A Tool for Validating and Monitoring Bibliographic Data in Open Research Information Systems: the OpenCitations Collections

Silvio Peroni[1], Elia Rizzetto[1,*]

[1]Research Centre for Open Scholarly Metadata, Department of Classical Philology and Italian Studies, University of Bologna, Bologna, Italy

Abstract

This paper presents methodology and software for ensuring data quality in open scholarly bibliographic collections. Considering the case study of OpenCitations Meta and OpenCitations Index, storing bibliographic metadata and citations respectively, two tools are introduced: a data validator and a data monitor. The validator checks the syntactic and semantic correctness of bibliographic data before ingestion, providing both machine-readable reports and user-friendly feedback. The monitor tracks known data issues post-ingestion using SPARQL queries, ensuring ongoing data integrity. Designed with accessibility in mind, both tools facilitate automated workflows and user interaction.

Keywords

bibliographic collections, validation, data quality, OpenCitations

1. Introduction

Reproducible and responsible research assessment increasingly relies on — or, at least, pushes for relying on — collections of open research information. These datasets are fundamental to implementing open science principles contextualised in research assessment exercises, as devised in several European and international initiatives, including the Coalition for Advancing Research Assessment (CoARA)[1] and the Barcelona Declaration on Open Research Information (DORI)[2]. Recognising their strategic value, public and private academic publishing organisations have prioritised developing and expanding such open datasets.

In this context, OpenCitations [1] — a nonprofit infrastructure organisation that fosters unrestricted access to global citation data and bibliographic metadata — emerges as a pivotal actor. OpenCitations provides two main collections: OpenCitations Index [2], storing citations links between scholarly entities, and OpenCitations Meta [3], storing the basic bibliographic metadata (title, authors, year of publication, publication venue, publishers, identifiers) of the citing and cited entities involved in the citations available in the OpenCitations Index. These datasets are derived from diverse sources, whose data is reshaped into the OpenCitations Data Model (OCDM) [4] and ingested into OpenCitations' collections through an established ingestion workflow.

However, ingesting large volumes of data from heterogeneous sources introduces potential errors and inconsistencies. These errors fall into two broad categories:

- errors in primary sources, arising from human inaccuracies or software bugs in the originating systems;
- errors in the OpenCitations ingestion software, stemming from bugs in the conversion and ingestion processes.

IRCDL 2025: 21st Conference on Information and Research Science Connecting to Digital and Library Science, February 20–21, 2025, Udine, Italy

*Corresponding author.

silvio.peroni@unibo.it (S. Peroni); elia.rizzetto1@unibo.it (E. Rizzetto)

0000-0003-0530-4305 (S. Peroni); 0009-0003-7161-9310 (E. Rizzetto)

[1]https://coara.eu/

[2]https://barcelona-declaration.org/

While OpenCitations' ingestion workflow already addresses many of these issues, either by sanitising invalid data or discarding it where automatic correction is not feasible, challenges still need to be solved. A preliminary validation process is essential to identify and understand incorrect data, minimise information loss during ingestion, and ensure the accuracy of the final datasets. To meet these objectives, OpenCitations has developed a custom validation tool presented in this paper, and designed to provide the precision and granularity necessary for addressing the unique requirements of OCDM.

However, despite these validation and curation mechanisms, some errors can be introduced in further ingestions, necessitating continuous data quality monitoring even after ingestion. To this end, a tool for monitoring the quality of data has been implemented, with the objective of tracking the existence of known errors in the data and helping implement effective prevention and correction strategies.

The rest of this paper is structured as follows. Section 2 analyses the case study and the data features involved, then describes the methodology followed to develop viable solutions for validating bibliographic metadata and citation data and monitoring their quality. Section 3 goes over more technical details on the implementation of the software components. Section 4 discusses other related work about the quality assessment of RDF data. Finally, Section 5 illustrates the final remarks and suggests possible future developments.

2. Material and Methods

This section presents an overview of OpenCitations' ingestion workflow and describes the proposed methodology for pre-ingestion data validation and post-ingestion quality assurance.

2.1. The data and the current ingestion workflow

In the OpenCitations Index [2], citations are represented as first-class data entities, meaning that each citation is represented as an entity in its own right, representing a directed link between two other entities (*publication A cites publication B*) with its properties, including: citing entity, cited entity, citation creation date, and citation timespan (i.e. the difference in days between the date of publication of the citing entity and the date of publication of the cited entity). All the data in OpenCitations Index is collected from raw citation data openly provided by other external sources, and it is then published under a CC0 waiver. The current (as of 18 December 2024) sources are Crossref [5], DataCite [6], the National Institute of Health Open Citation Collection (NIH-OCC) [7], OpenAIRE [8], and the Japan Link Center (JaLC) [9]. The metadata of the publications involved as either citing or cited entities in the OpenCitations Index are stored in OpenCitations Meta [3]. For each publication, this collection provides details including their persistent identifiers (PIDs) for the publication (e.g. DOI), its title, the publication type (e.g. journal article, book, dataset, etc.), the publication date, the venue and its PIDs, the page interval, the issue and volume numbers, and the name and PIDs of the agents involved in the publication, i.e. authors, editors and publisher.

Gathering citation data and bibliographic metadata from diverse primary sources and unifying them into the OpenCitations Index and the OpenCitations Meta poses significant challenges since each source represents the data in its own way, and some information might overlap or differ across the sources. To overcome this challenge, a workflow has been developed [2] to reshape the gathered data according to the OpenCitations Data Model (OCDM) [4] and then ingest it into the collections. The OCDM is a data model built by reusing existing ontologies for describing information in the scholarly bibliographic domain and essentially consists, in the scope of data validation and error detection, of the fundamental set of rules defining the correct relationships and properties of all entities in OpenCitations Meta and OpenCitations Index.

