimapping.NociMappingTest.test_get_all_files
```

[101] https://docs.python.org/3/library/unittest.html#unittest.TestCase
[102] https://docs.python.org/3/library/unittest.html#unittest.main

Script 6. NOCI test command-line call (method). *Command-line call for executing the testing function test_get_all_files included in the class NociMappingTest, contained in the file "11_nocimapping.py".*

5. Implementation: Technical Aspects

This chapter discusses the technical development aspects of the codebase expansion for the OpenCitations Index Software Library, introduced at a conceptual and structural level in the fourth chapter. The newly implemented scripts and the adaptations made are presented with an exhaustive description of the technical solutions adopted to address the requirements needed for the integration of the National Institute of Health Open Citation Collection data into the OpenCitations infrastructure.

More in detail, each functionality is presented through a description of the scripts expressly developed to implement it and the codebase affected by such adjustments. A special focus is given to the software library expansions concerning the ingestion of the new data, the management of the PMID identifier and the id-to-METAID mapping; in addition, a brief overview is provided also to the scripts which were either slightly modified or extended for secondary technical necessities, such as data storing purposes or specific file format operations.

For what concerns the codebase expansion related to the development of the mapping system, we also provide a brief introduction to the general software infrastructure, which on the one hand determined the constraints to be met and on the other hand provided a both theoretical and technical starting point for the software extension implementation.

In conclusion, we provide an insight on the RAMOSE API configuration file especially defined for NOCI, and in particular regarding the methods used for the metadata retrieval.

5. 1 NIH OCC Data Acquisition: Software Extension for Exploiting a New Data Source

The integration of the National Institute of Health Open Citation Collection in OpenCitations Infrastructure caused the need for new tools and adaptations to manage the ingestion of a new data source. The first step of the codebase expansion is to create a package for the NOCI Index data extraction tools, based on the instantiation of two scripts: (1) for the data retrieval from the original data source (i.e., `nationalinstituteofhealthsource.py`[103]) and (2) for the generation of support material containing data to be used in the main process (i.e., `glob.py`[104]).

[103] ariannamorettj, *Index*, Python, 2021, https://github.com/ariannamorettj/index/blob/0ecf77e5e5c91e4ae481f0abb4834fde9205e141/index/noci/nationalinstituteofhealthsource.py.

[104] ariannamorettj, *Index*, Python, 2021, https://github.com/ariannamorettj/index/blob/0ecf77e5e5c91e4ae481f0abb4834fde9205e141/index/noci/glob1.py.

5.1.1 Data Source Management : NIHCitationSource Class

The `NIHCitationSource` class is developed as an instantiation of the superclass `CSVFileCitationSource`, with the aim of retrieving from the source files provided by the National Institute of Health the initial data required for completing the six-element tuple representing the citation datum in OpenCitations. The structure of the class was partially determined by the superclass constructor `CSVFileCitationSource`, and largely inspired by the implementation adopted for the data retrieval of COCI and CROCI. More in detail, since the COCI citation source provides data in JSON format while CROCI's and NOCI's sources utilize CSV format, the `CrossrefCitationSource` class was implemented as a direct instantiation of the superclass `DirCitationSource`. Therefore, the class used to model the NOCI citation source class was the `CrowdsourcedCitationSource`, which manages source data in CSV format and thus instantiates the class `CSVFileCitationSource`(see Script 7 and Script 8).

```
class NIHCitationSource( CSVFileCitationSource ):
    def __init__(self, src, local_name=""):
        self.pmid = PMIDManager()
        super( NIHCitationSource, self ).__init__( src, local_name )

    def get_next_citation_data(self):
        row = self._get_next_in_file()

        while row is not None:
            citing = self.pmid.normalise(row.get("citing"))
            cited = self.pmid.normalise(row.get("referenced"))

            self.update_status_file()
            return citing, cited, None, None, None, None
            self.update_status_file()
            row = self._get_next_in_file()

        remove(self.status_file)
```

Script 7. NIHCitationSource Class. *NIHCitationSource Class, implemented as an extension of the abstract superclass CSVFileCitationSource.*

```
class CrowdsourcedCitationSource(CSVFileCitationSource):
    def __init__(self, src, local_name=""):
```

```
        self.doi = DOIManager()
        super(CrowdsourcedCitationSource, self).__init__(src,
local_name)

    def get_next_citation_data(self):
        row = self._get_next_in_file()

        while row is not None:
            citing = self.doi.normalise(row.get("citing_id"))
            cited = self.doi.normalise(row.get("cited_id"))

            if citing is not None and cited is not None:
                created = row.get("citing_publication_date")
                if not created:
                    created = None
cited_pub_date = row.get("cited_publication_date")
                if not cited_pub_date:
                    timespan = None
                else:
                    c = Citation(None, None, created, None,
cited_pub_date, None, None, None, None, "", None, None, None, None,
None)
                    timespan = c.duration

                self.update_status_file()
                return citing, cited, created, timespan, None, None

            self.update_status_file()
            row = self._get_next_in_file()

        remove(self.status_file)
```

Script 8. CrowdsourcedCitationSource Class. *CrowdsourcedCitationSource Class, implemented as an extension of the abstract superclass CSVFileCitationSource.*

The class constructor of both `CrowdsourcedCitationSource` and `NIHCitationSource` returns the six elements defining a citation datum in OpenCitations. However, none of the citation sources provides data to define whether or not the datum in question is a self citation (either with respect to the author or to the journal); therefore, the two fields stating the self citation are set to `None`. For what concerns the date of creation of the citation and the timespan between the dates of publication of the citing and cited entity, in the `CrowdsourcedCitationSource` the publication dates are provided, and thus the timespan can be retrieved using the functionalities implemented by the class `Citation`. On the contrary, the National Institute of Health source file for citation data only consists of a minimal CSV file, storing the PMID of the citing entity in the

field "`citing`" and the PMID of the cited entity in the field "`referenced`". The only two elements retrieved from the source are the mandatory ones for the definition of a citation, i.e. the two identifiers of the entities involved in the citation (i.e., the citing and cited entity).

"citing"	**"referenced"**
2140506	2942070
1523579	7097569
1509982	6501574
1968312	13673087
2330868	3958380
1854174	3037997
2038824	2494239
2373284	7189714
3591292	4092853
2368927	355650

Table 4. OCC CSV file sample. *Sample CSV extract of the Open Citation Collection by the National Institute of Health.*

5.1.2 Support Files Creation: NOCI Glob

The other source-specific functionality developed is based on the extraction of useful metadata to be stored in support files to be used in the indexes' population process, in order to avoid external API requests. This feature was already implemented for COCI, therefore the NOCI's glob[105] script for the extraction of support data was structured so as to accomplish the same purposes of COCI's glob, although adapted to the specified metadata source file.

More in detail, both the glob scripts of COCI and NOCI take as input the data from the respective sources and store the outputs in four CSV files, concerning:

1) *Validation*. These data assess whether or not the identifiers were successfully validated through the id-type-specific API service. A positive outcome of the validation process is signified by the character "v", while the negative outcome is identified by the letter "i".

[105] ariannamorettj, *Index*, Python, 2021, https://github.com/ariannamorettj/index/blob/0ecf77e5e5c91e4ae481f0abb4834fde9205e141/index/noci/glob1.py.

2) *ISSN*. The file contains associations between entities' ids and the ISSN identifiers of the publishing journals.
3) *ORCID*. The file contains associations between entities' ids and the ORCID identifiers of the authors of the entities.
4) *Date of creation*. The file stores associations between identifiers of bibliographic entities and the corresponding dates of publication, represented as a string value in the format YYYY-MM-DD (e.g.: "2019-05-27").

All the four generated CSV files share the same simple structure, consisting of a two-column CSV, storing in the first field the processed identifier (i.e., "id") and in the second the related metadata (i.e., "value"). An explicative sample of each of the four files is provided in the tables below (see Table 5, Table 6, Table 7, Table 8).

"id"	**"value"**
"pmid:3"	"v"
"pmid:9790254"	"v"
"pmid:9790254"	"v"

Table 5. Validation CSV file sample. *Sample of the CSV file generated with the OpenCitations CSVManager Class, associating a PMID identifier with a character representing its state of validity – "v" for valid and "i"for invalid.*

"id"	**"value"**
"pmid:3"	"1975"
"pmid:9790254"	"1998"
"pmid:23183539"	"2013"

Table 6. ID to year CSV file sample. *Sample of the CSV file generated with the OpenCitations CSVManager Class, associating a PMID identifier with the value of the year of publication (in string format).*

"id"	**"value"**

"pmid:3"	"0006-291X"
"pmid:9790254"	"0918-8959"
"pmid:9790254"	"1348-4540"
"pmid:23183539"	"1871-6784"

Table 7. ID to ISSN CSV file sample. *Sample of the CSV file generated with the OpenCitations CSVManager Class, associating a PMID identifier of a bibliographic entity with the eight-character ISSN code identifying its journal of publication.*

"id"	**"value"**
"pmid:3"	"0000-0002-0524-4077"
"pmid:23183539"	"0000-0003-4577-8759"
“pmid:13673087”	“0000-0001-6263-420X”
“pmid:13673087”	“0000-0001-5204-5460”

Table 8. ID to ORCID CSV file sample. *Sample of the CSV file generated with the OpenCitations CSVManager Class, associating a PMID identifier of a bibliographic entity with the ORCID code of its author(s).*

For NOCI glob[106], the given input is a CSV file storing the iCite Metadata dataset[107]. This CSV file stores metadata for bibliographic entities in twenty-four columns, many of which refer to biomedical technical aspects (e.g., `"molecular_cellular"`: Fraction of Medical Subject Headings[108] terms that are in the Molecular/Cellular Biology category, out of the article's Medical Subject Headings terms categorized as Human, Animal, or Molecular/Cellular[109]). The only information needed with respect to the OpenCitation purposes is provided in the fields listed below:

1) `"pmid"`: contains the primary identifier of the described entity.
2) `"doi"`. In case mapping information is available, the present field contains the DOI identifier of the same entity associated with the PMID.

[106] ariannamorettj, *Index*, Python, 2021, https://github.com/ariannamorettj/index/blob/0ecf77e5e5c91e4ae481f0abb4834fde9205e141/index/noci/glob1.py.
[107] George Santangelo, ‘ICite Database Snapshots (NIH Open Citation Collection)’, 11 February 2022, https://doi.org/10.35092/yhjc.c.4586573.v27.
[108] ‘Home - MeSH - NCBI’, accessed 20 February 2022, https://www.ncbi.nlm.nih.gov/mesh/.
[109] George Santangelo, ‘ICite Database Snapshots (NIH Open Citation Collection)’, 11 February 2022, https://doi.org/10.35092/yhjc.c.4586573.v27.

3) `"authors"`: contains a string with the names of the publication authors separated by commas. The names are expressed with the initial capital letters of the first names separated by whitespaces and the extended surname.
4) `"journal"`: the abbreviated name of the journal of publication of the bibliographic entity.
5) `"year"`: the date of publication is expressed as the string of the year of publication of the bibliographic entity and stored in the field "year".
6) `"references"`: a string including the PMIDs of all the publications cited by the bibliographic entity, separated by whitespaces.

However, most of the data in iCite Metadata do not respect the format used in OpenCitations, because of a different presentation of the same type of information. Indeed, both the data about the authors and the journal of publishing of the bibliographic entity are expressed as strings of their names, while the most trustable approach to identify both a person and a journal is using the official identifiers, i.e., ORCID and ISSN, respectively. Therefore, to retrieve trustable data, we exploited the PMID-to-DOI mapping information provided in iCiteMetadata file: through the OpenCitations DOI management functionalities[110] we obtain ORCID and ISSN data from external DOI API services, and we associate it with the PMID. Although homonymy is common among humans, this is not the case for journal names: each journal is supposed to have a unique name, but can have multiple ISSNs in case the same bibliographic entity is published either on different media or in different languages[111]. Therefore, an additional mechanism was developed to associate journals' abbreviated names with the corresponding ISSN code(s). This process is based on a support file consisting of a JSON dictionary where the keys are the abbreviations of the journals' names and the values are lists storing the ISSN identifier(s) of each journal (see Script 9). This process allows the software to avoid multiple API calls for retrieving the same set of ISSNs each time a journal name is encountered. However, as anticipated, the same procedure could not be adopted for the authors' ORCID identifiers, because of the high risk of misidentification of homonyms, i.e., author disambiguation issue (Hussain and Asghar 2017). Coherently, the retrieval approach for authors' ORCIDs adopted in NOCI glob does not rely on the data provided by the National Institute of Health, but only exploits DOI API services, when the PMID-to-DOI mapping information is available.

[110] *Index*, Python (2019; repr., OpenCitations, 2022), https://github.com/opencitations/index/blob/738a6a1244731a812b1bec15646c6bbba26dedcb/index/identifier/doimanager.py.
[111] 'ISSN, the Major Principles | ISSN', accessed 20 February 2022, https://www.issn.org/understanding-the-issn/assignment-rules/issn-the-major-principles/.

