2BFAIR: Framework for Automated FAIRness Assessment

Leonardo G. Azevedo[1,*], Eduardo Caroli[2,†], Beatriz S. Corrêa[1] and Viviane T. da Silva[1]

[1]IBM Research, Av. República do Chile, 330, 20031-170, Rio de Janeiro, RJ, Brazil
[2]PUC-Rio, Rua Marquês de São Vicente, 225, 22451-900, Rio de Janeiro, RJ - Brazil

Abstract

The FAIR Principles intend to act as guidelines to enhance the Findability, Accessibility, Interoperability, and Reusability of digital objects. Identifying how close a digital object is to abiding by the FAIR principles' characteristics (i.e., compute its FAIRness) is a challenge tackled by several automated tools. However, no such tools fully support the main requirements for automated assessment, primarily the customization of the FAIRness evaluation according to community needs. This work presents 2BFAIR, a framework for automated FAIRness assessment. As a framework, 2BFAIR provides points of flexibility that its users can customize to fit a community's specific needs of FAIRness evaluation. On the other hand, 2BFAIR encapsulates complex logic common to a family of related issues required by any FAIRness evaluation into pieces of code that the user does not have to change to create a tool based on 2BFAIR. We provide 2BFAIR with a default implementation, i.e., a tool implemented using the 2BFAIR framework. We analyzed this tool against the tools of the state-of-the-practice to demonstrate the usefulness of 2BFAIR. 2BFAIR supports 87% of the requirements that automated tools for FAIRness assessment should meet while other tools reach at most 74%. It makes 2BFAIR a good choice for implementing tools tailored to community needs.

Keywords

FAIR Principles, FAIRness assessment, FAIRness evaluation, FAIRness, Automated tools for FAIRness assessment

1. Introduction

Enhancing knowledge discovery for human and computational agents is a challenge for data-intensive sciences, involving accessing, integrating, and analyzing task-appropriate data [1]. Wilkinson *et al.* [1] proposed the FAIR principles, a set of 15 recommendations for improving the Findability, Accessibility, Interoperability, and Reusability of digital resources [2]. The FAIR principles are supposed to be domain-independent, aim to facilitate the reuse of data by both humans and machines [3], and are related to several Semantic Web standards [4, 5]. Research involving digital objects benefits from applying the FAIR principles to ensure goals like transparency, reproducibility, and reusability.

Several mechanisms aim to evaluate a digital object's so-called 'FAIRness level' [6]. FAIRness corresponds to a value (*e.g.*, a percentage) indicating how close a digital object is to abiding by the FAIR principles. Such mechanisms include methods, processes, data maturity models, questionnaires, and (semi-)automated tools. Manual mechanisms are essential to improve overall understanding and appreciation of the research life cycle [7]; nevertheless, assessing FAIRness with them is time-consuming, requires experience, carries difficulties when inspection is needed, and does not scale when considering several digital objects [8, 9]. An automated tool is more appropriate to handle those issues.

This work presents 2BFAIR, a framework for automated FAIRness assessment aiming to support full customization of the evaluation according to community characteristics. Framework development promotes the reuse of design and source code through its ability to support the generation of applications

2nd International Workshop on Natural Scientific Language Processing and Research Knowledge Graphs (NSLP 2025), co-located with ESWC 2025, June 01–02, 2025, Portorož, Slovenia

*Corresponding author.
†Work done while at IBM Research.
lga@br.ibm.com (L. G. Azevedo); ecaroli@aluno.puc-rio.br (E. Caroli); beatriz.s.correa@ibm.com (B. S. Corrêa); vivianet@ibm.com (V. T. d. Silva)
https://ibm.com/leonardo (L. G. Azevedo)
0000-0002-2109-1285 (L. G. Azevedo); 0009-0008-5127-4358 (E. Caroli); 0009-0005-4811-3930 (B. S. Corrêa); 0000-0003-4669-7299 (V. T. d. Silva)

directly related to a specific domain (*i.e.*, a family of related problems) by reusing the framework's code [10]. In our work, we are proposing 2BFAIR framework to tackle the challenge of customizability of FAIRness assessment. 2BFAIR provides points of flexibility (*i.e.*, hot spots) that its users can customize to fit a community's specific needs of FAIRness evaluation. On the other hand, 2BFAIR encapsulates complex logic common to a family of related issues required by any FAIRness evaluation into pieces of code that the user does not have to change (*i.e.*, frozen spots) to create a tool based on 2BFAIR [11].

We provide 2BFAIR with a default implementation, *i.e.*, the 2BFAIR code is available along with a FAIRness evaluation tool developed using the 2BFAIR framework. When users clone the 2BFAIR GitHub repository, they get the framework and the default implementation code base. This implementation consists of a set of instantiated evaluators, a structured configuration, and an implementation of a (meta)data collector. It aims to be as broad as possible so that users can customize community-specific details or use it as an example to create their tool using the framework.

Examples of 2BFAIR users are software developers and end users. Developers may use the framework to create FAIRness evaluation tools. End users can use the default implementation to perform FAIRness assessment of their digital objects. Although developers have plenty of customization options by working directly on the code, end users can still customize the assessment by adjusting configuration parameters which do not require much technical skills.

We analyze this default implementation of 2BFAIR against the tools of the state-of-the-practice based on a set of requirements elicited from the literature. Our goal is not to prove it is much better than existing tools, although it slightly outperforms them. We demonstrate 2BFAIR capabilities and coverage of the requirements. The results show that 2BFAIR is a good choice for executing FAIRness assessments customized to user needs and as the base to creating new tools using the its framework.

The remainder of this work is divided as follows. Section 2 presents the related work. Section 3 presents the 2BFAIR framework. Section 4 presents the analysis of 2BFAIR, and compares it against state-of-the practice tools. Finally, Section 5 presents the conclusion and proposals for future work.

2. Related Work

There are several automated tools for FAIRness evaluation. Automated tools automatically inspect the digital object's metadata and data representations to compute its FAIRness level. Examples of inspections executed by some tools are: assessing if the digital object identifier is globally unique by checking if the value of the identifier matches known GUID (Globally Unique Identifier) schemas like URLs, IRI, and DOI; evaluation if the identifier is persistent by checking if the identifier matches persistence schema (*e.g.*, PURL, W3ID, DOI, and ARK); assessing the richness of metadata by searching for metadata terms like creator, title, publication date, publisher, summary, and keywords.