The workflow currently implemented for making bibliographic metadata and citation data coming from external sources OCDM-compliant and ingesting it into OpenCitations' collections consists of three steps [2]:

1. *Source Preprocess.* This step reshapes the data by implementing a metadata crosswalk from

the diverse data models used by original sources to the OCDM, managing the differences in information content, structure and representation. A central operation in this phase is the normalisation and validation of external PIDs, such as DOIs for publications or ORCIDs for authors and editors. The output of the software dedicated to this step, the OpenCitations Data Sources Converter [10], are two OCDM-compliant tables which are used in the two following steps: one storing the bibliographic metadata for each publication involved in a citation (where each row represents a publication and columns store the values for supported metadata content), the other storing citations (where each row represents a citation and two columns store the PIDs of the citing and the cited publication, whose scheme depend on the original source, e.g. citations from Crossref are represented as DOI-to-DOI citations)

2. *Meta Process.* This step populates the OpenCitations Meta collection. Starting from the table produced in the first step, the bibliographic metadata to ingest is automatically curated by deduplicating records (i.e. table rows) that feature the same PID as another record and normalising and correcting their values. Records representing entities that had been registered in OpenCitations Meta in previous ingestions can be used to enrich the already available metadata for those entities or to merge them into a single entity (in the case the external PIDs appearing to pertain to a single entity in the record are instead linked to separate entities in OpenCitations Meta). Each entity in OpenCitations Meta is represented by the OpenCitations Meta Identifier (OMID), a PID that is minted and assigned to the entity in the moment of its generation: this is a crucial feature, as it allows the following step of the workflow to uniquely identify the publications linked by citations without relying on external identifiers.
   The output of this step, whose complete methodology and implementation are detailed thoroughly in [3], is the OpenCitations Meta dataset itself, stored both in a database and as dump files. Notably, the software responsible for the operations mentioned above also generates and stores in RDF files provenance information for each entity, keeping track of the agent that created, modified, merged or deleted it, the time of the action and the primary source providing the data.
3. *Index Process.* This step processes the citation tables from the Source Preprocess phase where each citation is represented as a link between two external PIDs (e.g., DOI-to-DOI, PMID-to-PMID). By making use of a mapping between these external identifiers and the OMID of the entity they have been associated with in the previous step (Meta Process), it converts these links into OMID-to-OMID citations, each of which is uniquely identified as a first-class entity by an Open Citation Identifier (OCI). Similarly to step 2, the process output consists of the OpenCitations Index dataset, with citation data stored in a database and as dump files, and provenance information saved in files only.

The workflow briefly described above is currently only applied to data from authoritative sources, such as Crossref or DataCite, which structure their data according to a defined data model. While this workflow is undoubtedly useful — particularly because it allows for the ingestion of a large volume of data with each execution — its application is effectively limited to data sources where implementing the metadata crosswalk from the source data model to the OCDM (as outlined in the Source Preprocess step) is feasible or advantageous. Since this process requires notable effort to manage the idiosyncratic complexities of each source, creating a custom data conversion system for each source may not be an applicable strategy, especially for sources that cannot provide certain conditions that facilitate this process (e.g. a defined data model, clear documentation of the data structure, etc.). Nonetheless, there are organisations and individuals who, despite lacking these characteristics, hold high-quality bibliographic data that is not yet easily accessible or reusable. Ingesting this data into collections like those of OpenCitations is crucial to making a large number of up-to-date citations and bibliographic metadata openly available.

As has been already pointed out by OpenCitations [11], an effective solution to broaden the number of open scholarly bibliographic data can be represented by crowdsourcing the data itself: users would be able to directly submit via dedicate service tables containing citations and metadata to be ingested into OpenCitations Index and OpenCitations Meta respectively. Users would need to submit tables that

Table 1
Two sample table cells of META-CSV, storing the surnames, names and identifiers of the authors of a bibliographic resource and the publication date of the bibliographic resource.

...	author	pub_date	...
...	Peroni, Silvio [orcid:0000-0003-0530-4305 viaf:309649450]; Shotton, David [orcid:0000-0051-5506-523X]	2023-03-13	...

are already OCDM-compliant and formatted to be natively processed by the relevant OpenCitations software (i.e. equivalent to the output of the current workflow's first step, Source Preprocess); in creating them, they should follow the guide provided in two reference documents [12, 13], in order for these tables to be interpreted correctly in the Meta Process and Index Process steps of the workflow. As the tabular documents obtained this way would not be built via a controlled internal process, validating them becomes imperative to ensure data quality. To fulfill this need, we have developed a custom validation tool described in the following subsection.

2.2. Pre-ingestion validation

The tabular format has been chosen for user submissions in that it is approachable even by scholars, researchers and professionals with little coding skills, yet the inherent complexity of the relationships and information expressable with the OCDM can fit into such a format only following precise rules, defined in [13, 12]. Following the nomenclature in these reference documents, we will henceforth refer to the table storing metadata as META-CSV and to the table storing citations as CITS-CSV.

In META-CSV, each row represents a bibliographic resource, i.e. a publication, and the eleven columns specify: the identifiers associated with the resource; the title; the surname, name, and identifiers of its authors and editors; the publication date; the venue (i.e. another bibliographic resource containing the represented document, e.g. the journal containing the article represented in the row); the volume of the venue containing the document; the issue of the venue containing the document; the page range; the type of publication; and the name and identifiers of the publisher.

In CITS-CSV, each row represents one citation, and the four columns store the values for the identifiers and the publication date of the citing and the cited bibliographic resource.

META-CSV and CITS-CSV tables have a layered structure that adds complexity to their validation. Beyond their tabular organisation of rows and columns, field values within each cell can consist of either single data units or collections of multiple data units separated by specific delimiters. These individual units, termed "items", represent the minimal "portion" used by the document to define a specific piece of information and, therefore, must be validated individually. For example, in CITS-CSV, the identifier fields for citing and cited resources may contain multiple items, while in META-CSV, fields for identifiers, authors, venues, publishers, and editors may similarly admit multiple items.