```
{
    "Biochem Biophys Res Commun": [
        "0006-291X"
    ],
    "Endocr J": [
        "0918-8959",
        "1348-4540"
    ],
    "N Biotechnol": [
        "1871-6784"
    ],
    "Biochem Med": [
        "0006-2944"
    ],
    "Magn Reson Chem": [
        "0749-1581"
    ]
}
```

Script 9. Journal name to ISSNs list sample JSON dictionary. *Sample of a support JSON dictionary, associating the abbreviations of the journals' names (used as dictionary keys) to the lists storing the potentially multiple ISSN identifier(s) of each journal (used as values).*

The information about the date of publication is not fully informative, since it is expressed only with the four-character string representing the year of publication. However, although minimal, this data is sufficient for defining and storing information about the date of publication in the support files.

Finally, the string stored in the field `"references"` is splitted using whitespaces as delimiter; these splitted items are first validated using the PMID-specific tools and then stored in the support file.

In the NOCI glob, the core functionalities are implemented by the function `process`, which relies on four inner functions serving specific purposes:

1) `get_all_files`: this function extracts all the files from a given repository. It takes as input a repository and returns a list of all its contained elements accompanied by the suitable method to open such files. The function handles the possibility of encountering compressed file formats.
2) `build_pubdate`: this operation takes as input a record of the iCite Metadata CSV file (i.e., a row), gets the value of the "year" field, normalizes the four-characters numeric string and returns it.

3) `issn_data_recover`: this function recovers the previously stored data mapping the journal names with their related ISSNs. In order to do that, the function takes a repository as input, checks the presence of a file named "`journal_issn.json`", and returns a dictionary containing the information stored in the JSON file. It returns an empty dictionary in the case the file does not exist.
4) `issn_data_to_cache`: this function takes as input a dictionary associating the names of journals (in string format) to a list of their ISSN identifiers and a path to a specific directory. The function stores the dictionary in the file "`journal_issn.json`", in the specified directory.

The workflow for the generation of the support files is managed by the function `process`, which takes as arguments both the directory storing the input material, the output folder to store the support files, and an integer representing the number of lines after which the data stored in the dictionary for the mapping between journal names and the related ISSNs are to be stored into a JSON cache file.

The process starts by checking the existence of the specified output folder. The folder is automatically created in case it does not exist . Then, almost all the variables are initialized using other OpenCitations Index Software Library classes (see Script 10).

```
def process(input_dir, output_dir, n):
    if not exists(output_dir):
        makedirs(output_dir)

    citing_pmid_with_no_date = set()
    valid_pmid = CSVManager( output_dir + sep + "valid_pmid.csv" )
    valid_doi = CSVManager("index/test_data/crossref_glob" + sep +
"valid_doi.csv")
    id_date = CSVManager( output_dir + sep + "id_date_pmid.csv" )
    id_issn = CSVManager( output_dir + sep + "id_issn_pmid.csv" )
    id_orcid = CSVManager( output_dir + sep + "id_orcid_pmid.csv" )
    journal_issn_dict = issn_data_recover(output_dir) #just an empty
dict in case the code never broke
    pmid_manager = PMIDManager(valid_pmid)
    crossref_resource_finder = CrossrefResourceFinder(valid_doi)
    orcid_resource_finder = ORCIDResourceFinder(valid_doi)
    doi_manager = DOIManager(valid_doi)
    issn_manager = ISSNManager()
    orcid_manager = ORCIDManager()
```

Script 10. NOCI glob - process function. *Snippet from the process function of the NOCI glob file, aimed at creating support files to be used in the main workflow for the NOCI index population.*

At this point, all the given files are iteratively accessed. The CSV files are read with the Pandas Python Library[112] in concatenated chunks of a thousand rows each, in order to manage the elaboration procedure of large CSV files (see Script 11).

```
all_files, opener = get_all_files(input_dir)
    len_all_files = len(all_files)

    print( "\n\n# Add valid PMIDs from NIH metadata" )
    for file_idx, file in enumerate( all_files, 1 ):
        df = pd.DataFrame()

        for chunk in pd.read_csv(file, chunksize=1000 ):
            f = pd.concat( [df, chunk], ignore_index=True )
            f.fillna("", inplace=True)
```

Script 11. NOCI glob - process function, Pandas DataFrame. *Snippet from the process function of the NOCI glob file: Pandas DataFrame creation for reading the input CSV files.*

At this point, the iteration over the rows of the CSV file begins. The index of the rows is used as a counter in order to check when the number of lines specified as input has been processed, so as to transcribe the associations between journal names and correspondent ISSNs to the cache files (see Script 12).

```
print( "Open file %s of %s" % (file_idx, len_all_files) )
for index, row in f.iterrows():
    if int( index ) != 0 and int( index ) % int( n ) == 0:
        print( "Group nr.", int( index ) // int( n ), "processed. Data
from", int( index ),
               "rows saved to journal_issn.json mapping file" )
        issn_data_to_cache( journal_issn_dict, output_dir )
```

Script 12. NOCI glob - process function, Transcription to cache file. *Snippet from the process function of the NOCI glob file: transcription of the data associating journals' names to the corresponding ISSN to the external JSON cache file.*

The following step stores in the variables `citing_pmid` and `citing_doi` the normalized versions of the PMID and DOI identifiers retrieved from the fields "pmid" and "doi" of the given CSV, respectively. The value of the `citing_pmid` is also validated and stored as either valid or invalid in the support file concerning the ids validation (see Script 13).

[112] 'Pandas Documentation — Pandas 1.4.1 Documentation', accessed 14 February 2022, https://pandas.pydata.org/docs/index.html.

```
citing_pmid = pmid_manager.normalise( row['pmid'], True )
pmid_manager.set_valid( citing_pmid )
citing_doi = doi_manager.normalise( row['doi'], True )
```

Script 13. NOCI glob - process function, Normalization and validation. *Snippet from the process function of the NOCI glob file: PMID normalization and validation, DOI normalization.*

After checking that no date is already stored in the support files for the PMID in question, the process extracts the date of publication for a given bibliographic entity, by calling the `build_pubdate` function. The association PMID-date is stored in the correspondent support file (see Script 14). In case the date of publication for a given entity is not provided, the PMID of such an entity is stored in the set `citing_pmid_with_no_date`.

```
if id_date.get_value( citing_pmid ) is None:
    citing_date = Citation.check_date( build_pubdate( row ) )
    if citing_date is not None:
        id_date.add_value( citing_pmid, citing_date )
        if citing_pmid in citing_pmid_with_no_date:
            citing_pmid_with_no_date.remove( citing_pmid )
    else:
        citing_pmid_with_no_date.add( citing_pmid )
```

Script 14. NOCI glob - process function, Date of publication management. *Snippet from the process function of the NOCI glob file: date of publication extraction and management.*

Once we are sure that there is no ISSN associated for a given identifier, the abbreviated name of the journal is extracted from the CSV row and researched among the keys of the `journal_issn_dict` dictionary. In case the journal is internally identified, the ISSN support file is updated with a PMID-ISSN pairing for each ISSN of the journal. Alternatively, we use the Crossref Resource Finder[113] – to retrieve the ISSN(s) of the journal of publication and update both the `journal_issn_dict` dictionary and the PMID-ISSN support file (see Script 15).

```
if id_issn.get_value( citing_pmid ) is None:
    journal_name = row["journal"]
    if journal_name:
        if journal_name in journal_issn_dict.keys():
            for issn in journal_issn_dict[journal_name]:
```

[113] *Index*, Python (2019; repr., OpenCitations, 2022), https://github.com/opencitations/index/blob/738a6a1244731a812b1bec15646c6bbba26dedcb/index/finder/crossrefresourcefinder.py.

```
                id_issn.add_value( citing_pmid, issn )
        else:
            if citing_doi is not None:
                json_res = crossref_resource_finder._call_api(
citing_doi )
                if json_res is not None:
                    issn_set = crossref_resource_finder._get_issn(
json_res )
                    if len( issn_set ) > 0:
                        journal_issn_dict[journal_name] = []
                    for issn in issn_set:
                        issn_norm = issn_manager.normalise( str( issn )
)
                        id_issn.add_value( citing_pmid, issn_norm )
                        journal_issn_dict[journal_name].append(
issn_norm )
```

Script 15. NOCI glob - process function, ISSN management. *Snippet from the process function of the NOCI glob file: ISSN data management.*

After having assessed that no PMID-ORCID association is already registered in the support files, the information is retrieved using the DOI associated with the PMID, which is further used while calling the ORCID Resource Finder API service. The PMID-ORCID pairs are stored in the support files. Before ending the loop, all the information stored in the `journal_issn_dict` dictionary is transcribed to the cache file (see Script 16).

```
   if id_orcid.get_value( citing_pmid ) is None:
        if citing_doi is not None:
            json_res = orcid_resource_finder._call_api( citing_doi )
            if json_res is not None:
                orcid_set = orcid_resource_finder._get_orcid( json_res
)
                for orcid in orcid_set:
                    orcid_norm = orcid_manager.normalise( orcid )
                    id_orcid.add_value( citing_pmid, orcid_norm )

issn_data_to_cache( journal_issn_dict, output_dir )
```

Script 16. NOCI glob - process function, ORCID management. *Snippet from the process function of the NOCI glob file: ORCID management.*

At this point, the iteration over all the input files is repeated: each file is read as a Pandas DataFrame and accessed as a concatenated thousand-line chunk. The aim of this second iteration is

to extract and validate all the identifiers from the reference lists provided in the field "reference" (see Script 17).

```
print( "\n\n# Checking the referenced pmids validity" )
    for file_idx, file in enumerate( all_files, 1 ):
        df = pd.DataFrame()

        for chunk in pd.read_csv( file, chunksize=1000 ):
            f = pd.concat( [df, chunk], ignore_index=True )
            f.fillna("", inplace=True)
            print( "Open file %s of %s" % (file_idx, len_all_files))
            for index, row in f.iterrows():
                if row["references"] != "":
                    ref_string = row["references"].strip()
                    ref_string_norm = re.sub("\s+", " ", ref_string)
                else:
                    print("the type of row reference is",
(row["references"]), type(row["references"]))
                    print(index, row )

                cited_pmids = set(ref_string_norm.split(" "))
                for cited_pmid in cited_pmids:
                    if pmid_manager.is_valid(cited_pmid):
                        print("valid cited pmid added:", cited_pmid)
                    else:
                        print("invalid cited pmid discarded:",
cited_pmid)
```

Script 17. NOCI glob - process function, References management. *Snippet from the process function of the NOCI glob file: extraction and validation of the PMID listed in the field "references" of the input iCite Metadata CSV file.*

In conclusion, all the PMIDs saved in the set citing_pmid_with_no_date are associated with empty strings and stored in the ID-date support file (see Script 18).