In a previous work [12], we conducted a systematic literature review (SLR) to discover automated tools for FAIRness assessment in the literature. A SLR is a research method with steps to organize the review methodically [13]. We searched the tools at three digital libraries (Scopus[1], IEEE Digital Library[2], and ACM Digital Library[3]). As a result, we found the automated tools: F-UJI [14, 15], FAIR Evaluator [7], and FAIR-Checker [9].

In this work, we expand the analyzed tools by including the ones listed by FAIRassist.org. FAIRassist.org [16] is a catalog that collects and describes resources aimed at helping developers and stakeholders make their digital objects FAIR. FAIR developers manually register their tools in FAIRassist.org. Afterward, a FAIRassist.org curator reviews the registration and adds it to FAIRassist if accepted. So, besides the automated tools we found in our literature review, FAIRassist.org lists[4]: FAIR Enough [17], FAIR EVA (Evaluator, Validator & Advisor) [18, 8], O'FAIRe [6, 19], HowFAIRis [20], FOOPS! [21, 22], FAIROs [23], CLARIN Metadata Curation Dashboard [24] [25], OpenAIRE Validator [26, 27], PresQT

[1]https://www.scopus.com
[2]https://ieeexplore.ieee.org
[3]https://dl.acm.org/dl.cfm
[4]Tools listed on March 17th, 2025.

(Preservation Quality Tool) [28, 29], and SciScore [30, 31].

Since the tools have different characteristics and goals, we define inclusion and exclusion criteria to analyze similar tools.

As an inclusion criterion, we consider only the tools that execute the following activities: (i) Receive a data digital object identifier (*e.g.*, URL or DOI[5]); (ii) Resolve this identifier to an object (like a landing page); (iii) Evaluate its FAIRness; and, (iv) Return results, *e.g.*, including FAIRness levels, metrics and test execution descriptions, and improvement recommendations. This general process fits different use cases, and the user only passes the digital object identifier to execute the assessment.

As exclusion criteria, we do not consider tools that meet at least one of the following characteristics: (a) Do not provide free access to run; (b) Do not target data digital objects; (c) Use other tools to perform the FAIRness assessment that we consider in our analysis or are discarded by any exclusion criteria.

Considering these criteria, we do not analyze the following tools. (a) OpenAIR Validator is not free to use [27]. (b) HowFAIRis targets FAIR software. It evaluates software code available in GitHub[6] or GitLab[7] [20]. It does not evaluate data digital objects. (c) SciScore evaluates scientific articles concerning transparency and reproducibility [30]. It does not evaluate the FAIRness of data digital objects. (d) PresQT uses FAIR Evaluator to perform automated assessment [32]. FAIR Evaluator is already considered in our analysis. (e) O'FAIRe evaluates ontologies available in the AgroPortal. It considers the ontology representation and the portal characteristics to perform the assessment [19]. The service is tied to the portal and cannot be used independently, *i.e.*, the tool cannot be used considering the process defined as our inclusion criteria. (f) FOOPS! targets ontology. It is a Web service that receives as input an OWL ontology or SKOS thesauri and returns their level of FAIRness [22]. It does not evaluate data digital objects. (g) FAIROs uses F-UJI and FOOPS! to perform the assessment. F-UJI is already considered in our work, and FOOPS! is discarded.

The application of inclusion and exclusion criteria results in the following tools: F-UJI, FAIR Evaluator, FAIR-Checker, FAIR Enough, and FAIR EVA. We analyze these tools considering the requirements we elicited in the SLR we conducted in a previous work whose goal was to [12]. The requirements were elicited by reading the papers resulting from the SLR and the papers describing the tools. We created a requirement for any mention of a desirable feature for an automated tool. The purpose of the requirements was to guide the appraisal and development of tools that effectively and automatically evaluate FAIRness. In the following list, we present the requirements, highlighting in bold the words we use to reference the them and the papers from which the requirements were identified.

- R1: The tool should be fully **automated** [33].
- R2: The tool should compute digital object **FAIRness grade** [4, 34, 35, 36].
- R3: The tool should be **autonomous**, *i.e.*, it should work **independently** from a specific domain, digital objects, or framework [36].
- R4: The tool should present **which principles** are evaluated [34].
- R5: The tool should specify the **types of digital object** it assesses (*e.g.*, data objects, data repositories, workflows, software) [7, 15].
- R6: The tool should be exposed as **APIs** (*e.g.*, as RESTful services [37]) [33].
- R7: The tool should be available on the internet as a **Web application** [33, 35].
- R8: The tool should indicate its usage **license** [34].
- R9: The tool should state its **development stage** [34].

[5]https://www.doi.org/
[6]https://github.com
[7]https://gitlab.com

- R10: The tool should be **customizable** according to the type of digital object and community [4, 35].
- R11: The tool should allow the user to supply **authentication** credentials [35].
- R12: The tool should provide a visual representation (*e.g.*, a **badge** of the FAIR assessment results [9, 15].
- R13: The tool should present the **data lifecycle** phases it supports [15, 38].
- R14: The tool should rely on **FAIR-enabling services** (*e.g.*, FAIRsharing.org, identifiers.org) to perform the assessment [7, 15].
- R15: The tool should offer **guidance** on how it is used (*e.g.*, providing user manual, help, publications, or explanatory tips) [34].
- R16: The tool should require **little expertise** to use [34].
- R17: The tool should export **machine-actionable** results (*e.g.*, JSON or RDF) [33, 35].
- R18: The tool should disclose its **rating system** [36].
- R19: The tool should be informative, *i.e.*, it should **teach** the user about the FAIR principles [36].
- R20: The tool should give **recommendations** on how to improve the FAIRness of the evaluated resource [34, 36].
- R21: The tool should export results readable in **natural language** [36].
- R22: The tool should store evaluation results in a **searchable** resource [35].
- R23: The tool should support **versioning** of FAIRness assessments [7].

In that previous work [12], we analyzed F-UJI, FAIR Evaluator, FAIR Checker, and FAIR Enough according to these requirements by reading their documentation, papers, and available code. In this work, we add FAIR EVA and 2BFAIR as well as evaluate all the tools by executing them in practice. The new results are presented in Table 1. We do not detail the evaluation due to the lack of space, although we summarize it. We compare the tools against 2BFAIR in Section 4.