Adding to this complexity, in META-CSV, some fields contain items composed of smaller components, such as names and identifiers of entities (e.g., authors, editors, publishers, and venues). Each component requires distinct validation rules based on its type, leading to diverse validation requirements for the content of a single field. Table 1, Figure 1, and Figure 2 can be used to understand the abstract representation of the structure of the table.

Validation rules for the tables encompass both formatting/syntactic and content criteria. Syntactic rules are defined in the specifications to write well-formed documents [12, 13] and ensure proper data types, formats, and required fields. The other rules extend beyond syntax, addressing requirements such as the existence of referenced identifiers in relevant registries and the correctness of the relationships expressed in the table, also for those requirements that are not explicitly mentioned in the table specifications (e.g. a META-CSV row corresponding to a bibliographic resource to which a given type has been assigned may have only a certain set of values in the identifier field to be compliant with OCDM).

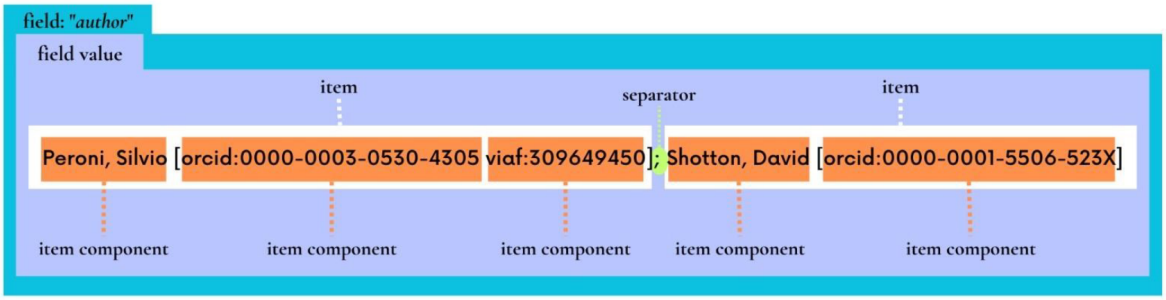


Figure 1: The abstract representation of the internal structure of the table cell containing the data for the author of a bibliographic resource (see Table 1). The cell contains two items, each of which corresponds to the entity of an author; each items has internal components of different kinds (the plain text of the surname and name, and the series of identifiers).

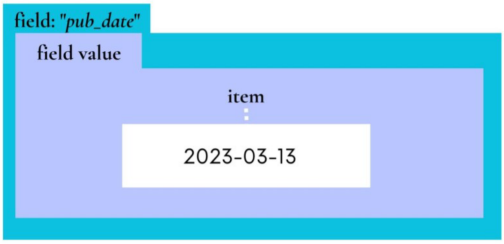


Figure 2: The abstract representation of the internal structure of the table cell containing the data for the publication date of a bibliography resource (see Table 1). The cell contains only one item, which corresponds to the value of the publication date. The publication date field always has a value containing a single item.

At the beginning of the validator designing phase, we first identified all the applicable validation rules for each of the two documents and grouped them into four different categories: rules related to the format and syntax of the document as prescribed by OpenCitations; rules based on the externally-defined syntax of PIDs (e.g. the valid structure of a DOI value); rules verifying the existence of an entity in the real world; and rules checking the relationships between the values. These categories have been used to structure the validation process into four levels, applied sequentially to the table elements:

1. *Wellformedness.* This step ensures the document complies with the syntactic rules defined in [12, 13] to generate well-formed tables, e.g. supported identifier schemes, correct date formats, etc. Errors at this level block further validation of affected items.
2. *ID Syntax.* All PID values are checked against syntax rules defined by their issuing organizations, ensuring formats such as the ones for DOI[3], ORCID[4], and PMID[5] are correctly applied.
3. *ID Existence.* The existence of mentioned entities in the real world is verified by using their associated identifiers as a proxy: PIDs are queried against official databases to confirm they are actually registered as such.

The implementation of the validation process has been guided by the following design principles:

1. *Maximum granularity.* Each document is validated by applying checks on its smallest parts (in most cases items, but if applicable also sub-parts of an item) to maximise granularity in the output and identify faulty table elements with high specificity.
2. *Maximum coverage at each execution.* At each execution of the process, the entire table is validated from start to end, i.e. without stopping the process if an error is found: all detectable errors are

[3]https://www.doi.org/
[4]https://orcid.org/
[5]https://pubmed.ncbi.nlm.nih.gov/

collected during the process and returned as a comprehensive collection in the output validation report. This process makes it easier for users to correct errors, since they are enabled to potentially address all issues in one correction cycle, avoiding repeated submissions.

3. *Non-redundancy*. A single item that has already failed a check is not validated against the rest of the rules, as it will need to be modified by the user. This principle only applies in cases where compliance with one rule is a prerequisite for compliance with the other rules for the same item. Otherwise, i.e. if the outcomes of two checks on the same item are mutually independent, both checks are executed straight away (i.e. before the user intervenes with any corrections).

The custom validator has been designed with the aim of providing a precise and information-rich feedback on the validity status of the data that is both suitable for programmatic use and human-readable. Machine-readability is essential for the primary objective of discarding invalid data before ingestion automatically, and for granting the possibility to use the validation output in other applications (e.g. for double-checking internally generated documents in the Source Preprocess phase of the ingestion workflow). Human-readability and user-friendliness are key to the fulfillment of the other fundamental objective of the validator: providing users with a tool to better understand how to create correct bibliodata tables.

The output is provided as a report listing all detected errors from a single execution of the validation process. Each error includes:

- *Position details*: the exact location of all the single pieces of data involved in that error (relative to the whole document) and whether the error regards a single item, multiple fields, or multiple rows.
- *Validation level*: The validation level where the error occurred.
- *Type of issue*: Whether the issue is a blocking "error" or a non-blocking "warning".
- *Unique error label*: A short label indicating the category to which each error instance belongs, which can be used by a machine to process the output.
- *User message*: Natural language explanation of the error and its potential causes.