```
   for pmid in citing_pmid_with_no_date:
        id_date.add_value( pmid, "" )
```

Script 18. NOCI glob - process function, PMID with no date. *Snippet from the process function of the NOCI glob file: management of the PMID of those entities whose date of publication was not possible to retrieve.*

5.2 PMID Data Integration: Code Integrations and Adaptations

This section is dedicated to the presentation of the OpenCitations Index Software Library codebase extensions and adaptations to handle the management of the PMID identifier in a DOI-based system. We introduce the codebase expansions implemented through the creation of the `PMIDManager` and `NIHResourceFinder` classes, as well as all the other modifications made to the pre-existing software.

5.2.1 PMIDManager: a New Identifier Manager For PMIDs

Before ingesting PMID-based entities, the three typologies of identifiers managed by the OpenCitations Index Software Library have been: DOI, ISSN and ORCID. Each identifier is managed by a specific class instance of the superclass `IdentifierManager`, thus all the classes share the same structure and the same two core methods for validating and normalizing the processed identifiers.

Coherently, the `PMIDManager` class was developed as an instantiation of the `IdentifierManager` class as well; and it is largely influenced by the same development strategy adopted by the `DOIManager` class. Indeed, the DOI and the PMID managers also share the function which handles the identifiers of the bibliographic entities. Therefore, in addition to the two core methods for validating and normalizing the processed identifiers, both the classes also provide a method for storing in the support files the information concerning the validity state of the identifiers (i.e., `set_valid`), and a method for assessing the existence of the identifier through API services (i.e., `__doi_exists` and `__pmid_exists`).

The `set_valid` function takes as input a PMID string, normalizes it (using the id-type-specific normalizer function), and checks whether or not the information about its validation is already stored in the validity support file for PMIDs. In case the datum is not stored yet, it fills out a row in the CSV support file associating the processed identifier with the one-character string “v”, which assesses the validity of the PMID (See Script 19).

```
def set_valid(self, id_string):
    pmid = self.normalise(id_string, include_prefix=True )
    if self.valid_pmid.get_value( pmid ) is None:
        self.valid_pmid.add_value( pmid, "v" )
```

Script 19. PMIDManager - Validation setting method. *Script of the method set_valid of the class PMIDManager.*

The validator function works on assessing the validity of the identifier given as input. Just like the normalizer function, also `is_valid` is strictly dependent on the identifier structure. Indeed, after having normalized the PMID using the id-type-specific `normalize` function, it checks that the identifier exists and that it is a sequence of digits which does not start from zero(s). At this point, the function checks if the identifier is already present in the validation support file: in the negative case, the existence of the PMID is assessed with the id-specific `_pmid_exists` function. With respect to the two alternative possible outcomes, the data about the validity of the identifier is stored in the support file, either with the string "`v`" in case of validity or with the string "`i`" for the opposite case.

```
    def is_valid(self, id_string):
        pmid = self.normalise( id_string, include_prefix=True )
        if pmid is None or match( "^pmid:[1-9]\d*$", pmid ) is None:
            return False
        else:
            if self.valid_pmid.get_value( pmid ) is None:
                if self.__pmid_exists( pmid ):
                    self.valid_pmid.add_value( pmid, "v" )
                else:
                    self.valid_pmid.add_value( pmid, "i" )
            return "v" in self.valid_pmid.get_value( pmid )
```

Script 20. PMIDManager - Validation method. *Script of the method is_valid of the class PMIDManager, for PMID validation.*

The normaliser is a function which takes as input a PMID identifier in string format without the "pmid:" prefix and corrects potential inconsistencies with respect to the identifier standard format. In the PMID case, the identifier consists of a series of digits representing the sequential number given to a specific publication. For this reason, using the regular expressions, all the non-digit characters and all the initial zeros are removed (see Script 21).

```
    def normalise(self, id_string, include_prefix=False):
        id_string = str(id_string)
        try:
            pmid_string = sub( "^0+", "", sub( "\0+", "", (sub(
"[^\d+]", "", id_string )) ) )
            return "%s%s" % (self.p if include_prefix else "",
pmid_string)
        except:
            return None
```

Script 21. PMIDManager - Normalization method. *Script of the method normalise of the class PMIDManager, for the normalization of PMIDs structure.*

Beyond the formal correctness of the identifier structure, the only way to assess the factual existence of an identifier is through an API service. The one used for the PMID case is a service provided by the National Institute of Health PubMed at https://pubmed.ncbi.nlm.nih.gov/[114], using the suffix option "`/?format=pmid`". The response to the request is an HTML page. We use the Beautiful Soup Python library[115] for HTML and XML data extraction to read the output of the call. Information about the searched identifier is stored in the head of the HTML response in the `meta` element, with two attributes, namely `name` and `content`. The value "`uid`" of the attribute `name` identifies the tag containing the ID information, stored , as the value of the attribute `content`. The existence of the PMID can be asserted in case the value of the attribute `content` corresponds to the searched identifier.

```
    def __pmid_exists(self, pmid_full):
        pmid = self.normalise( pmid_full )
        if self.use_api_service:
            tentative = 3
            while tentative:
                tentative -= 1
                try:
                    print( self.api + quote( pmid ) + "/?format=pmid" )
                    r = get( self.api + quote( pmid ) +
"/?format=pmid", headers=self.headers, timeout=30 )
                    if r.status_code == 200:
                        r.encoding = "utf-8"

                        soup = BeautifulSoup( r.content )
                        for i in soup.find_all( "meta", {"name": "uid"}
):
                            id = i["content"]
                            if id == pmid:
                                return True

                except ReadTimeout:
                    pass
                except ConnectionError:
                    sleep(5)

        return False
```

[114] «PubMed», consultato 31 gennaio 2022, https://pubmed.ncbi.nlm.nih.gov/.
[115] «Beautiful Soup Documentation — Beautiful Soup 4.9.0 documentation», consultato 12 febbraio 2022, https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

Script 22. PMIDManager - Assertion of existence method. *Script of the method __pmid_exists of the class PMIDManager, for assessing the existence of PMIDs identifiers via API service calls.*

```
<meta name="uid" content="12">.
```

Script 23. NIH API response "pmid" - HTML <meta> element "uid". *Example of a meta tag with the attribute name="uid" for assessing the existence of the identifier. Sample for the PMID "12", extracted from the head of the response HTML document received for the API request https://pubmed.ncbi.nlm.nih.gov/12/?format=pmid .*

5.2.2 NIHResourceFinder: Managing API Services For PMID

In the OpenCitations Index Software Library, a resource finder is a tool used to gather data from external services, such as Crossref, DataCite and ORCiD, for retrieving the identifiers metadata via HTTP REST APIs. Just like `DataCiteResourceFinder`, `ORCIDResourceFinder`, and `CrossrefResourceFinder`, the `NIHResourceFinder` was developed as an instantiation of the abstract class `ApiIDResourceFinder`, and thus provides methods for calling the API service and retrieve data about the date of publication and the ISSN of the publishing journal. Despite the fact that the abstract class `ApiIDResourceFinder` provides a constructor for retrieving ORCIDs, the National Institute of Health Pubmed API service used for the NIH Resource Finder does not provide any information about ORCID. Therefore, the method was not implemented.

The National Institute of Health Pubmed API service is available at https://pubmed.ncbi.nlm.nih.gov/ . To implement the functionalities of the resource finder, the display option used was "PubMed". The response to the request is provided in HTML (see Script 24).

```
<!DOCTYPE html>
<html lang="en">
<head  itemscope=""  itemtype="http://schema.org/WebPage"  prefix="og:
http://ogp.me/ns#">
…
</head>

<body>
<div class="article-page" id="article-page">
<pre class="article-details" id="article-details">
PMID- 970516
OWN - NLM
STAT- MEDLINE
DCOM- 19761201
LR  - 20190514
```

IS - 0090-0036 (Print)
IS - 1541-0048 (Electronic)
IS - 0090-0036 (Linking)
VI - 66
IP - 10
DP - 1976 Oct
TI - Quality of care given to first time birth control patients at a free clinic.
PG - 986-7
AB - In an empirical study of the quality of care at a free clinic, criteria for optimal
care for female first visits for birth control were established and 100 charts were
reviewed, 50 in April 1974 and 50 in April 1975 with an interval in between of
in-service training accompanied by new medical forms and procedures. An encouraging
improvement in record keeping was observed. The authors feel it is important that
free clinics concentrate on quality as well as quantity and accessibility of care.
FAU - Grover, M
AU - Grover M
FAU - Greenberg, T
AU - Greenberg T
LA - eng
PT - Journal Article
TA - Am J Public Health
JT - American journal of public health
JID - 1254074
RN - 0 (Contraceptives, Oral, Hormonal)
SB - IM
MH - Adult
MH - California
MH - Clinical Laboratory Techniques/standards
MH - Contraceptives, Oral, Hormonal
MH - *Family Planning Services
MH - Female
MH - Humans
MH - Male
MH - Medical History Taking
MH - Physical Examination/standards
MH - *Quality of Health Care
PMC - PMC1653443
EDAT- 1976/10/01 00:00
MHDA- 1976/10/01 00:01
CRDT- 1976/10/01 00:00
PHST- 1976/10/01 00:00 [pubmed]
PHST- 1976/10/01 00:01 [medline]
PHST- 1976/10/01 00:00 [entrez]
AID - 10.2105/ajph.66.10.986 [doi]
PST - ppublish
SO - Am J Public Health. 1976 Oct;66(10):986-7. doi: 10.2105/ajph.66.10.986.
</pre>

```
</body>
</html>
```

Script 24. NIH API response "pubmed" - HTML <pre> element. *HTML response to the API request to The National Institute of Health Pubmed service, at https://pubmed.ncbi.nlm.nih.gov/, selecting the display option "PubMed".*

The required data are stored in a `<pre>` element identified by the attributes `class="article-details"` and `id="article-details"`. The textual object contained in the element is then parsed with regular expression tools provided with the Python module re — Regular expression operations[116].

The functionality of API response retrieval and the extraction of the textual object containing the required data is implemented by the `NIHResourceFinder` method `_call_api` (see Script n 25). This method takes as input a PMID identifier and uses it when calling the corresponding API request to the NIH PubMed service, using the display option "PubMed". In case of a positive response, the method uses the Beautiful Soup Python library[117] to extract the textual object contained in the `<pre>` element identified by the `id` attribute `"article-details"`. Finally, the method returns a textual object containing the metadata of the entity identified by the PMID used as a parameter in the API request. This textual object (i.e., `mdata`) is then given as input to the other methods provided by the class `NIHResourceFinder`, described below (see Script 26, Script 27).

```
    def _call_api(self, pmid_full):
        if self.use_api_service:
            pmid = self.pm.normalise(pmid_full)
                r = get(self.api + quote(pmid) + "/?format=pubmed",
headers=self.headers, timeout=30)
            if r.status_code == 200:
                r.encoding = "utf-8"
                soup = BeautifulSoup(r.text, features="lxml")
                mdata = str(soup.find(id="article-details"))
                return mdata
```

Script 25. NIHResourceFinder - API call method. *NIHResourceFinder method _call_api, implementing the functionalities of API response retrieval and extraction of the textual object containing the required data.*

The method `_get_issn` (see Script 26) takes as input the textual object retrieved by the `_call_api` method and extracts a set of ISSN codes by using regular expressions. The first

[116] 'Re — Regular Expression Operations — Python 3.10.2 Documentation', accessed 13 February 2022, https://docs.python.org/3/library/re.html.
[117] 'Beautiful Soup Documentation — Beautiful Soup 4.9.0 Documentation', accessed 12 February 2022, https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

regex is used to retrieve all the strings starting with the uppercase characters “IS” followed by one or more spacing characters, a dash, one or more spacing characters, and two sequences of four digits separated by a dash, forming the eight-digit ISSN code. The structure of the ISSN code can be expressed as NNNN-NNNC, where N is a digit and the C is a check digit, which can be either another numerical digit or an uppercase “X” character (Rozenfeld 2001). The substrings of the ISSN codes retrieved (i.e. NNNN-NNNC) are extracted and added to a set, which is ultimately returned.