F-UJI [15] automatically evaluates the FAIRness of research data objects according to FAIRsFAIR metrics[8]. The tool's primary entities are Principles, Metrics, and Tests. Each FAIRness test is executed using a pass-or-fail approach and returns a score and a maturity level to represent an overview of the digital object's fitness to each FAIR principle. F-UJI is available as a RESTful Web service and has a Web Client that renders a FAIRness badge. The service[9] and the Web Client[10] source codes are available on GitHub under the MIT License. The tool is only customizable by implementing new tests or deactivating some of them by altering the source code. The evaluation results are returned in JSON with scores, practical tests, inputs and outputs, and the evaluation context for each metric.

FAIR Evaluator [7] is a framework not tied to a specific domain and can be adapted to fit community needs, *i.e.*, the stakeholders may participate in the creation of community-specific Maturity Indicators, develop their own compliance tests, and define which tests to use in an evaluation. The FAIR Evaluator is primarily designed for mechanized interaction through its RESTful Web Service API and provides a demonstrative user interface [40] for form-based access. The client-side interface is implemented in JavaScript using the AngularJS framework. The backend service is available as a Ruby on Rails

[8]The version V0.5 of the metrics are available at Zenodo [39].
[9]https://github.com/pangaea-data-publisher/fuji
[10]https://github.com/MaastrichtU-IDS/fairificator

Table 1
Tools' appraisal considering the elicited requirements, using symbols to represent entire (✓ = Yes), partial (◗ = Partially) or lack of (✗ = No) fulfillment.

Req.	Req. Keyword	2BFAIR	F-UJI	FAIR Evaluator	FAIR Enough	FAIR Checker	FAIR EVA
R1	Automated	✓	✓	✓	✓	✓	✓
R2	Score	✓	✓	◗	✓	✓	✓
R3	Independent	✓	✓	✓	✓	✓	✓
R4	Refer principles	✓	✓	✓	✓	✓	✓
R5	Types of digital object	✓	✓	✓	✓	✓	✓
R6	RESTful services	✓	✓	✓	✓	✓	✓
R7	Web application	✓	✓	✓	✓	✓	◗
R8	License	✓	✓	✓	✓	✓	✓
R9	Development stage	✓	◗	✗	✗	◗	✓
R10	Customizable	✓	◗	◗	◗	◗	◗
R11	Allow authentication	✗	✓	✗	✗	✗	✗
R12	Badge	✓	✓	✗	✗	✓	✓
R13	Data lifecycle	✓	✗	✗	✗	✗	✗
R14	FAIR-enabling services	✓	✓	✓	✓	◗	✓
R15	Guidance	✓	✓	✓	✓	✓	◗
R16	Little expertise	✓	✓	✓	✓	✓	✓
R17	Machine-actionable	✓	✓	✓	✓	✗	✓
R18	Rating system	✓	✓	✓	✓	✓	✓
R19	Teach	✓	◗	✓	◗	◗	✓
R20	Recommendations	✓	✗	✗	✗	✓	◗
R21	Natural language	✓	◗	◗	◗	✓	✓
R22	Searchable	✗	✗	✗	✗	✗	✗
R23	Versioning	✗	✗	✗	✗	✗	✗

application. The codes of the framework[11] and the front-end[12] are available on GitHub under the MIT license. The FAIR Evaluator has 15 defined Maturity Indicators, evaluated by 22 Compliance Tests, that the user can group in a collection. This feature lets the user indicate which metrics should be considered for an assessment. For deeper customization, however, the user should have software development skills to create new tests or change existing ones by refactoring the source files.

FAIR Enough [17] implementation is based on FAIR Evaluator and F-UJI tools. It is available as a RESTful Web service implemented in Python using FastAPI[13] and provides a frontend Web interface implemented using React[14]. The code is available in GitHub[15] under MIT license. The tool does not present a visual representation summarizing the evaluation results. The tool and the tests are customizable but require technical skills. Test implementation uses FAIRsharing services. There is little documentation available on GitHub. The Web Client is easy to use in the same way as the FAIR Evaluator. However, Web service development skills are required to use the RESTful Web API.

FAIR-Checker [9] can evaluate any digital object as long as they are described by metadata available on a landing page. The tool execution process starts with the user supplying a URL as input. Then, it extracts semantic annotations from the Web page to create the first version of a Knowledge Graph (KG)[16]. Public KGs are queried using SPARQL[17] during the FAIRness evaluation against metrics tailored

[11] https://github.com/FAIRMetrics/Metrics
[12] https://github.com/FAIRsharing/FAIR-Evaluator-FrontEnd
[13] https://fastapi.tiangolo.com/
[14] https://react.dev/
[15] https://github.com/MaastrichtU-IDS/fair-enough
[16] "A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge" [41].
[17] https://www.w3.org/TR/sparql11-query/

to the evaluation of computation ontologies according to the FAIR principles. FAIR-Checker is a Web application developed in Python based on the Flask Web Framework and it provides a RESTful API. The tool is available on GitHub[18] under the MIT license and can be used on the Web [42]. The tool customization requires software development skills.

FAIR EVA [8] is available on GitHub[19] under the Apache 2.0 license. It can be deployed as a RESTful Web service API or as a Web client, neither of which is readily available, as the user must obtain the code from the project's repository and deploy both the RESTful Web service and the Web client. The tool proposes to achieve flexibility via a plugin architecture to support customization according to particular technical features of data repositories. This customization requires adaptations of the source code. Another possibility is to customize by altering configuration files specific to each plugin; however, this modality offers limited possibilities. The evaluation results contain explanations of what is evaluated by each indicator, which aids the user in learning about the FAIR concepts related to the metric. They detail the evaluation's technical implementation and give technical feedback regarding the evaluation result. In some cases, the evaluation also returns recommendations (listed as "tips") on how to improve the evaluated object's score on the particular metric being assessed.

Tools analysis: Considering values of 1, 0.5, and 0 to complete (✓), partial (◗), and no (✗) fulfillment, respectively, to present the coverage of requirements, the tools have the percentage: F-UJI (74%), FAIR EVA (72%), FAIR-Checker (70%), FAIR-Evaluator (63%), and FAIR Enough (63%). Customizability is a big issue because a user should have high software development skills to customize them.

Since no tool meets all requirements, standing out as state-of-the-art, and customizability is not fulfilled by none of them, we decided to develop the 2BFAIR framework, inspired by F-UJI, to fill the gaps. We present 2BFAIR in Section 3 and analyze it against the other tools in Section 4.