Particular attention has been given to finding a feasible way to express the position of the error in the document, so that the specific data points involved in the error could be retrieved and processed automatically, while at the same time grant the user the possibility to exactly see the single parts of the document involved in the error. Nonetheless, the format specially designed to indicate error positions reflects the complexity of the internal structure of the table and can be cumbersome for humans to read. To solve this limitation, we went further in the direction of user-friendliness and paired the validator with a component entirely dedicated to the visualisation of the validation report in a graphical interface. This solution allows users to grasp the basic information about the errors more easily, visualising directly, on a new tabular representation of the input document, where they are located and their explanation.

2.3. Post-ingestion data quality assurance

Managing large collections of bibliographic and citation metadata from diverse sources requires a robust and systematic process to ensure data quality. Despite preventive measures such as validation during ingestion, errors can persist or emerge over time. To maintain data integrity, it is essential to monitor the correctness of ingested data regularly, implement correction strategies, and evaluate the success of these interventions.

A data monitoring tool has been developed for OpenCitations Meta and OpenCitations Index to address this need. This tool systematically searches for and tracks the presence of pre-identified and categorised errors in the data, providing actionable insights to guide corrections and assess the outcomes of previous improvements.

As a preliminary step, all known errors existing in OpenCitations Meta and OpenCitations Index have been collected and described via the use of a basic framework, which enables us to keep track,

for each error, of details like the assigned error label, actual examples, a description of the issue, the interested collection, and any strategy or programming solution to find actual data affected by the error.

From here, a subset of error types has been selected, including those reproducible by querying the databases of the two collections, which are accessible online via the SPARQL endpoint. This is because retrieving content by querying the databases for a given pattern is much faster than other solutions, like the programmatic analysis of the whole dump.

The monitoring process works by a simple logic: for each error, the data are tested by trying to retrieve from the associated collection any results that fall within the pattern representing it; if any result is retrieved, it means that there is wrong data (i.e. not compatible with OCDM or at any rate not expected) and the test for that particular error fails. The outcome of all the tests (i.e. whether it passed or not) is gathered during the process and returned in the final output of the monitor, which includes:

- general details about the execution of the process: the SPARQL endpoint URL of the accessed database, the queried collection (i.e. either OpenCitations Meta or OpenCitations Index), the date and time of the execution, the total running time, the path of the configuration file used to specify which errors have been tested.
- the details for each single error: label, natural language description, retrieving method (i.e. the exact SPARQL query), test result and details about the execution of that single test, including its individual running time and execution errors in case they were raised.

The monitoring tool has been designed with extendability, automation, and accessibility in mind. Its modular architecture ensures that new error tests can be added seamlessly, provided the errors can be represented through SPARQL queries. This flexibility allows the system to adapt continuously as new types of errors are discovered and defined.

To facilitate continuous oversight, the monitor can operate fully automatically, with periodic, scheduled executions. Currently, the process runs every Monday for both OpenCitations Meta and OpenCitations Index. Similarly to the validator, the monitor prioritizes user-friendliness by presenting its results in both machine-readable and human-readable formats. This dual representation supports diverse use cases, from automated integration into workflows to manual examination of findings. Notably, as part of the automated workflow, the human-readable results are used to update a publicly accessible web page, offering full transparency into the latest monitoring outcomes. The results can be viewed at: https://ocmonitor.opencitations.net/.

3. Implementation

This section presents technical details on the implementation of the methodology described above.

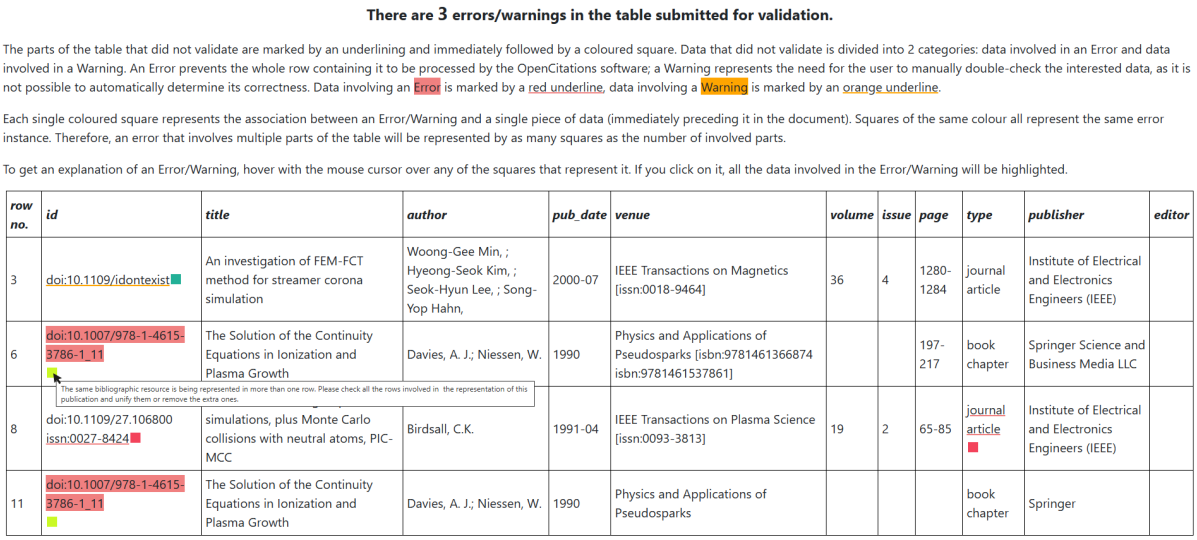
3.1. oc_validator

The validator tool described in Section 2 has been implemented in a Python software named `oc_validator`, available as a public repository[6] under an ISC license and as an installable library in PyPI[7].

The main process is managed by a dedicated class, named `Validator`, that takes as input the path to the table to validate (a CSV file that must be formatted as either META-CSV or CITS-CSV) and the path to the directory where the output files will be stored. As this interface only deals with one document at a time, another class was added to enable the simultaneous cross-validation of both metadata and citation data contained in two separate documents (also formatted as META-CSV and CITS-CSV respectively), which simply wraps in two instances of `Validator`, one for either document type.