```
def _get_issn(self, txt_obj):
    result = set()
    issns = re.findall("IS\s+-\s+\d{4}-\d{4}", txt_obj)
    for i in issns:
        issn = re.search("\d{4}-\d{4}", i).group(0)
        result.add(issn)
    return result
```

Script 26. NIHResourceFinder - Get ISSN method. *NIHResourceFinder method _get_issn, implementing the ISSN data extraction from the textual object derived from the API call response.*

Just like the `_get_issn` method, also `_get_date` (see Script 27) takes as input the textual object retrieved by `_call_api`. The date of publication of the entity identified by the PMID could be defined at three incremental levels of detail, either in full format (YYYY Mon DD), or with the year and the month of publication (YYYY Mon), or with the year only (YYYY). For this reason, the script of the method is organized in if-else blocks aimed at storing the date at the maximum level of informativity available: first of all, the date is searched in full format, in case it is not found, it is searched in the format including the year and the month, and, in case of a further failure, it is searched as the four digit string representing the year only. In case none of the previous attempts produced a result, the method returns `None`.

Each of the three research attempts is established using the `re.search` method.. The full format regular expression is implemented to match the uppercase string “DP”, followed by one or more spaces, a dash character, one or more spaces, the four digits representing the year, zero or one spacing character, a three-character string identifying a month of the year, zero or more spacing characters and two digits representing the day of the month.

In case at least one of the three attempts produced a result, the method uses the Datetime library[118] to turn the raw string representing the date into a proper datetime object with the method

[118] ‘Datetime — Basic Date and Time Types — Python 3.10.2 Documentation’, accessed 13 February 2022, https://docs.python.org/3/library/datetime.html.

`strptime`. At this point, the datetime object is turned again into a string using the method `strftime`, and the string is eventually returned[119].

```
   def _get_date(self, txt_obj):
        date =
re.search("DP\s+-\s+(\d{4}(\s?(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|
Nov|Dec))?(\s?((3[0-1])|([1-2][0-9])|([0]?[1-9])))?)",
txt_obj).group(1)
        re_search =
re.search("(\d{4})\s+(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)\
s+((3[0-1])|([1-2][0-9])|([0]?[1-9]))", date)
        if re_search is not None:
            result = re_search.group(0)
            datetime_object = datetime.strptime(result, '%Y %b %d')
            return datetime.strftime(datetime_object, '%Y-%m-%d')

        else:
            re_search =
re.search("(\d{4})\s+(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)"
, date)
            if re_search is not None:
                result = re_search.group(0)
                datetime_object = datetime.strptime(result, '%Y %b')
                return datetime.strftime(datetime_object, '%Y-%m')

            else:
                re_search = re.search("(\d{4})", date)
                if re_search is not None:
                    result = re.search("(\d{4})", date).group(0)
                    datetime_object = datetime.strptime(result, '%Y')
                    return datetime.strftime(datetime_object, '%Y')

                else:
                    return None
```

Script 27. NIHResourceFinder - Get date method. *NIHResourceFinder method _get_date, aimed at retrieving the string of the date of publication of the entity identified by the searched PMID.*

[119] 'Datetime — Basic Date and Time Types — Python 3.10.2 Documentation', accessed 13 February 2022, https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior.

5.3 Side Pre-existing Codebase Adjustments for PMID Integration and NIH Data Source Management

After having analyzed the scripts explicitly implemented for the PMID management, the present section presents an overview of the side adjustment applied to the pre-existing codebase, to adapt a DOI-based system to manage identifiers of different types.

The necessity to split the process following the id-type motivated the introduction of two attributes in the class `OCIManager`[120], in order to provide the opportunity of specifying the type of id to be processed. The two additional attributes are named `doi_type` and `pmid_type`, and the values associated are respectively the "`doi`" and "`pmid`", that are the strings of the id type names.

To get the data from the new source for populating NOCI index, the `DataHandler` class got the addition of the source class `"noci" : NIHCitationSource` among the other source classes, which nominally are `"csv": CSVFileCitationSource`, `"crossref": CrossrefCitationSource`, and `"croci": CrowdsourcedCitationSource`[121].

Other adaptations have been made to the `FileDataHandler` class for the creation of the citations. More in detail, the `id_type` parameter was added as an ulterior argument of the class. The parameter is used to differentiate the process with respect to the type of identifier used by the specific data source to express citation data, and consequently call the correct identifier manager, i.e., using the `PMIDManager` for the data from the National Institute of Health, and the `DOIManager` for all the other sources (see Script 28).

```
   def init(self, data, doi_file, date_file, orcid_file, issn_file,
orcid, no_api, id_type):
        valid_doi, id_date, id_orcid, id_issn = \
            FileDataHandler._create_csv(doi_file, date_file,
orcid_file, issn_file)

        if id_type == OCIManager.doi_type:
            self.id_manager = DOIManager(valid_doi, use_api_service=not
no_api)
            crossref_rf = CrossrefResourceFinder(
                date=id_date, orcid=id_orcid, issn=id_issn,
doi=valid_doi, use_api_service=not no_api )
            datacite_rf = DataCiteResourceFinder(
```

[120] ariannamorettj, *Index*, Python, 2021, https://github.com/ariannamorettj/index/blob/29dd843b4c138073bb8ffd8c7aea113b8ceff9b4/index/citation/oci.py.
[121] ariannamorettj, *Index*, Python, 2021, https://github.com/ariannamorettj/index/blob/29dd843b4c138073bb8ffd8c7aea113b8ceff9b4/index/storer/datahandler.py. .

```
                date=id_date, orcid=id_orcid, issn=id_issn,
doi=valid_doi, use_api_service=not no_api )
            orcid_rf = ORCIDResourceFinder(
                date=id_date, orcid=id_orcid, issn=id_issn,
doi=valid_doi,
                use_api_service=True if orcid is not None and not
no_api else False, key=orcid )
            self.rf_handler = ResourceFinderHandler( [crossref_rf,
datacite_rf, orcid_rf] )


        elif id_type == OCIManager.pmid_type:
            self.id_manager = PMIDManager(valid_doi,
use_api_service=not no_api)
            nih_rf = NIHResourceFinder(
                date=id_date, orcid=id_orcid, issn=id_issn,
pmid=valid_doi, use_api_service=not no_api )
            self.rf_handler = ResourceFinderHandler( [nih_rf] )
```

Script 28. FileDataHandler - Class constructor. *FileDataHandler class constructor, extended with the inclusion of the new source class "noci" : NIHCitationSource .*

Similar codebase extensions affected all the resource finder classes.

First of all, the `id_type` parameter was added to the arguments of the constructor of the abstract class `ResourceFinder`, in order to call the correct identifier manager for the type of the identifier used by each specific API service (see Script 29).

```
if hasattr( self, 'use_api_service' ):
    if id_type is OCIManager.doi_type:
        print( id.csv_path )  # None
        self.dm = DOIManager( id, self.use_api_service )
    elif id_type is OCIManager.pmid_type:
        self.pm = PMIDManager( id, self.use_api_service )
    else:
        print("The id_type specified is not compliant")
else:
    if id_type is OCIManager.doi_type:
        self.dm = DOIManager( id )
    elif id_type is OCIManager.pmid_type:
        self.pm = PMIDManager( id )
    else:
        print("The id_type specified is not compliant")
```

Script 29. ResourceFinder - id_type parameter inclusion. *id_type parameter inclusion among the arguments of the constructor of the abstract class ResourceFinder, in order to differentiate the process with respect to the type of identifier used by each specific API service.*

The subclass `APIIDResourceFinder` inherits the `id_type` parameter from the abstract superclass `ResourceFinder`. This information is then used for splitting the process into

alternative paths for those methods implementing normalization and validation of the identifiers. Indeed, these two operations have id-specific implementations in `PMIDManager` and `DOIManager` class, and the correct service must be called with respect to the type of identifier managed by the resource (see Script 30). In order to do that, the `APIIDResourceFinder` class uses the instance variables `self.pm` e `self.dm`, inherited from the constructor of the `ResourceFinder` class .

```
    def is_valid(self, id_string):
        if self.id_type == OCIManager.doi_type:
            return self.dm.is_valid(id_string)
        elif self.id_type == OCIManager.pmid_type:
            return self.pm.is_valid(id_string)
        else:
            print("The id_type specified is not compliant")

     def normalise(self, id_string):
        if self.id_type is OCIManager.doi_type:
            return self.dm.normalise(id_string, include_prefix=True)
        elif self.id_type is OCIManager.pmid_type:
            return self.pm.normalise(id_string, include_prefix=True)
        else:
            print("The id_type specified is not compliant")
```

Script 30. APIIDResourceFinder - id_type parameter inclusion. *Process differentiation in both the is_valid and normalise methods of the APIIDResourceFinder, with respect to the type of identifier.*

Since the `id_type` parameter was added in the `ResourceFinder` superclass constructor, the argument was also coherently defined in the constructors of all the classes extending `ResourceFinder`, nominally: `ORCIDResourceFinder`, `CrossrefResourceFinder` and `DataCiteResourceFinder`. The extension can be noticed in the script below (see Script 31), showing the `DataCiteResourceFinder` constructor as an example case.