3. 2BFAIR

2BFAIR is developed as an application framework implemented in Python. As a framework [11], it consists of ready-to-use (*frozen spots*) and semi-finished building blocks (*hot spots*). The overall architecture is predefined, and to produce specific applications, the user has to adjust the hot spot building blocks to particular needs by overriding some methods in subclasses. Its goal is to facilitate data research stakeholders with varying software development expertise in implementing automated FAIRness evaluators that suit their community's needs.

We developed 2BFAIR considering the requirements presented in Section 2, which were elicited in a previous work [12]. We chose this because the requirements were defined through an SLR to guide the appraisal and development of tools that effectively and automatically evaluate FAIRness. They were used to evaluate tools found in the literature, and this evaluation identified several gaps that should be filled.

The 2BFAIR core components are the following.

- `Digital Object Information Collector` obtains the information used to evaluate the digital object's level of FAIRness from a provided digital object identifier. The retrieved information includes, *e.g.*, the digital object's landing page with its data and metadata.

- `Core Evaluator` performs individual FAIRness tests using the collected information and computes numeric scores for the metrics that represent each FAIR principle.

- `FAIRness Result Aggregator` formats and groups individual test results into coherent sets, *e.g.*, results of metrics, principles, and dimensions[20]. It enriches results with information that makes the evaluation more transparent and clarifies the used FAIR concepts.

[18] https://github.com/IFB-ElixirFr/FAIR-checker
[19] https://github.com/EOSC-synergy/FAIR_eva
[20] We define F, A, I, and R as FAIR dimensions.

Typically, the implementation of an automated FAIRness evaluator entails developing such components from the ground up. 2BFAIR aims to reduce the complexity and cost of this task by implementing them as the framework modules. The evaluation is performed according to a configuration defined for each test, reflecting the community's preferences. The evaluation comprises Dimensions (F, A, I, or R), Principles (F1, F2, A1, etc.), Metrics, and Tests.

A Metric defines a rule to evaluate a digital object according to a principle. Examples of such rules are the metrics defined by FAIRsFAIR [39] or the indicators of the RDA FAIR Data Maturity Model [43]. A Metric includes a set of Tests. A Test corresponds to the algorithm that evaluates the metric's rule. The evaluation of the test may result in `pass`, `fail`, or `not executed`. For example, for Principle F1 ("(meta)data are assigned a globally unique and persistent identifier"), we may have the metric "A globally unique identifier should be assigned to the data." An example of a test for this metric would be the algorithm to check if the value of the identifier follows the syntax of a GUID (Globally Unique Identifier), *i.e.*, an identifier guaranteed to uniquely identify a particular Resource, irrespective of the context, like a URL, IRI, or DOI.

The `Core Evaluator` module performs individual tests to attribute numeric scores to the metrics of each FAIRness principle. Each test must be tailored to the targeted research community, *e.g.*, the standards against which attributes of the Digital Object must be compared, such as identifier types, vocabularies, metadata schemas, and the assessment process itself. The framework was designed with the evaluation of FAIRness principles as a hot spot structured as the `PrincipleEvaluator` abstract class (Figure 1).

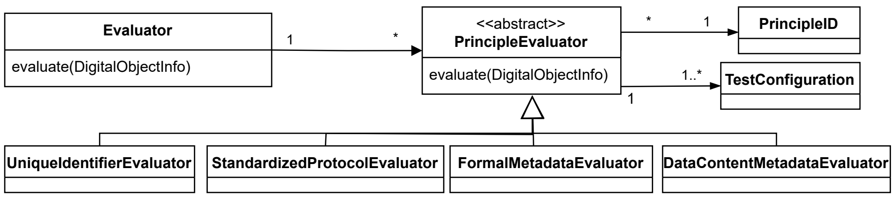

Figure 1: Class diagram of the `PrincipleEvaluator` hierarchy.

To tailor the evaluation, the user must define a class inheriting from the `PrincipleEvaluator`, decorate[21] the class with `@principle_evaluator`, set the assessed `PrincipleID`, specify the configuration of the FAIRness tests the class executes, and implement the `evaluate` abstract method. This method receives a `DigitalObject` and returns a list of `TestResult`. The decorator `@principle_evaluator` registers each `PrincipleEvaluator` in the `EvaluatorRegistry` and allows them to be referenced by the `evaluate` method of the `Evaluator`.

The process of a FAIRness evaluation is presented in Figure 2. The evaluation starts when the service client calls the method `evaluate` of the class `Evaluator` passing the `DigitalObject` as a parameter. The `Evaluator` calls `EvaluatorRegistry.get_evaluators()` to retrieve all the `PrincipleEvaluators` that are registered. Then, it calls all `PrincipleEvaluator.evaluate()`, which returns a list of `TestResult` corresponding to the FAIRness tests it executes. Finally, the `Evaluator` creates a FAIRness evaluation response to return to the service client.

2BFAIR offers a set of utilities like the `fairness_test` decorator and constructors for the `TestResult`. The `fairness_test` decorator aims to reduce boilerplate code by allowing the register of the function that evaluates a FAIRness test. It also controls which tests should be executed and skipped via the configuration. The constructors allow to generate results from the configuration received by the method. FAIRness tests are implemented as functions. Each test takes a `DigitalObject` instance as a parameter and returns an instance of the `TestResult` class. A `PrincipleEvaluator` executes all the tests corresponding to the metrics to evaluate the FAIRness of a digital object according to the guidelines of a FAIR Principle.

[21] A decorator allows to modify or extend the behavior of functions or methods, without changing their actual code (https://book.pythontips.com/en/latest/decorators.html).

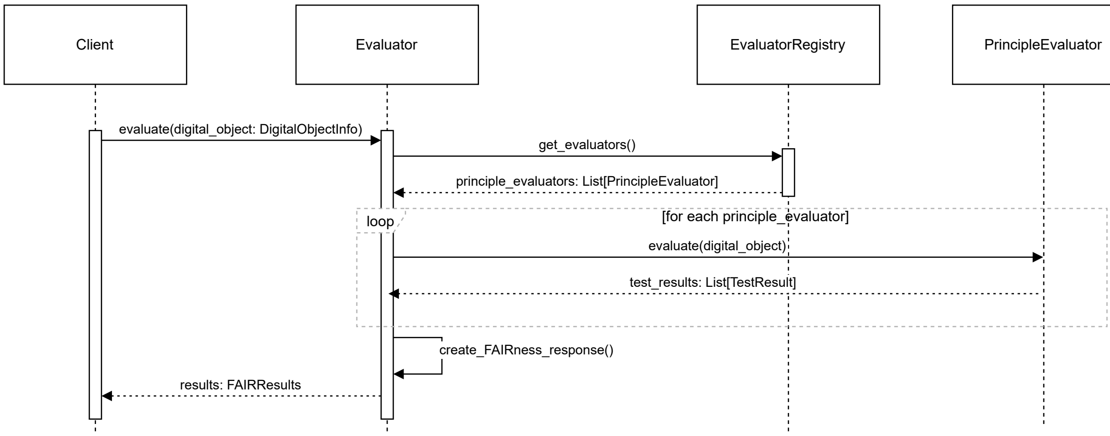

Figure 2: Sequence diagram of the FAIRness evaluation.