As a first step of the process, the type of table is automatically determined, and a specific method managing the related operations is called accordingly. Internally, in fact, the processes for validating META-CSV and CITS-CSV are distinct, though their inner workings are similar.

[6] https://github.com/opencitations/oc_validator
[7] https://pypi.org/project/oc-validator/



There are 3 errors/warnings in the table submitted for validation.

The parts of the table that did not validate are marked by an underlining and immediately followed by a coloured square. Data that did not validate is divided into 2 categories: data involved in an Error and data involved in a Warning. An Error prevents the whole row containing it to be processed by the OpenCitations software; a Warning represents the need for the user to manually double-check the interested data, as it is not possible to automatically determine its correctness. Data involving an Error is marked by a red underline, data involving a Warning is marked by an orange underline.

Each single coloured square represents the association between an Error/Warning and a single piece of data (immediately preceding it in the document). Squares of the same colour all represent the same error instance. Therefore, an error that involves multiple parts of the table will be represented by as many squares as the number of involved parts.

To get an explanation of an Error/Warning, hover with the mouse cursor over any of the squares that represent it. If you click on it, all the data involved in the Error/Warning will be highlighted.

row no.	*id*	*title*	*author*	*pub_date*	*venue*	*volume*	*issue*	*page*	*type*	*publisher*	*editor*
3	doi:10.1109/idontexist	An investigation of FEM-FCT method for streamer corona simulation	Woong-Gee Min, ; Hyeong-Seok Kim, ; Seok-Hyun Lee, ; Song-Yop Hahn,	2000-07	IEEE Transactions on Magnetics [issn:0018-9464]	36	4	1280-1284	journal article	Institute of Electrical and Electronics Engineers (IEEE)	
6	doi:10.1007/978-1-4615-3786-1_11	The Solution of the Continuity Equations in Ionization and Plasma Growth	Davies, A. J.; Niessen, W.	1990	Physics and Applications of Pseudosparks [isbn:9781461366874 isbn:9781461537861]			197-217	book chapter	Springer Science and Business Media LLC	
8	doi:10.1109/27.106800 issn:0027-8424	simulations, plus Monte Carlo collisions with neutral atoms, PIC-MCC	Birdsall, C.K.	1991-04	IEEE Transactions on Plasma Science [issn:0093-3813]	19	2	65-85	journal article	Institute of Electrical and Electronics Engineers (IEEE)	
11	doi:10.1007/978-1-4615-3786-1_11	The Solution of the Continuity Equations in Ionization and Plasma Growth	Davies, A. J.; Niessen, W.	1990	Physics and Applications of Pseudosparks				book chapter	Springer	

Figure 3: A screenshot of an example HTML page for the visualisation of a META-CSV document validation report.

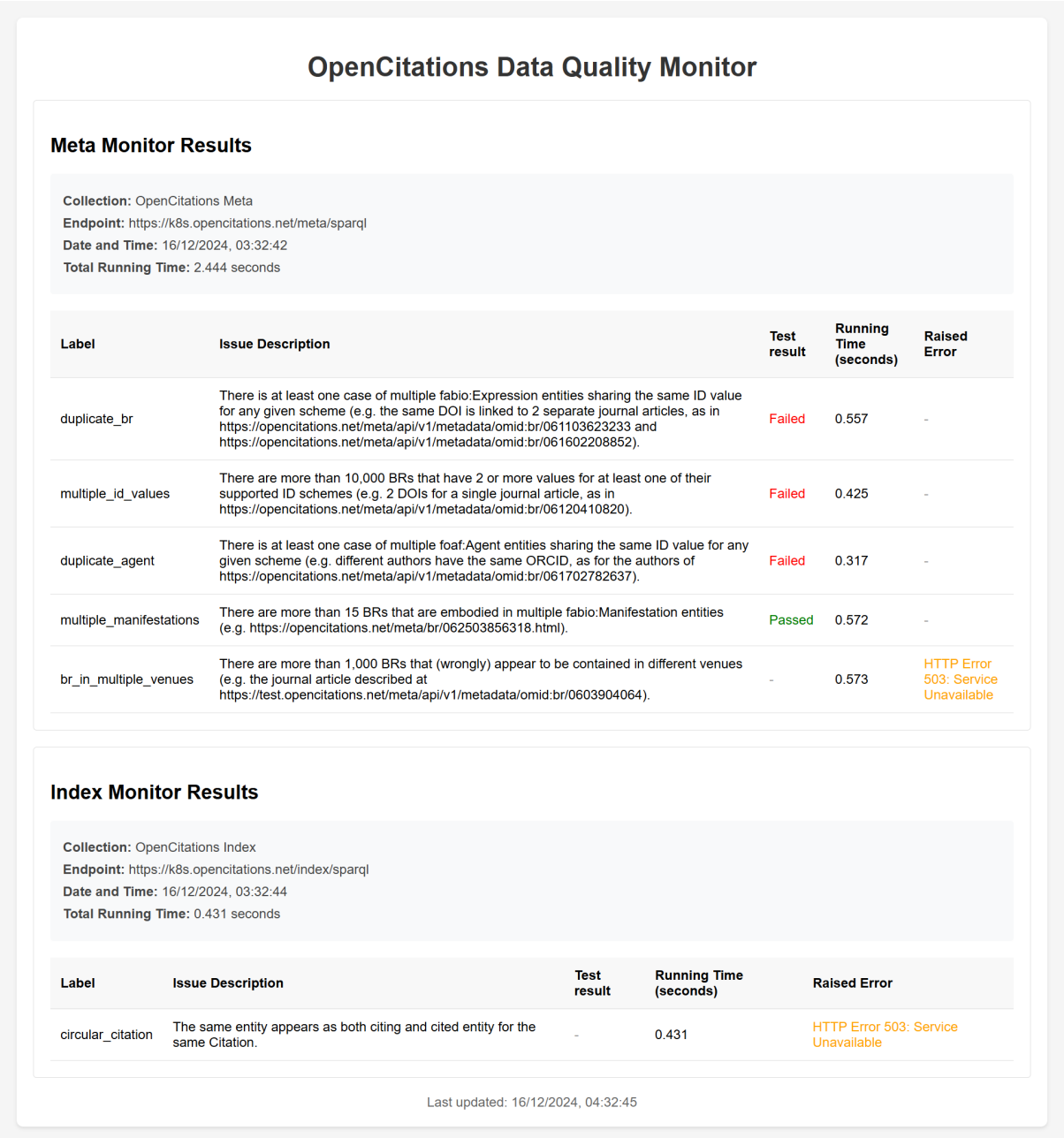
The validation process then iterates over all the table rows and columns (i.e. fields) and, depending on the field name, elaborates the internal content of the cell (which is initially interpreted as a single string) according to its abstract internal structure. The appropriate validation rules are sequentially checked by executing specific functions for each of the extracted items and their sub-components. Each of the four validation levels has its corresponding class, and each validation rule of the level is checked by a dedicated class method. In compliance with the design principle of non-redundancy mentioned in the Material and Methods section, if an element fails a check it is not tested for the other validation rules that might apply to it, unless the latter are independent of the first check's outcome.