```
class DataCiteResourceFinder(ApiIDResourceFinder):
    def __init__(self, date=None, orcid=None, issn=None, doi=None,
use_api_service=True):
        self.api = "https://api.datacite.org/dois/"
        self.use_api_service = use_api_service
        super(DataCiteResourceFinder, self).__init__(date=date,
orcid=orcid, issn=issn, id=doi, id_type=OCIManager.doi_type,
use_api_service=use_api_service)
```

Script 31. DataCiteResourceFinder - id_type parameter inclusion. *Inclusion of the id_type parameter in the constructor of the DataCiteResourceFinder class.*

5.4 Mapping: a METAID-based System

The current section is dedicated to the discussion of the technical aspects concerning the mapping system implemented to unequivocally identify entities in OpenCitations, independently from the type of identifier originally used for describing the citation data ingested from the sources.

Even if currently the only mapping data available are the PMID-DOI associations provided in the iCite Metadata dataset, the system is structured to be easily updatable in case new mapping information is obtained. Further, even if until now DOIs and PMIDs are the only two types of identifiers managed by the mapping system, minimal adaptations will be required in case a new type of identifier is introduced in OpenCitations Infrastructure. However, the implementation of a system open to potential corrections caused the necessity of extending the codebase with extra functionalities to both amend CSV support files and remove invalidated data from the triplestore.

Coherently with the data populating the OpenCitations Indexes, the mapping information is stored as RDF triples and uploaded to a triplestore. Each triple datum is composed as follows:

a) *Subject*. In each triple datum, the subject is represented by the identifier of a bibliographic entity, currently expressed either as DOI or PMID. The subject is expressed by the identifier string preceded by the id-specific URI prefix, which is "https://pubmed.ncbi.nlm.nih.gov/" for PMIDs and "http://dx.doi.org/" for DOIs.

b) *Predicate*. The only predicate selected to represent the relation intercurring between an identifier and the METAID identifying the same entity is `DCTERMS.relation`[122], which is aimed at locating by the use of a URI the resource related to the subject.

c) *Object*. The object is composed by the METAID identifier, preceded by the uri prefix "https://w3id.org/oc/meta/br/060"

Once the store is populated, it can be queried with the SPARQL language. In this way, it can be easily assessed whether or not two identifiers refer to the same entity by checking if they share the same triple object, and thus are associated with the same METAID.

5.4.1 Mapping Premises: The METAID

The idea of homogeneously expressing all the citation data managed by OpenCitations with internally assigned identifiers precedes the work of this thesis and the arose necessity of a PMID-DOI mapping.

[122] 'DCMI Metadata Terms', accessed 14 February 2022, https://www.dublincore.org/specifications/dublin-core/dcmi-terms/.

In 2020, Fabio Mariani graduated in Digital Humanities and Digital Knowledge at the University of Bologna with a dissertation titled *Development of OpenCitations Meta: methodology and implementation*[123]. The project consisted of the development of a software for data cleaning, entity deduplication, and creation of OpenCitations Data Model-compliant RDF files, for a better identification of the resources involved in citation data and a more efficient metadata retrieval. The term "METAID" was used for the first time in Mariani's thesis, as the name of the internal identifier for uniquely labeling each entity managed by the OpenCitations Meta software (Mariani 2020, 39).

The structure of a METAID identifier includes the prefix "`br/060`", which is common to all the METAIDs, and a distinctive number assigned to the entity, according to the criterion of sequentiality (see Figure 18).

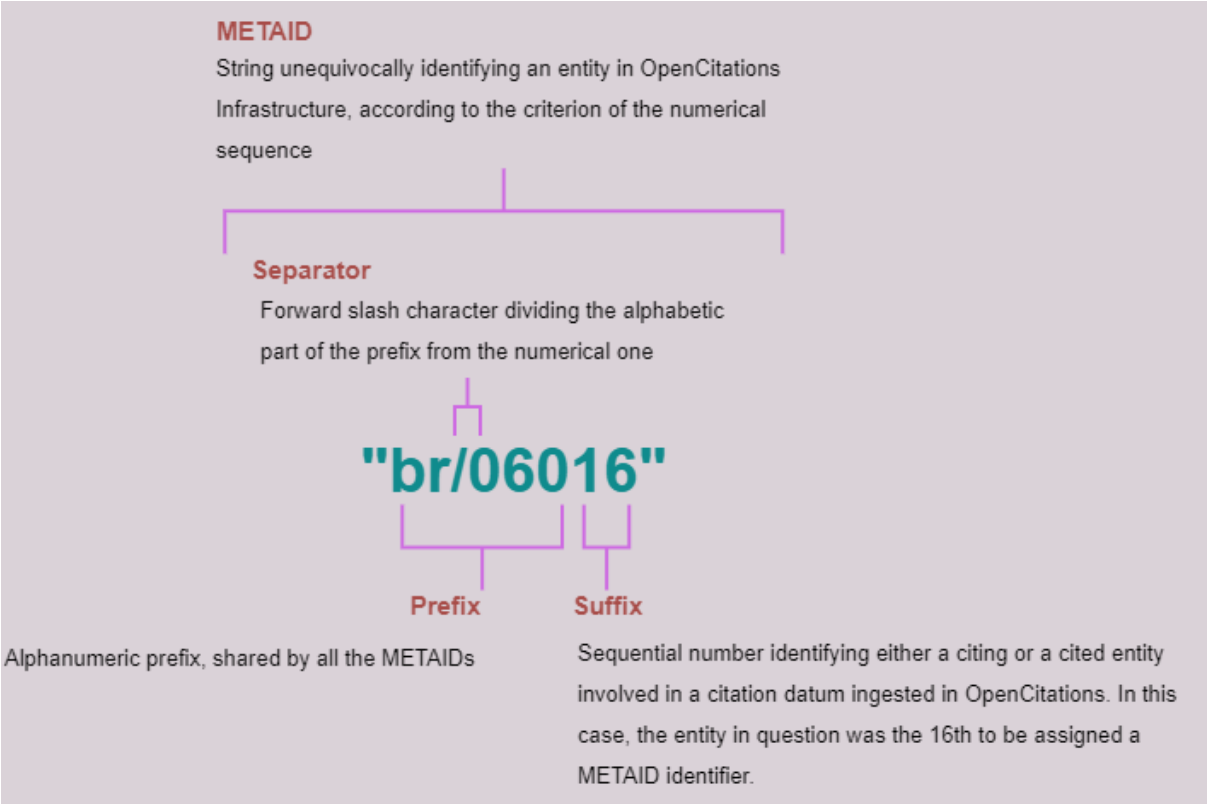

Figure 18. METAID structure. *Structure of the METAID identifier, consisting of two main parts: the prefix "br/060", which is common to all the METAID instances, and an integer number suffix, unequivocally identifying the specific entity.*

[123] Fabio Mariani, 'Development of OpenCitations Meta: Methodology and Implementation' (2020).

5.4.2 Development of Mapping Software Components

This subsection delves into the implementation detail of the software components for the mapping files generation, assigning a METAID to each processed identifier, storing the ID-to-METAID pairings information in RDF format, and uploading the triples to the mapping triplestore.
First, we introduce the code of the `Mapping` class, implementing the core functionalities of the mapping process. Then, we present the `Nocimapping` class, which was developed to extract mapping data from the iCite Metadata source material, in order to generate a CSV file to be used as input to run the Mapping process. The conclusive paragraphs of this subsection are dedicated to the side codebase expansions implementing correction functionalities to handle the possibility that new mapping information potentially invalidates previously defined id-to-METAID pairings.

Mapping; Associating Each Id To a Metaid

`Mapping` is the brand new class developed with the aim of generating an id-to-METAID mapping system for uniquely identifying entities in OpenCitation and tracing back to the potentially multiple ids referring to the same object. At the end of each process, this class produces two RDF files, one containing the new associations to be uploaded to the datastore, and another to store the triples of the id-to-METAID pairings invalidated during the process, in light of new mapping data.
The core method of the `Mapping` class is `process`. It manages the main aspects of the mapping pipeline, including new METAID generation, METAID invalidation and reassignment, support file updating and output CSV files generation. The method takes as input the following arguments:

1) `input_dir1`: a directory where the data generated by launching the `cnc.py` global process for the population of a given index are stored.
2) `input_dir2`: a directory where the mapping files generated by a source-specific and id-specific tool are stored. Currently, the only implemented software component for mapping data extraction from a specific source is `Nocimapping`. This class uses the iCite Metadata dataset to extract PMID-to-DOI mapping information and eventually store it in a CSV file.
3) `midmcsv`: a CSV file containing mapping data between ids and METAIDs, potentially generated in a previous process run. In the case of the first launch of the process, the CSV file is generated on the fly.

4) `ctncsv`: a CSV file containing mapping data between previously assigned (now invalidated) METAIDs, and the corresponding valid METAIDs. Indeed, each time new mapping information leads to the reassignment of a new METAID to an identifier, the METAID previously associated to the same id is invalidated, and the datum refering the invalidated METAID to the replacing METAID is stored into the CSV file. In case of a first launch of the process, the CSV file is generated on the fly.
5) `lmid`: a TXT file containing a numerical string representing the last assigned METAID. At the first run of the process, the file can either be initialized with the digit "0" or, in case it does not exist at all, be created on the fly.
6) `output_dir`: a directory for the storage of the two main output files of the process: the RDF n-triples files respectively containing the new mapping data and the invalidated mapping data.

While running the process, all the support files could be accessed, updated, and also corrected, in case of the invalidation and reassignment of a METAID.

Starting from the CSV citation data generated by executing the script for the indexes' population (i.e., `cnc.py`), the software processes the CSV file row by row and assigns to the variable `citing` the identifier of the citing entity, and to the variable `cited` the identifier of the cited entity.

The main phase of the process starts by accessing the mapping files and checking for the presence of the identifiers of both the citing and the cited entity , in order to assess whether the identifiers refer to the same entity as other identifiers of different types (see Script n.32).

```
for index, row in f.iterrows():
    # check presence of multiple ids for the same publication
    if citing in row["id"] or citing in row["value"]:
        print( "Multiple ids for same publication found" )
        same_id_as.add( row["id"] )
        same_id_as.add( row["value"] )
    else:
        print( "No multiple ids found for:", citing )

    # check presence of multiple ids for the same publication (cited)
    if cited in row["id"] or cited in row["value"]:
        print( "Multiple ids for same publication found" )
        same_id_as_cited.add( row["id"] )
        same_id_as_cited.add( row["value"] )
    else:
        print( "No multiple ids found for:", cited )
```

Script 32. Mapping - Id-to-id mapping files check. *Mapping class: mapping files access and check of the presence of the identifiers of both the citing and the cited entity.*

Once all the correspondent ids identifying the same entity are gathered in a set named `same_id_as`, the process retrieves all the pre-assigned METAIDs paired with any of the corresponding ids, by consulting the support file mapping ids to assigned METAIDs, i.e. `metaid_mapping` (see Script 33). Currently, only two types of identifiers are managed by the system (i.e., PMID and DOI), and thus it is possible to retrieve at most one METAID for each of the two potentially paired identifiers, for a maximum of two METAIDs previously unknowingly assigned to the same entity. At the end of this step, all the METAIDs assigned to at least one of the paired identifiers are stored in a set named `previous_metaids`.

```
previous_metaids = set()
for id in same_id_as:
    assigned_metaid = metaid_mapping.get_value( id )
    if assigned_metaid is not None:
        for id in assigned_metaid:
            # verify the type of the id
            assigned_metaid.remove( id )
            assigned_metaid.add( int( id ) )
        previous_metaids.update( assigned_metaid )
```

Script 33. Mapping - Id-to-METAID mapping file check. *Mapping class: id-to-METAID mapping file check, aimed at collecting previously assigned METAID for correspondant identifiers of different type, if any.*

At this point, we can have three possible scenarios:

1) *Only one METAID was already assigned* to one of the correspondent ids for the same publication. In this case, all the other ids for the same publication (if any) must be assigned the same METAID.
2) *Multiple METAIDs were already assigned* to correspondant ids for the same publication. In this case, all the ids must be assigned/re-assigned the same METAID, which is the lowest among the assigned ones. The other metaids are invalidated.
3) *No METAID* is assigned to any of the ids identifying the same entity. A new METAID is assigned to all the corresponding identifiers.

Once the identifier of the citing entity has been processed, the workflow is repeated for the cited entity as well.

At the end of the process, the output data are transposed to RDF files. Up to this moment, the output material was stored into CSV files, however, in order to update the triplestore, the data must be

exposed in RDF format. For this reason, `process` calls the ancillary method `create_rdf_from_csv`, which takes as input a folder to store the output files, and the two CSV files containing the just processed information about the new id-to-METAID associations and about the invalidated pairings, if any. Once the creation of the RDF files is completed, the two CSV files are deleted (see Script 34).

```
create_rdf_from_csv(new_mappings, deleted_mappings, output_dir)
os.remove(new_mappings)
os.remove(deleted_mappings)
```

Script 34. Mapping - CSV files deletion. *Mapping class: after the creation of the output RDF files, the CSV files generated during the process are deleted.*

The function `create_rdf_from_csv` performs an additional check, in order to prevent the possibility of including in the new mapping file any data appearing also among the invalidated pairings. Despite the unlikeliness of the event, the process provides the possibility to be run with more mapping files as input, providing mapping information expressed with more than two different types of identifiers. In this case, there is the possibility that the same id-to-METAID mapping datum is both created and invalidated during the same process run. In this case, the data can be part of the triples to ingest in the triplestore as well as part of the triples to remove from the triplestore. To avoid this situation, the data appearing in both the files of mapping (i.e., creation and invalidation) are not transposed to the RDF n-triples files at all. The full script of the `create_rdf_from_csv` method is provided below (see Script 35).