The configuration classes stand for FAIR Dimensions, Principles, Metrics, and Tests, depicted in Figure 3 and described as follows.

- `DimensionConfiguration` and `PrincipleConfiguration`: The user may add or remove FAIR dimensions or principles to be evaluated.
- `MetricConfiguration`: The metrics configuration are:
  - `priority`: The user may define priorities to be considered for each metric. The default is `Essential`, `Important`, and `Useful` [43]. An essential metric is crucial to achieve FAIRness under most circumstances. A metric is important when it might not be of the utmost importance under specific circumstances. A useful metric corresponds to a nice-to-have aspect, which is not indispensable.
  - `score_mechanism`: A metric's score can be computed as `Alternative` and `Cumulative`. In case a metric is evaluated in alternative fashion, even if multiple FAIRness tests pass, only one of the tests is exposed as having passed, the one with the highest score and maturity. In the cumulative computation, the metric's score corresponds to the sum of the scores of tests that passed.
  - `alternate_test_behavior`: It specifies the status and score of tests that do not pass. It may be set to either `skip` or `fail`. In both cases, the `TestResult`'s score is set to 0, and their status is set as skipped or failed.
- `TestConfiguration`: The test configuration attributes are:
  - `descriptions_for_passed, failed, missing_information`: FAIRness tests can pass, fail, or not be executed, and each attribute provides a detailed description of this status. Missing information is used when a test is not executed due to a lack of necessary information. These messages are exposed in the evaluation result along with the `result_description`.
  - `agnostic_requirements`: It defines characteristics to be tested. An example would be a list of citation elements (like title, creator, and publication date) that should be found in the metadata when executing the test that evaluates if *the metadata includes citation attributes*.
  - `community_requirements`: It defines characteristics to be tested that are specific to a community. For example, for the Chemistry community, the SMILES[22] element should be present as a core element of a chemical digital object's metadata.

[22]Simplified Molecular Input Line Entry Specification

– `recommendation`: It offers short, objective advice on how to improve the digital object FAIRness. Recommendations are useful for teaching FAIR. They have an associated priority, which reflects the gains in the degree of FAIRness they allow for when followed.

– `recommendation_details`: It details the `recommendation` on how the user can improve the evaluated digital object's degree of FAIRness.

– `max_score`: It is a value between 0 and 1 and it corresponds to the maximum value for a test that passed. For a cumulative metric, the sum of the maximum score of all of the metric's tests should be equal to or less than 1. For an alternate metric, its tests may have distinct maximum scores between 0 and 1 since only one of the tests that passes is considered for the metric's score.

– `skip`: When defined as "true", the test is not executed.

– `losses`: It details how a not-passed test negatively impacts its degree of FAIRness. *E.g.*, in the case of a test that evaluates *globally unique identifier*, losses will detail how the lack of such an identifier may make more difficult or even impossible to find a digital object.

– `maturity`: It defines the maturity to be applied to a metric if the test passes. The maturity may be [15]: (a) `Incomplete`: The Digital Object does not address the FAIR Principles. (b) `Initial`: Some initial characteristics are handled toward FAIR, although they do not meet the Principles' definitions. There is intent to comply with the FAIR principles and awareness of existing issues. (c) `Moderate`: Limited FAIR characteristics are handled, and there is a focus toward on complete alignment. (d) `Advanced`: Complete coverage of the FAIR Principles' characteristics aligned with organizational standards and practice.

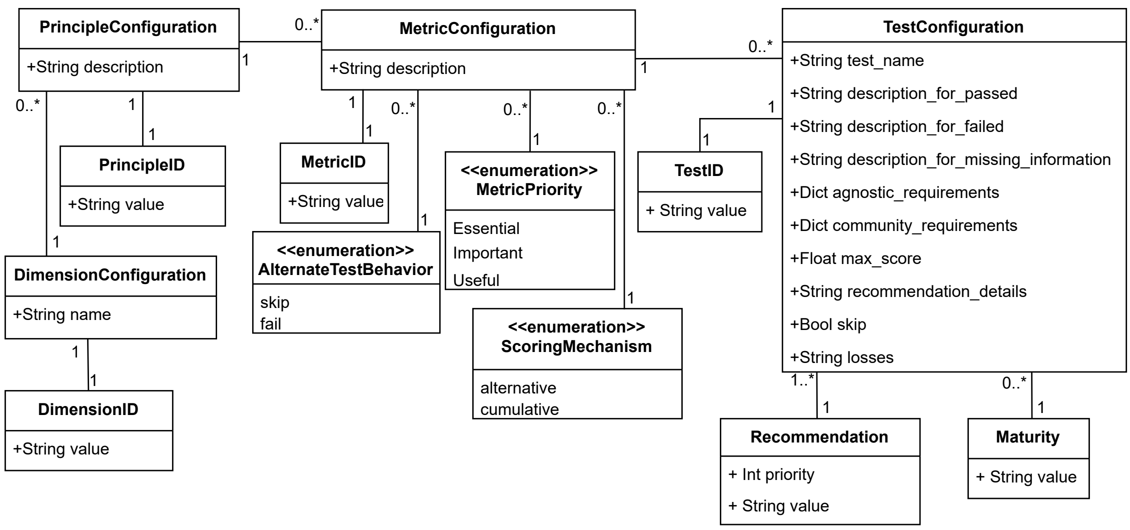

Figure 3: Class diagram of the FAIRness Configuration.

The data encapsulated by each class of the configuration structure is, for the most part, research community-specific. It was designed to enable non-expert stakeholders to alter the behavior of the evaluator. The `EvaluationConfiguration` class parses the configuration, creates instances of the configuration classes, and makes them available throughout the evaluation process.

The `FAIRness Result Aggregator` module implements functionalities for formatting test results, grouping them into coherent sets, and enriching them with relevant information. This information makes the evaluation process more transparent, indicates how the evaluation reflects the digital object's level of FAIRness, and clarifies the FAIR concepts related to the evaluation.