Whenever a check fails, a dictionary object is generated representing the error and added to a list that will constitute the validation process output. An example of an error object is represented below in the form of a JSON object:

```
{
    "validation_level": "csv_wellformedness",
    "error_type": "error",
    "error_label": "duplicate_br",
    "valid": false,
    "message": "The same bibliographic resource is being represented
        in more than one row. Please check all the rows involved in
        the representation of this publication and unify them or
       remove the extra ones.",
    "position": {
        "located_in": "row",
        "table": {
            "2": {"id": [0, 1]},
            "3": {"id": [0, 1]}
        }
    }
}
```

When the end of the document is reached, and all the elements of the table have been validated, the list containing the error object is stored in a JSON file constituting the machine-readable output of the process and is converted into a TXT file summarising the latter, which consists of its human-readable version.

After the validation phase, the JSON output creates a graphical user interface as an HTML page to visualise the validation outcome better. In the HTML document, the integrated CSS styling and JavaScript code allow users to interact with the content. The rows of the original document that contain one or more errors are presented in an HTML table, where the location of the error is signalled by underlining exactly the wrong content associated with it and accompanying it with a square. By clicking on each square, the related faulty content is highlighted in all the involved locations, while hovering on it with the cursor shows the explanation of the error. An example the interface is provided as a static image in Figure 3.



OpenCitations Data Quality Monitor

Meta Monitor Results

Collection: OpenCitations Meta
Endpoint: https://k8s.opencitations.net/meta/sparql
Date and Time: 16/12/2024, 03:32:42
Total Running Time: 2.444 seconds

Label	Issue Description	Test result	Running Time (seconds)	Raised Error
duplicate_br	There is at least one case of multiple fabio:Expression entities sharing the same ID value for any given scheme (e.g. the same DOI is linked to 2 separate journal articles, as in https://opencitations.net/meta/api/v1/metadata/omid:br/061103623233 and https://opencitations.net/meta/api/v1/metadata/omid:br/061602208852).	Failed	0.557	-
multiple_id_values	There are more than 10,000 BRs that have 2 or more values for at least one of their supported ID schemes (e.g. 2 DOIs for a single journal article, as in https://opencitations.net/meta/api/v1/metadata/omid:br/06120410820).	Failed	0.425	-
duplicate_agent	There is at least one case of multiple foaf:Agent entities sharing the same ID value for any given scheme (e.g. different authors have the same ORCID, as for the authors of https://opencitations.net/meta/api/v1/metadata/omid:br/061702782637).	Failed	0.317	-
multiple_manifestations	There are more than 15 BRs that are embodied in multiple fabio:Manifestation entities (e.g. https://opencitations.net/meta/br/062503856318.html).	Passed	0.572	-
br_in_multiple_venues	There are more than 1,000 BRs that (wrongly) appear to be contained in different venues (e.g. the journal article described at https://test.opencitations.net/meta/api/v1/metadata/omid:br/0603904064).	-	0.573	HTTP Error 503: Service Unavailable

Index Monitor Results

Collection: OpenCitations Index
Endpoint: https://k8s.opencitations.net/index/sparql
Date and Time: 16/12/2024, 03:32:44
Total Running Time: 0.431 seconds

Label	Issue Description	Test result	Running Time (seconds)	Raised Error
circular_citation	The same entity appears as both citing and cited entity for the same Citation.	-	0.431	HTTP Error 503: Service Unavailable

Last updated: 16/12/2024, 04:32:45

Figure 4: A screenshot of an example HTML page showing the results of oc_monitor. The test results and the execution time values are for illustrative purposes only.

3.2. oc_monitor

The monitor tool has also been implemented in the Python software `oc_monitor`, available as a public repository[8] under an ISC license and as an installable library[9] in PyPI.

The central component in the monitor software is the JSON configuration file that is passed as argument of the two separate classes that actually manage the process: `MetaMonitor`, in charge of looking for errors in OpenCitations Meta, and `IndexMonitor`, looking for them in OpenCitations Index. The configuration file specifies the URL of the SPARQL endpoint to interrogate and a list of objects representing an error or an issue in the data. Each of these objects contains: a short label for the issue; a textual description explaining the nature of the problem and possibly providing examples of faulty data; a flag to indicate whether the specific issue should be verified at the execution of the monitoring process; and the actual test for the error, a SPARQL query defining the pattern that would catch results if the error was present in the collection.