```
def create_rdf_from_csv(self, csvfile_add, csvfile_delete, output_dir):
    del_folder = "%striples_to_remove" % (sep)
    add_folder = "%striples_to_add" % (sep)
    del_folder_path = output_dir + del_folder
    add_folder_path = output_dir + add_folder

    if not exists(del_folder_path):
        makedirs(del_folder_path)

    if not exists(add_folder_path):
        makedirs(add_folder_path)

    g = Graph()
    gdel = Graph()

    doi_uri_pref = "http://dx.doi.org/"
    pmid_uri_pref = "https://pubmed.ncbi.nlm.nih.gov/"
    metaid_uri_pref = "https://w3id.org/oc/meta/br/060"
    predicate = DCTERMS.relation
```

```
    cur_date = datetime.today().strftime( '%Y-%m-%d' )

    with open(csvfile_delete, 'r' ) as deletefile:
        del_existing_lines = [line for line in reader(deletefile,
delimiter=',' )]

    discarded_rows = []
    with open(csvfile_add, 'r' ) as addfile:
        reader_addfile = reader( addfile, delimiter=',' )
        for row in reader_addfile:
            if row in del_existing_lines:
                discarded_rows.append(row)
            elif row:
                id = row[0].strip()
                if id.startswith("pmid:"):
                    id_uri =URIRef(pmid_uri_pref+id[5:])
                elif id.startswith("doi:"):
                    id_uri =URIRef(doi_uri_pref+id[4:])
                metaid_uri = URIRef(metaid_uri_pref+row[1])
                if id_uri and metaid_uri:
                    g.add((id_uri, predicate, metaid_uri))
                    print("added triple:", id_uri, predicate,
metaid_uri)

        filename = "new_mappings"
        rdf_filename = filename + "_" + cur_date + ".ttl"
        g.serialize(destination= add_folder_path + "%s" % (sep) +
rdf_filename, format='nt11')

    for row in del_existing_lines:
        if row and row not in discarded_rows:
            id = row[0].strip()
            if id.startswith( "pmid:" ):
                id_uri = URIRef( pmid_uri_pref + id[5:] )
            elif id.startswith( "doi:" ):
                id_uri = URIRef( doi_uri_pref + id[4:] )
            metaid_uri = URIRef( metaid_uri_pref + row[1] )
            if id_uri and metaid_uri:
                gdel.add( (id_uri, predicate, metaid_uri) )
                print( "added triple to delete file:", id_uri,
predicate, metaid_uri )

    filename = "del_mappings"
    rdf_filename = filename + "_" + cur_date + ".ttl"
    gdel.serialize(destination= del_folder_path + "%s" % (sep) +
rdf_filename, format='nt11')
```

Script 35. Mapping - RDF generation method. *Mapping class: script of the create_rdf_from_csv method.*

Nocimapping; Generation Of The First Mapping File

`Nocimapping` class was developed to extract PMID-to-DOI mapping data from the iCite Metadata dataset provided by the National Institute of Health and store it in a two-column CSV file

(see Script 36). The main process is run by the method `process`, which derives the input material from a directory storing the iCite Metadata CSV file and stores the processed data into a specific output directory. Since the mapping information is provided among many other data in the source CSV file, the extraction process is simple and only aims at generating a lighter reduced version of the input file, which maintains only the pieces of information which are exploitable in the mapping process. The whole process is performed using the Pandas Python library[124] to access and extract data. The OpenCitations Index `CSVManager` is used to create and fill out the output file. The generated output is going to be used for the general mapping process. The process of id-to-id mapping data extraction is source-specific and id-type specific, thus the code implemented for extracting PMID-to-DOI information from iCite Metadata is reusable to a limited extent. Therefore, in case new mapping material will be integrated in the OpenCitations Infrastructure, brand-new mapping classes should be developed for the purpose, in order to extract from new data sources the same type of information and to store it in similarly structured output files.

```
    def process(self):
        if not exists( self.output_dir ):
            makedirs( self.output_dir )

        mapping_folder = self.output_dir + "/NIH_mapping"
        if not exists( mapping_folder ):
            makedirs( mapping_folder )

        cur_date = datetime.today().strftime( '%Y-%m-%d' )
        pmid_doi_mapping = CSVManager( mapping_folder + sep +
"pmid_doi_mapping_" + cur_date + ".csv" )
        valid_pmid = CSVManager( "index/test_data/nih_glob1" + sep +
"valid_pmid.csv" )
        valid_doi = CSVManager( "index/test_data/crossref_glob" + sep +
"valid_doi.csv" )
        pmid_manager = PMIDManager( valid_pmid )
        doi_manager = DOIManager( valid_doi )
        all_files, opener = self.get_all_files()
        len_all_files = len( all_files )

        # Read all the CSV file in the NIH dump to create the main
information of all the indexes
        print( "\n\n# Add valid PMIDs from NIH metadata" )
        for file_idx, file in enumerate( all_files, 1 ):
            df = pd.DataFrame()
            for chunk in pd.read_csv( file, chunksize=1000 ):
                f = pd.concat( [df, chunk], ignore_index=True )
                print( "Open file %s of %s" % (file_idx, len_all_files)
)
```

[124] 'Pandas Documentation — Pandas 1.4.1 Documentation', accessed 14 February 2022, https://pandas.pydata.org/docs/index.html.

```
                for index, row in f.iterrows():
                    print( index )
                    pmid = str( row["pmid"] )
                    doi = str( row["doi"] )
                    if pmid != "" and doi != "":
                        if pmid_manager.is_valid( pmid ) and
doi_manager.is_valid( doi ):
                            pmid = pmid_manager.normalise( pmid,
include_prefix=True )
                            doi = doi_manager.normalise( doi,
include_prefix=True )
                            pmid_doi_mapping.add_value( pmid, doi )
```

Script 36. Nocimapping - Process method. *Nocimapping class: process method script, developed to extract PMID-to-DOI mapping data from the iCite Metadata dataset.*

5.4.3 Mapping: Side Codebase Integrations and Adaptations

CSV Manager Expansion

`CSVManager` is the OpenCitations class for CSV management. It provides methods for creating new files, adding and accessing values, but not for overriding data once included. The inclusion of the mapping system in the OpenCitations Infrastructure caused the necessity to handle the issue of the METAID reassignment and the consequent data correction in CSV support files. Because of that, the `substitute_value` method was implemented, whose script is provided below (see Script 37). This function substitutes values (stored in the field "`value`") associated with the specified identifiers (stored in the field "`id`") in two-column CSV files . The method corrects id-to-METAID pairs in case previously defined associations are invalidated by new mapping information.

```
def substitute_value(self, id_string, value):
   if id_string in self.data and str( value ) not in
self.data[id_string]:
       self.data[id_string].clear()
       self.data[id_string].add( str( value ) )
       filename = self.csv_path
       tempfile = NamedTemporaryFile(mode='w', newline='',
delete=False)
       fields = ["id", "value"]
       with open( filename, 'r', encoding="utf8", newline='') as
csvfile, tempfile:
           reader = csv.DictReader(csvfile, fieldnames=fields)
           writer = csv.DictWriter(tempfile, fieldnames=fields)
           for row in reader:
               if row["id"] == id_string and str(value) !=
row["value"]:
```

```
                print( 'updating row', row['id'] )
                row["id"], row["value"] = id_string, str(value)
            row = {"id": row["id"], "value": row["value"]}
            writer.writerow(row)
    shutil.move( tempfile.name, filename )
```

Script 37. CSVManager - Value substitution method. *CSVManager class: substitute_value method creation for correcting CSV files.*

Upload Expansion

Another way to manage the correction issues that arose after the integration of the mapping system is through a removal of the data from the triplestore. The original objective of the `upload.py` script was only to upload triples to the triplestore, however, similarly to the previous additions made to the `CSVManager` class, also in this case the possibility of invalidation for previously stored data caused the necessity of a recovery system, permitting the maintenance of a correctly updated triplestore.

The function implemented for the purpose is `remove` (see Script 39), which was developed to perform the inverse action of the `add` function (see Script 38). However, since SPARQL query language does not provide an inverse of the LOAD operation[125] (for deleting with a single query all the triples included in an RDF file) we parse to a graph all the triples of the input file containing the data to be removed and iterate over it using the DELETE DATA operation to delete specific triples.

```
def add(server, g_url, f_n, date_str, type_file):
    print("ADD")
    server = SPARQLWrapper(server)
    server.method = 'POST'
    my_query = 'LOAD <' + pathlib.Path(abspath(f_n)).as_uri() + '> INTO
GRAPH <' + g_url + '>'
    server.setQuery(my_query)
    server.query()

    with open("updatetp_report_%s_%s.txt" % (type_file, date_str), "a",
              encoding="utf8") as h:
        h.write("Added file '%s'\n" % f_n)
```

Script 38. Update.py - Add function. *update.py add method for loading triples to a triplestore, using the LOAD SPARQL query operation.*

```
def remove(server, g_url, f_n, date_str, type_file, n):
```

[125] 'SPARQL 1.1 Query Language', accessed 14 February 2022, https://www.w3.org/TR/sparql11-query/.

```
    print("REMOVE")
    server = SPARQLWrapper(server)
    server.method = 'POST'
    file_path = pathlib.Path(abspath(f_n)).as_uri()

    #remove all the triples from the specified file
    g = Graph()
    g.parse(file_path, format="nt11" )

    i = 0
    triples_group = ""

    for index, (s, p, o) in enumerate(g):
        triple = "<" + str( s ) + ">" + "<" + str( p ) + ">" + "<" +
str( o ) + ">" + "."
        i += 1
        if i == int(n):
            triples_group = triples_group + triple + " "
            i = 0
            my_query = 'DELETE DATA {GRAPH <' + g_url + '> {' +
triples_group + '} }'
            server.setQuery(my_query)
            server.query()
            triples_group = ""

        else:
            triples_group = triples_group + triple + " "

    if triples_group != "":
        triples_group = triples_group + triple + " "
        my_query = 'DELETE DATA {GRAPH <' + g_url + '> {' +
triples_group + '} }'
        server.setQuery( my_query )
        server.query()

    with open("updatetp_report_%s_%s.txt" % (type_file, date_str), "a",
              encoding="utf8") as h:
        h.write("Removed triples from file '%s'\n" % f_n)
```

Script 39. Update.py - Remove function. *update.py remove method for deleting already stored triples in a triplestore, using the DELETE DATA SPARQL operation .*

5.5 API Configuration File and Id-specific Metadata Retrieval

In order to make the data of NOCI more accessible to inexperienced SPARQL users, a specific configuration file[126] has been defined to enable such operation through an API service created with

[126] ariannamorettj, *Restful API Manager Over SPARQL Endpoints (RAMOSE)*, Python, 2022, https://github.com/ariannamorettj/ramose/blob/d3c4ee1696943f41c194aacbc9bb23192a73e099/test/test_m4.hf.

the OpenCitations tool RAMOSE[127]. In addition to that, the python file `indexapi.py`[128] was extended with a parser function (i.e., `__nih_parser`) for the PMID metadata retrieval via the National Institute of Health PubMed API service[129].

The API configuration file of NOCI is similar to the one defined for COCI API service.

Indeed, of all the operations implemented for COCI (i.e., `references`, `citations`, `citation`, `metadata`, `citation-count`, `reference-count`), `metadata` is the only one requiring access to id-type specific services.

5.5.1 Metadata

The definition of the `metadata` API operation for NOCI required some codebase expansions. In particular, the new parser function `__nih_parser` has been defined to retrieve metadata for PMIDs using the National Institute of Health PubMed API service[130] (see Script 40). The script is modeled following the Crossref and DataCite parsers, it takes as input an identifier and returns a list containing:

1) a string including the names of the authors for a specific bibliographic entity (different authors are separated by a semicolon)
2) the year of publication of the bibliographic entity
3) the title of the bibliographic entity
4) the title of the journal/source of the bibliographic entity
5) the volume in which the bibliographic entity is included
6) the issue number in which the bibliographic entity appears
7) the numbers of the pages in which the bibliographic entity appears
8) the ISSN and/or ISBN identifier of the journal/source of the bibliographic entity

Each of the abovementioned elements is retrieved using an ancillary function modeled on the `NIHResourceFinder` methods. This function reuses the same approach adopted for the other OC indexes, adapted to the new data source and response format: it extracts textual data from the HTML response of the API request by means of the tools provided by the Beautiful Soup Python

[127] *Restful API Manager Over SPARQL Endpoints (RAMOSE)*, Python (2018; repr., OpenCitations, 2022), https://github.com/opencitations/ramose.Restful API Manager Over SPARQL Endpoints (RAMOSE), Python (2018; repr., OpenCitations, 2022), https://github.com/opencitations/ramose.
[128] ariannamorettj, *Restful API Manager Over SPARQL Endpoints (RAMOSE)*, Python, 2022, https://github.com/ariannamorettj/ramose/blob/d3c4ee1696943f41c194aacbc9bb23192a73e099/test/indexapi.py.
[129] 'PubMed', accessed 31 January 2022, https://pubmed.ncbi.nlm.nih.gov/.
[130] 'PubMed', accessed 31 January 2022, https://pubmed.ncbi.nlm.nih.gov/.

library[131]. Then, the required data are extracted using the Re Python library for regular expressions[132].