The result structure (Figure 4) is composed of the `TestResult`, `MetricResult`, `PrincipleResult` and `DimensionResult` classes. Instances of these classes expose the information defined in the configuration to users of the FAIRness evaluator generated by the framework. The constructor methods build `TestResult` instances copying from `TestConfiguration` all relevant information (*e.g.*, `max_score`, `recommendation_details`) to be available to the user.

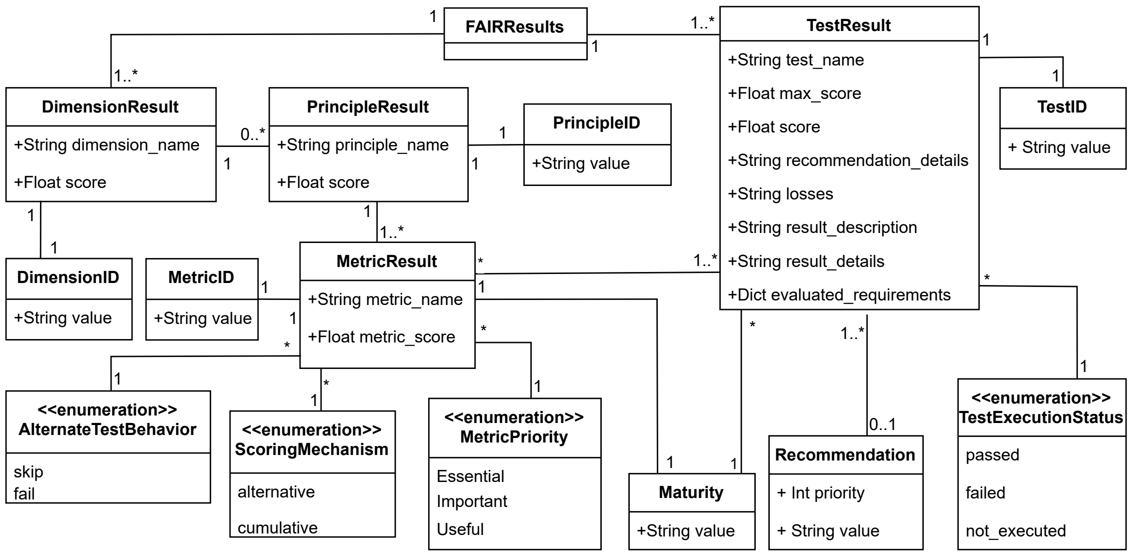

Figure 4: Class diagram of the FAIRness test results.

The framework implementation and configuration allow flexibility to create FAIRness evaluation services tailored to the community's needs. To provide an out-of-the-box solution, 2BFAIR is also offered with a default implementation, *i.e.*, we provide the 2BFAIR framework code along with a FAIRness evaluation tool developed using the framework itself. This implementation consists of a set of instantiated evaluators, a structured configuration, and an implementation of a (meta)data collector. It automatically evaluates the FAIRness of digital objects according to the FAIRsFAIR metrics [39] with a numeric score in the range of 0 to 100. It is available as a RESTful Web service (backend) using the proposed framework, implemented in Python using the FastAPI Web framework described using the OpenAPI[23] specification.

We also offer a 2BFAIR Web application (frontend) that consumes the default implementation (the backend). This frontend is divided into eight pages. We present some screenshots of the tool, highlighting its main Web pages.

- Evaluate: the start page where the user supplies a digital object identifier (*e.g.*, URI, URL, DOI) and runs the FAIRness evaluation (Figure 5).

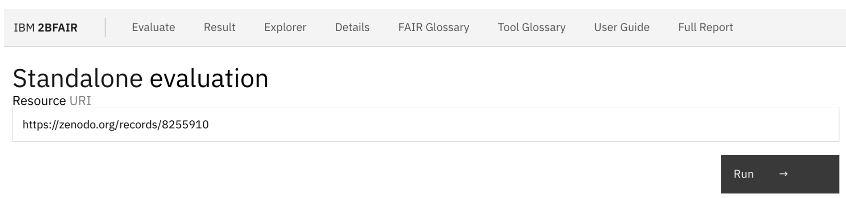

Figure 5: Evaluate: Web page to run a FAIRness evaluation.

- Result: a visual representation of the overall FAIRness result represented mainly by a badge.

[23] https://www.openapis.org/

- Explorer: an overview of the evaluation for each FAIR dimension (Figure 6). It is composed of four elements: (i) A header presenting information about the evaluated object, a link to download the evaluation result, and the FAIRness badge. (ii) A graphic presenting the score in percentage achieved for each dimension, with boxes representing the result achieved for each evaluated metric for the dimension. (iii) A legend (below the graphic) that explains the importance of each metric explains each metric's importance. (iv) A list of priority recommendations, *i.e.*, the most important recommendations for improving the FAIRness of the digital object.

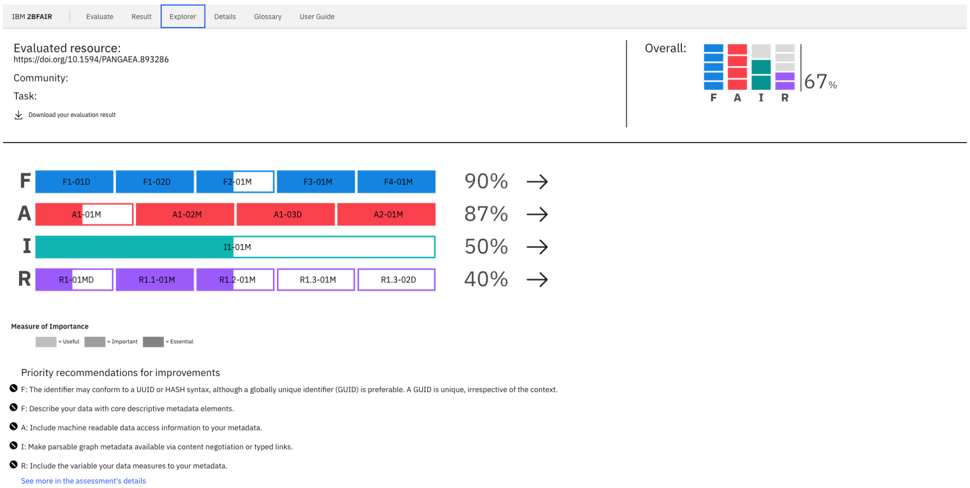

Figure 6: Explorer: Web page that presents an overview of the evaluation.