`MetaMonitor` and `IndexMonitor` must each be passed the appropriate configuration file to work properly, but their inner working is similar: for each test in the configuration file, the SPARQL query is executed against the specified endpoint and a report for the test is generated in the form of a dictionary. This object contains the information specified in the configuration (label, description and query text) and a boolean indicating whether the test passed or not. Moreover, details concerning the execution are included: running time of the query[10] and, in case execution errors were raised, e.g. due to HTTP errors, the problem itself is reported too, providing also a useful insight on the status of the system infrastructure at the moment of the attempted connection to the server. An example of an object representing a test result is given below:


```
    {
    "label": "duplicate_br",
    "description": "There is at least one case of multiple fabio:
       Expression entities sharing the same ID value for any given
       scheme (e.g. the same DOI is linked to 2 separate journal
       articles, as in https://opencitations.net/meta/api/v1/
       metadata/omid:br/061103623233 and https://opencitations.net/
       meta/api/v1/metadata/omid:br/061602208852).",
    "query": "PREFIX datacite: <http://purl.org/spar/datacite/>\
       nPREFIX literal: <http://www.essepuntato.it/2010/06/
       literalreification/>\nPREFIX fabio: <http://purl.org/spar/
       fabio/>\n\nASK {\n    ?br1 datacite:hasIdentifier/literal:
       hasLiteralValue ?lit ;\n    a fabio:Expression .\n    ?br2
       datacite:hasIdentifier/literal:hasLiteralValue ?lit ;\n    a
       fabio:Expression .\n    FILTER(?br1 != ?br2)\n}",
    "run": {
        "got_result": true,
        "running_time": 2.4839844703674316,
        "error": null
    },
    "passed": false
}
```


The output of the monitoring phase consists of these objects, stored in a JSON file with some general details such as the exact date and time of the execution and the total running time.

[8] https://github.com/opencitations/oc_monitor

[9] https://pypi.org/project/oc-monitor/

[10] The execution of each test requires from less than a second to less than 5 minutes, depending on the type of test and factors such as other ongoing processes on the working server, simultaneous network traffic, etc. Especially considering the small number of tests, using the SPARQL endpoint is much faster than processing the whole dump, currently comprising more than 4,6 billion triples for OpenCitations Meta and more than 8,5 billion triples for OpenCitations Index.

As for the validator, the JSON output is used to create a more user-friendly visualisation of the monitoring outcome, in this case simply by restructuring the content into an HTML page.

Notably, OpenCitations has paired oc_monitor with an automatisation pipeline developed via GitHub Actions[11] and available in the software's repository: every Monday, the monitor is run against both collections, and the latest results are added for storage in the public repository and exposed on a public web page[12]. For an example of such HTML page see Figure 4.

4. Related Work

As concerns RDF data quality assessment, there have been several endeavours to tackle this problem [14], though the great majority of the associated tools (where they exist) either require a high degree of manual configuration or provide information of limited use. An interesting work in this field, whose approach resembles the one proposed by our paper, is presented in [15]. Here the authors present a methodology and a tool for assessing the quality of RDF data by following a test-driven approach inspired by software engineering. They propose the concept of Data Quality Test Patterns (DQTPs), which encapsulate common data quality issues into structured SPARQL query templates. These patterns are then used to instantiate actual test-cases (SPARQL queries) by binding specific values to the variables in the templates. Notably, test-cases can be automatically generated with Test Auto Generators (TAGs), which interrogate the RDF data to test and try to instantiate test-cases based on the OWL axioms and RDFS constructs defined in the ontologies used therein. With TAGs, it is possible to ensure that the data complies with simple schema constraints such as property domain, range and cardinality. For more complex tests or tests that are not derivable from the constraints formalised in the ontologies, the authors suggest to manually instantiate specific tests by re-using the DQTPs. The methodology presented in [15] also defines coverage metrics, which measure the adequacy of the test cases by assessing how well they capture different aspects of data quality (e.g. property domain coverage, class membership), represents the test-cases in RDF and associates a URI to each of them.

While this methodology proves effective for datasets with broad and heterogeneous structures —such as DBpedia, which was evaluated by the authors —its applicability to more controlled environments like OpenCitations requires careful consideration. The schemas included in the ontologies reused by the OpenCitations Data Model (OCDM) are well-suited for verifying general or simpler semantic relationships, but they are often too basic to support the automatic generation of meaningful tests tailored to the specific constraints and application context relevant to OpenCitations. In contrast to crowdsourced datasets like DBpedia, the data in OpenCitations is converted and ingested within a strictly controlled environment, ensuring that many fundamental constraints are already satisfied by design. As a result, tests generated purely from ontology schemas would, in many cases, verify relationships whose correctness is already guaranteed, adding little value to the quality assessment process.

Moreover, many of the properties and classes defined in the ontologies reused by OpenCitations are not currently utilized in its datasets. This means that an automatic test generation approach, as described by [15], would likely produce a significant number of non-applicable test cases. Furthermore, OpenCitations' current post-ingestion quality assessment relies on a small set of specific tests, carefully designed based on known issues that have been previously identified in the dataset. At this stage, implementing a more generalized pattern-based testing methodology would require substantial effort without offering clear advantages. Instantiating tests manually provides greater flexibility, allowing for fine-grained selection of the rules to be checked while at the same time avoiding the computational cost of executing unnecessary test cases: coherently with the aim of providing an insight on the data quality that is a good representation of its fitness for use, our approach focuses on ensuring that quality assessments can be performed frequently and that the results are presented in a way that is clear and easily interpretable.

[11] https://github.com/features/actions
[12] https://ocmonitor.opencitations.net/

Additionally, there are practical concerns regarding the long-term viability of adopting the methodology proposed by [15], as the software tool implementing their approach does not appear to be actively maintained.

Nevertheless, [15] remains relevant for large-scale RDF quality assessment, and some of its aspects could be valuable for OpenCitations in the future. The scalability of its approach makes it suitable for analyzing extensive datasets, though, as mentioned, in the case of OpenCitations the results might include information of limited relevance. The representation of test cases in RDF with structured metadata is another strength of their methodology, offering a formalized approach to documenting and managing quality assessments. Furthermore, relying on a library of patterns facilitates the creation of new test-cases in a systematic and reusable way: in case the number of pre-identified issues should grow, it might be an interesting avenue to explore also for OpenCitations Meta and OpenCitations Index.