```
def __nih_parser(pmid):
    api = "https://pubmed.ncbi.nlm.nih.gov/%s"
    pmid_sep = str(pmid) + "%s"
    display_opt = "/?format=pubmed"

    try:
        r = get(api % pmid_sep % display_opt,
                headers={"User-Agent": "NOCI REST API (via
OpenCitations - "
                                       "http://opencitations.net;
mailto:contact@opencitations.net)"}, timeout=30)
        if r.status_code == 200:
            r.encoding = "utf-8"
            soup = BeautifulSoup(r.text, features="lxml")
            body = str(soup.find(id="article-details"))

            authors = _get_author_nih(body)


            year = ""
            nih_date = _get_date_nih(body)
            if nih_date is not None:
                year = __normalise(nih_date)

            title = ""
            nih_title = _get_title_nih(body)
            if nih_title is not None:
                title = __create_title(nih_title)

            source_title = ""
            nih_cont_title = _get_cont_title_nih(body)
            if nih_cont_title is not None:
                source_title = __create_title(nih_cont_title)

            volume = ""
            volume_title = _get_volume_nih(body)
            if volume_title is not None:
                volume = volume_title

            issue = ""
            nih_issue = _get_issue_nih(body)
            if nih_issue is not None:
                issue = __normalise(nih_issue)

            page = ""
            pagination = _get_page_nih(body)
            if pagination is not None:
```

[131] 'Beautiful Soup Documentation — Beautiful Soup 4.9.0 Documentation', accessed 12 February 2022, https://www.crummy.com/software/BeautifulSoup/bs4/doc/.
[132] 'Re — Regular Expression Operations — Python 3.10.2 Documentation', accessed 13 February 2022, https://docs.python.org/3/library/re.html.

```
                page = __normalise(pagination)

            source_id = ""
            issn = _get_issn_nih(body)
            if issn is not None:
                source_id = issn

            return ["; ".join(authors), year, title, source_title,
volume, issue, page, source_id]
```

Script 40. RAMOSE indexapi.py extension. *RAMOSE indexapi.py codebase expansion: __nih_parser function script, aimed at retrieving data from the National Institute of Health PubMed API service*

The parser returns back a JSON block, which contains the identifier of the bibliographic entity, the identifiers of the references and citations, the number of citations received by the entity, and - if available - the url of its full-text if open[133].

[133] *Restful API Manager Over SPARQL Endpoints (RAMOSE)*, Python (2018; repr., OpenCitations, 2022), https://github.com/opencitations/ramose.

6. Computational Cost and Process Evaluation

This chapter is dedicated to the process evaluation and computational cost assessment of the OpenCitations Index software execution for the ingestion of NOCI. This is done through a descriptive statistics analysis to evaluate the performances of the new software components. The efficiency of the software has been evaluated with respect to the execution time of the process analyzed with respect to the number of the processed data and the memory required for its storage. The analysis starts first with a declaration of the hardware and software materials used in this elaboration, such as the type of processor, operative system, RAM, and internet connection.

6.1 Hardware and Software Characteristics

Evaluating a software performance is a complex task which requires a consideration of all the factors that might have an influence on the process execution.

An overview of the hosting machine characteristics (hardware and software) is fundamental for the interpretation and understanding of the elaboration and outputs.

The machine used in our analysis is a Dell Inspiron 15, 5000 series, and its main influential characteristics with respect to the evaluation of the software efficiency are listed below.

- Processor: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
- 8,00 GB System memory (RAM) installed, of which (7,90 GB usable)
- Windows 64-bit operating system, with an x64-based processor

In addition to the machine features (listed above), another important aspect to consider is the internet connection. Indeed, since the process uses API calls to retrieve the citation metadata which are not already in the system, the quality and the type of the connection used when running the code is an important factor. As for the evaluation of the efficiency of this software, two different types of connections have been adopted, for comparative purposes: (1) a mobile internet connection, offering a 4G network coverage with download and upload speed up to 30 Mbps, and (2) a Wi-Fi 5 connection with a 5GHz network band and download/upload speed up to 390 Mbps.

6.2 Data Sampling

In order to evaluate the performance of the software, random samples of the most recent snapshot[134] of NIH Open Citation Collection[135] were extracted using the script below (see Script 41). We used the Pandas Python Library[136] to read the input file and store the collection into CSV sample files. The Random Python module[137] was used to select representative data, fairly approximating the entire corpus with an approach in which each datum has the same chances of being selected[138]. To this end, the function sampler takes in input a CSV file and a number, where the number defines the proportional dimension of the output file randomly selected from the given CSV. For example, the execution of `sampler("open_citation_collection.csv", 0.001)` will generate a reduced version of `"open_citation_collection.csv"`, consisting of one randomly selected datum every thousand data contained in the input source file.

The NIH-OCC contains 670 million citation links, provided in a 11.7 GB CSV file, and the iCite Metadata collection, which is provided both as a compressed CSV and JSON file. For the present study, for computational convenience in relation to the huge size of the files, we considered the lighter CSV version, occupying 20.6 GB of memory. The software efficiency results have been generated using repeated process executions on differently resized data samples, either extracted from the National Institute of Health Open Citation Collection or from the annex iCite Metadata Collection.

```
def sampler(csv_file, n):
     df = pd.read_csv(csv_file, header=0, skiprows=lambda i: i>0 and
random.random() > n)
     df.to_csv( "occ_reduced.csv", index = None)