- Details: details of the evaluation for principles, metrics, and tests (Figure 7). The elements have the exact definition as presented for the configuration elements. For each principle, it presents: name: the definition of the principle; score: a value between 0 and 1 corresponding to the average of the scores of all principle's metrics. For each metric, it presents: name, maturity, score, and score mechanisms. For each test, it presents: status: if the test passed, failed, or was not executed; score: a value between 0 and the maximum test score, which varies from test to test; results: a summary of the test's result; results details: it provides details about the test execution; recommendations: it gives recommendations about what should be done to get a higher FAIRness score; recommendations details: it provides details about the recommendation, *e.g.*, steps teaching how to execute the improvement.

- FAIR Glossary: presents the definition of the FAIR concepts used in the tool development (Figure 8. We included links in all terms that appears on 2BFAIR's Web pages so that the user can navigate to the Tool Glossary by clicking on the word to access its definition, and learn about FAIR.

- Tool Glossary: presents the explanation of concepts used specifically in 2BFAIR-frontend tool.

- User Guide: explanations about each page and functionality available in the 2BFAIR-frontend.

- Full Report: presents a compilation of the Result, Explorer, and Details pages in a single page so that the user has a single view of the evaluation.

2BFAIR offers four axes of customization: (i) Definition of new tests by altering the source code; (ii) Definition of community-specific test parameters via test-specific configuration files; (iii) Deactivation of tests via the configuration file. (iv) Adaptation of test, metric, principle, and dimension attributes and weights according to the community needs via the configuration file. The first kind of customization requires software development expertise; however, the architectural pattern enforced by the 2BFAIR framework makes this process easier and less work-intensive. A non-expert user may perform other customizations, except setting community-specific parameters, which may require semantic web skills.

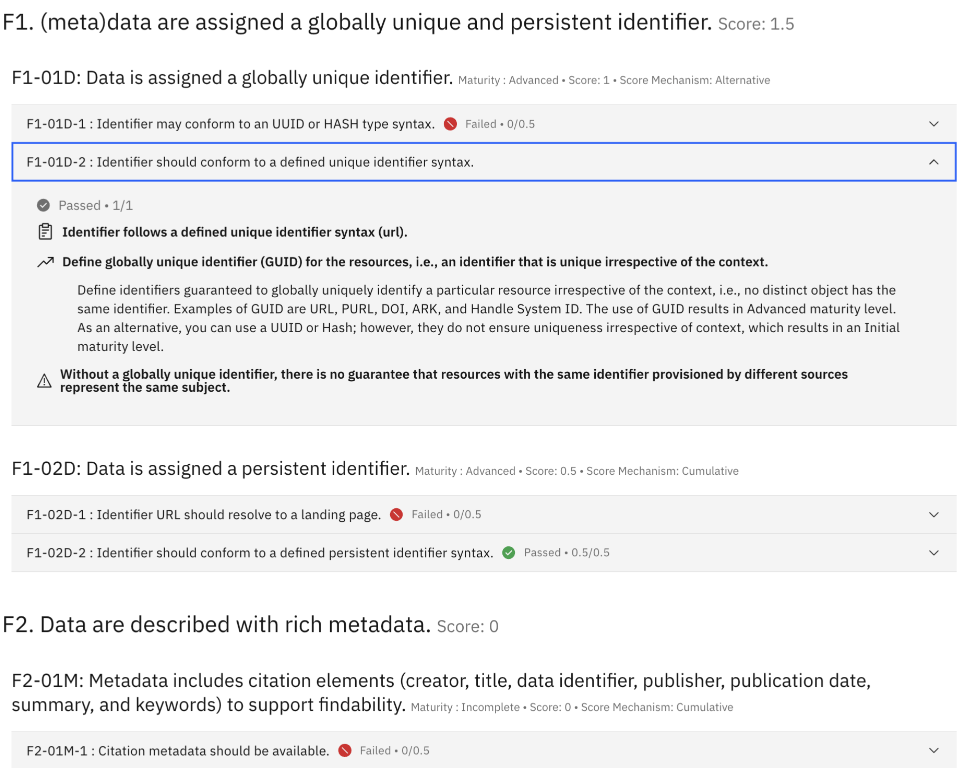

Figure 7: Details: Web page that presents all the evaluation details.

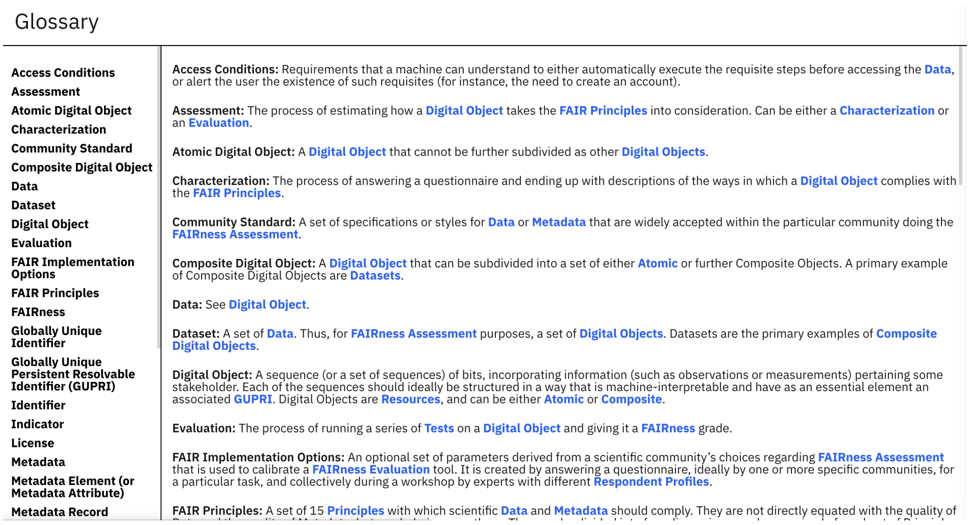

Figure 8: Glossary: Web page that presents definitions of FAIR concepts.

The web service's evaluation results are returned in JSON format, thus being processable by computing agents. The code is going to be available at GitHub[24]. In Section 4, we present the evaluation of 2BFAIR, comparing it against existing tools.

[24]The frontend and backend codes are in the process of becoming open-source by IBM. It was not concluded before the paper submission. This URL (https://github.com/leogazevedo/2BFAIR) will include the links when the process is finished.