5. Conclusion and Future Works

This work has presented a framework for ensuring the quality of bibliographic and citation data within the OpenCitations Meta and OpenCitations Index. By addressing pre-ingestion validation, we have built a system aimed at verifying the compliance of the data eligible for ingestion with the data model adopted by OpenCitations and with its existing ingestion workflow: the implemented validator ensures the syntactic and semantic correctness of documents storing data to ingest, identifying errors with granular precision and enabling both programmatic and user-friendly use. As concerns post-ingestion quality monitoring, we developed a tool that can be used to continuously evaluate the current data by verifying its status for known, previously identified issues.

A key feature of both the validation and the monitoring tools is their human-oriented design: the endeavour to provide both internal and external users with clear and accessible information led to pairing the machine-readable results of both tools with human-readable and user-friendly interfaces.

While these tools can facilitate data quality management, certain limitations remain. The validator is inherently tied to OpenCitations' specific table formats (META-CSV and CITS-CSV) and the rules of the OCDM, making it unsuitable for use with other data structures and models. Moreover, the monitor relies on SPARQL queries to detect errors, which limits its capacity to identify issues that manifest only in data dumps or API behaviours.

Future efforts will focus on integrating the validator into a fully automated data submission workflow, allowing seamless validation during the ingestion process. This objective is particularly relevant within the context of data crowdsourcing, as it would streamline operations that can be done automatically while at the same time valuing the role of human agents in ensuring data quality.

Acknowledgments

This project has been funded by the European Union's Horizon Europe framework programme under grant agreement No. 101095129 (GraspOS Project).

References

[1] S. Peroni, D. Shotton, OpenCitations, an infrastructure organization for open scholarship, Quantitative Science Studies 1 (2020) 428–444. URL: https://doi.org/10.1162/qss_a_00023. doi:10.1162/qss_a_00023.

[2] I. Heibi, A. Moretti, S. Peroni, M. Soricetti, The OpenCitations Index: description of a database providing open citation data, Scientometrics (2024). URL: https://link.springer.com/10.1007/s11192-024-05160-7. doi:10.1007/s11192-024-05160-7.

[3] A. Massari, F. Mariani, I. Heibi, S. Peroni, D. Shotton, OpenCitations Meta, Quantitative Science Studies (2024) 1–26. URL: https://doi.org/10.1162/qss_a_00292. doi:10.1162/qss_a_00292.

[4] M. Daquino, S. Peroni, D. Shotton, G. Colavizza, B. Ghavimi, A. Lauscher, P. Mayr, M. Romanello, P. Zumstein, The OpenCitations Data Model, in: J. Z. Pan, V. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne, L. Kagal (Eds.), The Semantic Web – ISWC 2020, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2020, pp. 447–463. doi:10.1007/978-3-030-62466-8_28.

[5] G. Hendricks, D. Tkaczyk, J. Lin, P. Feeney, Crossref: The sustainable source of community-owned scholarly metadata, Quantitative Science Studies 1 (2020) 414–427. URL: https://doi.org/10.1162/qss_a_00022. doi:10.1162/qss_a_00022.

[6] DataCite Metadata Working Group, DataCite Metadata Schema Documentation for the Publication and Citation of Research Data and Other Research Outputs v4.5 (2024). URL: https://datacite-metadata-schema.readthedocs.io/en/4.5/. doi:10.14454/G8E5-6293, publisher: DataCite Version Number: 4.5.

[7] B. I. Hutchins, K. L. Baker, M. T. Davis, M. A. Diwersy, E. Haque, R. M. Harriman, T. A. Hoppe, S. A. Leicht, P. Meyer, G. M. Santangelo, The NIH Open Citation Collection: A public access, broad coverage resource, PLOS Biology 17 (2019) e3000385. URL: https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.3000385. doi:10.1371/journal.pbio.3000385, publisher: Public Library of Science.

[8] C. Atzori, A. Bardi, P. Manghi, A. Mannocci, The OpenAIRE Workflows for Data Management, in: C. Grana, L. Baraldi (Eds.), Digital Libraries and Archives, volume 733, Springer International Publishing, Cham, 2017, pp. 95–107. URL: https://doi.org/10.1007/978-3-319-68130-6_8. doi:10.1007/978-3-319-68130-6_8, series Title: Communications in Computer and Information Science.

[9] T. Kato, E. Tsuchiya, S. Kubota, Y. Miyagawa, Japan Link Center (JaLC): link management and DOI assignment for Japanese electronic scholarly contents, Journal of Information Processing and Management 55 (2012) 42–46. doi:10.1241/johokanri.55.42.

[10] A. Moretti, A. Massari, E. Rizzetto, M. Soricetti, I. Heibi, oc_ds_converter, 2024. URL: https://doi.org/10.5281/zenodo.12911527.

[11] I. Heibi, S. Peroni, D. Shotton, Crowdsourcing open citations with CROCI – An analysis of the current status of open citations, and a proposal, 2019. URL: http://arxiv.org/abs/1902.02534. doi:10.48550/arXiv.1902.02534, arXiv:1902.02534 [cs].

[12] A. Massari, How to produce well-formed CSV files for OpenCitations, 2022. URL: https://zenodo.org/record/6597141. doi:10.5281/zenodo.6597141, publisher: Zenodo.

[13] A. Massari, I. Heibi, How to structure citations data and bibliographic metadata in the OpenCitations accepted format, 2022. URL: http://arxiv.org/abs/2206.03971. doi:10.48550/arXiv.2206.03971, arXiv:2206.03971 [cs].

[14] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, S. Auer, Quality assessment for Linked Data: A Survey: A systematic literature review and conceptual framework, Semantic Web 7 (2015) 63–93. doi:10.3233/SW-150175.

[15] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, A. Zaveri, Test-driven evaluation of linked data quality, in: Proceedings of the 23rd International Conference on World Wide Web, ACM, Seoul Korea, 2014, pp. 747–758. doi:10.1145/2566486.2568002.