if __name__ == '__main__':
     sampler("open_citation_collection.csv", 0.001)
```

[134] 'ICite Database Snapshot 2022-01' (The NIH Figshare Archive, 10 February 2022), https://doi.org/10.35092/yhjc.19149149.v1.
[135] ICite, Hutchins, B. Ian, and Santangelo, George, 'ICite Database Snapshots (NIH Open Citation Collection)', 2022, https://doi.org/10.35092/YHJC.C.4586573.V26.
[136] 'Pandas Documentation — Pandas 1.4.1 Documentation'. Accessed 14 February 2022. https://pandas.pydata.org/docs/index.html.
[137] 'Random — Generate Pseudo-Random Numbers — Python 3.10.2 Documentation'. Accessed 17 February 2022. https://docs.python.org/3/library/random.html.
[138] 'Technically Speaking: Why We Use Random Sampling in Reading Research | Iowa Reading Research Center'. Accessed 17 February 2022. https://iowareadingresearch.org/blog/technically-speaking-random-sampling.

Script 41. Random sampling from CSV function. *Function for generating random samples from input CSV files.*

6. 3 Computational Cost Evaluation: Citation Data Generation

The first evaluation of the process performances on the NIH-OCC data was performed on a 839 bytes CSV sample, containing 45 citation data. The OpenCitations Index main process produces citation data in three format, i.e. CSV, RDF and Scholix, together with two output files storing provenance data both in RDF and CSV. In addition to that, the workflow execution produces three CSV support files, storing data about dates, ISSNs, and validity state of the processed identifiers. In this case, from the 839 bytes sample input, the process harvested:

- 7662 bytes of CSV support files, respectively divided in 2481 bytes of data concerning dates, 3312 bytes for ISSNs, and 1869 bytes for identifiers validity;
- 86073 bytes of harvested citation data, divided in 3685 of CSV, 39545 of RDF, and 42843 of Scholix.
- 79322 bytes of provenance data, consisting of a 7782 bytes CSV file and a 71540 bytes RDF file.

Run	Citations	Time elapsed (seconds)	Seconds per citation	Type of connection
1	45	157.5370	3.5008	mobile data
2	45	62.3992	1.3866	mobile data
3	45	59.2242	1.3160	mobile data
4	45	72.0243	1.6005	mobile data
5	45	62.2751	1.3838	mobile data
6	45	50.5852	1.1241	mobile data
7	45	51.4273	1.1428	mobile data
8	45	43.9039	0.9756	mobile data
9	45	42.9983	0.9555	mobile data
10	45	44.6104	0.9913	mobile data

Table 9. Computational cost evaluation, 45 citations. *Table of data gathered in ten process runs with a sample CSV input file, containing 45 citations, and a mobile data internet connection.*

The first set of tests consists of ten runs of the process with mobile data internet connection. As shown in Table 9, the computational time spent for processing a citation datum ranges from a maximum of 3.50 seconds to a minimum of 0.95. However, the first run clearly presented an outlier negative performance, and thus a balanced metrics should be adopted for the process evaluation. For assessing the efficiency of the process, we used the tools provided by the Timeit Python module[139]. With respect to the best criterion to be adopted for assessing the efficiency of a process, in the documentation of the module, the arithmetic mean is deprecated for this purpose:

> It's tempting to calculate mean and standard deviation from the result vector and report these. However, this is not very useful. In a typical case, the lowest value gives a lower bound for how fast your machine can run the given code snippet; higher values in the result vector are typically not caused by variability in Python's speed, but by other processes interfering with your timing accuracy. So the `min()` of the result is probably the only number you should be interested in. After that, you should look at the entire vector and apply common sense rather than statistics.[140]

Coherently, solutions less sensitive to the outliers can be adopted, such as the median. With respect to the process evaluation exposed in Table 9, it can be assessed that the execution takes a median time of 55.3257 seconds (i.e. 0.9 minutes), which means 1.229 seconds for processing a citation. However, the above mentioned passage of the Python Timeit module documentation suggests the reasonability of adopting the best time obtained as representative of the optimal performance possibilities of the machine. Following this assumption, it is legit to take into a major account the lower datum obtained, which is of 42.9983 seconds (0.7 minutes), corresponding to 0.9555 seconds for processing a citation.

Run	Citations	Time elapsed (seconds)	Seconds per citation	Type of connection
1	669	685.5343	1.0247	mobile data
2	669	677.0614	1.0120	mobile data
3	669	689.2819	1.0303	mobile data
4	669	688.0632	1.0284	mobile data
5	669	707.8006	1.0579	mobile data
6	669	452.7027	0.6766	Wi-Fi 5

[139] 'Timeit — Measure Execution Time of Small Code Snippets — Python 3.10.2 Documentation', accessed 19 February 2022, https://docs.python.org/3/library/timeit.html.
[140] ibidem

7	669	439.3261	0.6566	Wi-Fi 5
8	669	441.7730	0.6603	Wi-Fi 5
9	669	453.1448	0.6773	Wi-Fi 5
10	669	451.1852	0.6744	Wi-Fi 5

Table 10. Computational cost evaluation, 669 citations. *Table of data gathered in ten process runs with a sample CSV input file, containing 669 citations, and mixed internet connections.*

Starting from 12.1 KB (12391 byte) - 16.0 KB (16384 byte) in memory - of CSV input data, the process generated 1281989 bytes of citation data (54555 from CSV, 590293 from RDF, and 637141 from Scholix), 1183402 bytes of provenance data (115126 from CSV, 1068276 from RDF), and the support files, for a total amount of 2400000 bytes of harvested data.

The process took a median time of 565.1031 seconds (i.e .9.4 min), which means 0.84 seconds per citation. However, the best performances are significantly lower: 7.32 minutes for executing the whole process, with 0.66 seconds to process a citation. Indeed, as it is clearly highlighted in Table 10, the efficiency is highly correlated with the quality of the internet connection. which is a determinant factor when the process resorts to API for retrieving data about the received identifiers.

6.4 Computational Cost Evaluation: Glob Support Files Generation

For what concerns the generation of the support files for NOCI, the data harvesting has a significant computational cost, because of the rich structure of the input iCite Metadata CSV file[141]. The process is particularly expensive not only with respect to the amount of data to elaborate for each citation, but also when considering the time of computation. Indeed, the process was tested with a Wi-Fi 5GHz network band connection in three code runs, which respectively lasted for 2353.9910, 2289.6007, and 2286.5014 seconds: this means a median time of 6.7 seconds per bibliographic entity, for a total duration of 38 min and 9 sec. For what concerns the evaluation of the computational cost with respect to the memory occupied, the iCite Metadata sample contained 338 bibliographic entity data, corresponding to 206 KB (211899 byte) on memory. From this input

[141] 'ICite Database Snapshot 2022-01' (The NIH Figshare Archive, 10 February 2022), https://doi.org/10.35092/yhjc.19149149.v1.

material, the process generated five support files, storing mapping data about:id-to-date associations: CSV file of 7.81 KB (7999 byte);

- id-to-ISSN associations: CSV file of 10.1 KB (10357 byte);
- id-to-ORCID associations: CSV file of 7.09 KB (7264 byte);
- id validity state: CSV file of 157 KB (161188 byte);
- journal-to-ISSN associations: JSON file of 14.7 KB (15067 byte).

7. Conclusions

The aim of this thesis was extending the OpenCitations Index software for the creation of a new index, NOCI – citation data gathered from the National Institute of Health (NIH) Open Citation Collection. Since the entities part of the NIH Open Citation Collection are identified with PMIDs, codebase adjustments to the current software infrastructure of OpenCitations were necessary, to include the new identifier typology (i.e., PMID) into a DOI-based system.

These adjustments introduced the possibility of having data duplication issues, i.e., the possibility of adding the same citation twice (represented as a DOI-to-DOI and PMID-to-PMID citation datum) into the OpenCitations dataset. Therefore, in order to unequivocally identify each citation in the system, the use of a third type of identifier for bibliographic entities, i.e. the METAID, was necessary. The purpose of this identifier is dual: on the one hand, it is used for mapping DOIs and PMIDs of the same entity to a specific METAID pointing to the metadata block identifying such entity; on the other hand, it represents the first step for achieving a unified system of identification for all the citation data regardless of the type of identifier they use.

At implementation level, the population of NOCI required the development of a software extension to manage the new data source and the data gathered, i.e., the NIH Open Citation Collection[142], composed by the citation corpus itself (storing PMID to PMID citations) and the iCite Metadata, containing metadata about bibliographic entities identified by PMIDs. In addition to that, the inclusion of a new type of identifier caused the necessity to adjust the codebase to differentiate the process according to the id type of the entity to elaborate. Coherently, the software was integrated with a PMID manager extension, performing all the id-specific operations previously used to handle DOI identifiers only.

The final result of this thesis project is the creation of a new OpenCitations index, NOCI, populated with citations expressed as RDF triples, stored in a triplestore. Similarly, also the id-to-METAID mapping information will be exposed with the same approach. Therefore, once all the harvested

[142] 'ICite Database Snapshot 2022-01' (The NIH Figshare Archive, 10 February 2022), https://doi.org/10.35092/yhjc.19149149.v1.

data will be uploaded to the triplestore, the DOI-to-PMID mapping information will be easily retrievable via SPARQL queries.

To facilitate the access/retrieval of the new citation data of NOCI. We developed an Application Programming Interface with RAMOSE, i.e. the Restful API Manager Over SPARQL Endpoints released by OpenCitations (Daquino et al. 2022). The implementation of the software expansions for the PMID management and the usage of the new data source involved in the ingestion of NOCI were largely inspired by the parts of the codebase handling the same procedures to the other citation data managed by OpenCitations. The development of the mapping system met an unprecedented need, and therefore was developed from scratch. For this reason, the structure of the class managing the mapping process may be refactored for a better reorganization of the methods. Similarly, the process of id-to-id mapping information extraction from the data sources (currently implemented only for PMID-to-DOI information from NIH iCite Metadata[143]) should be generalized, in order to be easily applicable for the future management of new input materials from other sources.

The contribution of this project to the OpenCitations Infrastructure primarily consisted in the development of tools to ingest and properly manage the new citation data provided by the National Institute of Health (NIH), expressed as PMID-to-PMID citations. However, the potentially most fruitful aspect of the work was the development of an easily adaptable model for future integration of new data based on other types of identifiers. Indeed, both the codebase adjustments and the implementation of new features were performed in view of OpenCitations' further evolutions. Therefore, the ingestion of the NIH Open Citation Collection not only resulted in a concrete increase of the OpenCitations data, but also provided the opportunity to set an efficient environment for the reception of ulterior citations, potentially expressed with different types of identifiers.

[143] Ibidem.

Bibliography

Canese, Kathi, and Sarah Weis. PubMed: The Bibliographic Database. 2013. In: The NCBI Handbook. Bethesda (MD): National Center for Biotechnology Information (US). https://www.ncbi.nlm.nih.gov/books/NBK153385/

Cousijn, Helena, Patricia Feeney, Daniella Lowenberg, Eleonora Presani, and Natasha Simons. 2019. 'Bringing Citations and Usage Metrics Together to Make Data Count'. *Data Science Journal* 18 (1): 9. https://doi.org/10.5334/dsj-2019-009.

Daquino, Marilena, Ivan Heibi, Silvio Peroni, and David Shotton. 2022. 'Creating RESTful APIs over SPARQL Endpoints Using RAMOSE'. Edited by Armin Haller. *Semantic Web* 13 (2): 195–213. https://doi.org/10.3233/SW-210439.

Daquino, Marilena, Silvio Peroni, David Shotton, Giovanni Colavizza, Behnam Ghavimi, Anne Lauscher, Philipp Mayr, Matteo Romanello, and Philipp Zumstein. 2020. 'The OpenCitations Data Model'. In *The Semantic Web – ISWC 2020*, edited by Jeff Z. Pan, Valentina Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, 12507:447–63. Lecture Notes in Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-62466-8_28.

Delgado López-Cózar, Emilio, Enrique Orduña-Malea, and Alberto Martín-Martín. 2019. 'Google Scholar as a Data Source for Research Assessment'. In *Springer Handbook of Science and Technology Indicators*, edited by Wolfgang Glänzel, Henk F. Moed, Ulrich Schmoch, and Mike Thelwall, 95–127. Springer Handbooks. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-02511-3_4.

Elsevier. n.d. 'Content - How Scopus Works - Scopus - | Elsevier Solutions'. Elsevier.Com. Accessed 21 February 2022. https://www.elsevier.com/solutions/scopus/how-scopus-works/content.

Heibi, Ivan, Silvio Peroni, and David Shotton. 2019a. 'Crowdsourcing Open Citations with CROCI -- An Analysis of the Current Status of Open Citations, and a Proposal'. *ArXiv:1902.02534 [Cs]*, June. http://arxiv.org/abs/1902.02534.

Heibi, Ivan, Silvio Peroni, and David Shotton. 2019b. 'Software Review: COCI, the OpenCitations Index of Crossref Open DOI-to-DOI Citations'. *Scientometrics* 121 (2): 1213–28. https://doi.org/10.1007/s11192-019-03217-6.

Hussain, Ijaz, and Sohail Asghar. 2017. 'A Survey of Author Name Disambiguation Techniques: 2010–2016'. *The Knowledge Engineering Review* 32: e22. https://doi.org/10.1017/S0269888917000182.

Hutchins, B. Ian, Kirk L. Baker, Matthew T. Davis, Mario A. Diwersy, Ehsanul Haque, Robert M. Harriman, Travis A. Hoppe, Stephen A. Leicht, Payam Meyer, and George M. Santangelo. 2019. 'The NIH Open Citation Collection: A Public Access, Broad Coverage Resource'. *PLOS Biology* 17 (10): e3000385. https://doi.org/10.1371/journal.pbio.3000385.

López-Cózar, Emilio Delgado, Enrique Orduna-Malea, and Alberto Martín-Martín. 2018. 'Google Scholar as a Data Source for Research Assessment'. *ArXiv:1806.04435 [Cs]*, June. http://arxiv.org/abs/1806.04435.

Mariani, Fabio. 2020. 'Development of OpenCitations Meta: Methodology and Implementation'.

Martín-Martín, Alberto, Mike Thelwall, Enrique Orduna-Malea, and Emilio Delgado López-Cózar. 2021. 'Google Scholar, Microsoft Academic, Scopus, Dimensions, Web of Science, and OpenCitations' COCI: A Multidisciplinary Comparison of Coverage via Citations'. *Scientometrics* 126 (1): 871–906. https://doi.org/10.1007/s11192-020-03690-4.

Peroni, Silvio, Alexander Dutton, Tanya Gray, and David Shotton. 2015. 'Setting Our Bibliographic References Free: Towards Open Citation Data'. *Journal of Documentation* 71 (2): 253–77. https://doi.org/10.1108/JD-12-2013-0166.

Peroni, Silvio, and David Shotton. 2018. 'Open Citation: Definition'. https://doi.org/10.6084/m9.figshare.6683855.v1.

Peroni, Silvio, and David Shotton. 2019. 'Open Citation Identifier: Definition', 147574 Bytes. https://doi.org/10.6084/M9.FIGSHARE.7127816.

Peroni, Silvio, and David Shotton. 2020. 'OpenCitations, an Infrastructure Organization for Open Scholarship'. *Quantitative Science Studies* 1 (1): 428–44. https://doi.org/10.1162/qss_a_00023.

Peroni, Silvio, David Shotton, and Fabio Vitali. 2016. 'A Document-Inspired Way for Tracking Changes of RDF Data'. In *Proceedings of the 1st Workshop on Detection, Representation and Management of Concept Drift in Linked Open Data*, edited by Laura Hollink, Sándor Darányi, Albert Meroño Peñuela, and Efstratios Kontopoulos, 1799:26–33. CEUR Workshop Proceedings. Bologna, Italy: CEUR. http://ceur-ws.org/Vol-1799/#Drift-a-LOD2016_paper_4.

Rozenfeld, Slawek. 2001. 'Using The ISSN (International Serial Standard Number) as URN (Uniform Resource Names) within an ISSN-URN Namespace'. Request for Comments RFC 3044. Internet Engineering Task Force. https://doi.org/10.17487/RFC3044.

Santangelo, George. 2022. 'ICite Database Snapshots (NIH Open Citation Collection)', February. https://doi.org/10.35092/yhjc.c.4586573.v27.

Shotton, David. 2013. ‘Publishing: Open Citations’. *Nature* 502 (7471): 295–97. https://doi.org/10.1038/502295a.

Shotton, David. 2018a. ‘Funders Should Mandate Open Citations’. World View. Nature. 9 January 2018. https://www.nature.com/articles/d41586-018-00104-7.

Shotton, David. 2018b. ‘Citations as First-Class Data Entities: Introduction’. *OpenCitations Blog* (blog). 19 February 2018. https://opencitations.wordpress.com/2018/02/19/citations-as-first-class-data-entities-introduction/.

Shotton, David. 2021. ‘Coverage of Open Citation Data Approaches Parity with Web of Science and Scopus’. *OpenCitations Blog* (blog). 27 October 2021. https://opencitations.wordpress.com/2021/10/27/coverage-of-open-citation-data-approaches-parity-with-web-of-science-and-scopus/.

Van Noorden, Richard. 2014. ‘Google Scholar Pioneer on Search Engine’s Future’. *Nature*, November. https://doi.org/10.1038/nature.2014.16269