4. Tools analysis

In this section, we present, in Section 4.1, how 2BFAIR caters to requirements elicited from the literature (Section 2), and, in Section 4.2, we compare 2BFAIR against the tools of the state-of-the-practice.

4.1. 2BFAIR

2BFAIR automatically (**R1**:✓) evaluates the FAIRness of data digital objects (**R5**:✓) according to the FAIRsFAIR metrics with a numeric score in the range of 0 to 100 (**R2**:✓). The evaluation can be applied to digital objects belonging to any domain since 2BFAIR is domain agnostic, but domain-specific evaluation criteria may be defined by the user (**R3**:✓). 2BFAIR's primary entities are Dimensions, Principles, Metrics, and Tests. Each Dimension (F, A, I, or R) has one or more principles (*e.g.*, F1, F2), which has one or more metrics (*e.g.*, F1-01D, which stands for the metric 01 for assessing Data according to Principle F1). Metrics are automatically evaluated by tests, *e.g.*, F1-01D-1, the test 1 of the metric F1-01D (**R4**:✓).

2BFAIR is available as a RESTful Web service, implemented in Python using the FastAPI Web framework, and its API is described using the OpenAPI specification (**R6**:✓). The service accepts the identifier of the data object to be evaluated as input. 2BFAIR is also available as a standalone web application (**R7**:✓), which provides a FAIRness badge offering an overview of the evaluation results for each dimension and a chart presenting details about metrics evaluation results (**R12**:✓). Currently, 2BFAIR's service and web client's code are publicly available under Apache-2.0 license (**R8**:✓). It is under development, although it can already be used to assess data digital objects (**R9**:✓). 2BFAIR supports the following stages of the data lifecycle [44]: processing and analyzing data, publishing and sharing data, preserving data, and re-using data (**R13**:✓).

2BFAIR offers customization (**R10**:✓) by altering the source code and adjusting configuration parameters. The former requires software development expertise but is supported by the architectural pattern we used to develop the framework, making this process easier and less work-intensive. The latter can be executed by non-expert users. The exception is the setting of community-specific parameters, which may require Semantic Web skills. 2BFAIR employs FAIR-enabling services (**R14**:✓). 2BFAIR still does not support the user to supply authentication credentials to access (meta)data available in private repositories for proper evaluation (**R11**:✗).

2BFAIR has a glossary that explains the main concepts of FAIR; it has a user manual that describes how to use the tool and details specific concepts of its user interface; the framework code has documentation (**R15**:✓).

2BFAIR's frontend requires little expertise. The user provides only the digital object identifier to perform an evaluation. The pages that present the results are straightforward to follow. On the other hand, Web service development skills are required to use the RESTful Web API, though this is expected (**R16**:✓).

The web service's evaluation results are returned in JSON format, thus processable by computing agents (**R17**:✓). Each test result includes recommendations, losses, and a description of the results. The result description details the criteria that caused the test to have passed, failed, or not to have been executed (**R18**:✓). The loss information presents what is lost if the digital object does pass the test. The recommendations instruct the user on how to improve the evaluated digital object so that it can pass the test (**R20**:✓). Thus, each test result helps the user learn about the FAIR concepts related to the test. Besides, the words related to FAIR used in the tool's web interface are linked to their definitions in a glossary of terms (**R19**:✓).

The results are available in readable natural language in the Web application (**R21**:✓). However, they are not stored in a searchable engine (**R22**:✗), and the versioning of FAIRness assessments is not supported (**R23**:✗).

4.2. Comparison of 2BFAIR against other tools

This section presents an analysis of the tools of the state-of-the-practice (F-UJI, FAIR Evaluator, FAIR Enough, FAIR Checker, and FAIR Eva) compared to 2BFAIR.

Almost all the tools fulfill the requirements R1 to R8. They are automated (R1), present a FAIRness grade (except FAIR-Evaluator) (R2), and execute independently from any specific domain (R3). Their metrics reference the principles they evaluate (R4). They point out the types of digital objects they evaluate (R5). They are available as RESTful Web services APIs (R6), provide a Web Application (R7), and exhibit their licenses (R8).

However, they differ regarding the other requirements, which are detailed below.

Development stage (R9): F-UJI and FAIR-Checker present evidence of wide usage; FAIR Evaluator and FAIR Enough do not have such proof, but none of these four tools explicitly explain their development stage. FAIR EVA is in the beta stage, and 2BFAIR is still under development.

Customization (R10): FAIR Evaluator and FAIR Enough are customizable by non-expert users when choosing the Maturity Indicators collections to run in an evaluation. Still, they do not support customization of the test parameters. FAIR EVA achieves customization via a plug-in architecture of specific FAIR metrics or maturity indicators. 2BFAIR allows non-expert stakeholders to alter the behavior of the evaluator via its configuration file, *e.g.*, definition of community-specific test parameters, deactivation of tests, and adaptation of test, metric, principle, and dimension attributes and weights. All tools can have their code customized by skilled software developers with knowledge of the FAIR principles, Semantic Web technologies, and standards. However, the architectural pattern enforced by the 2BFAIR framework makes this process easier and less work-intensive.

Authentication (R11): It is supported only by F-UJI.

Badge (R12): FAIR Evaluator and FAIR Enough do not provide a visual representation that summarizes the evaluation. F-UJI, FAIR-Checker, and FAIR EVA present visual representation as a multi-level pie chart and radar chart, respectively. 2BFAIR provides an overview of the evaluation results for each dimension and a chart presenting details about metrics results.

Data lifecycle (R13): Only 2BFAIR explicitly presents the data lifecycle phases to which it can be applied.

FAIR-enabling services (R14): F-UJI, FAIR evaluator, FAIR Enough, FAIR EVA, and 2BFAIR employ FAIR-enabling services. FAIR-Checker focuses on using Semantic Web technologies. All of them use libraries and standards.

Guidance (R15): The tools meet user guidance, except FAIR EVA, which has no documentation available regarding how to use the tool, like a user manual. The other tools have plenty of documentation, including papers and GitHub pages, with examples of using the tools. The code is open to be cloned, which allows software engineering experts to inspect it and deepen their knowledge.

Little expertice (R16): The tools Web interfaces are easy to use, requiring one to supply the digital object identifier and a few other pieces of data. However, using their APIs requires technical skills, which is expected.

Machine-actionable results (R17): Only FAIR-Checker does not export machine-actionable results.

Rating system (R18): The tools' reports present evidence and reasoning about evaluation.

Teach (R19): The reports are very technical for F-UJI, FAIR Enough, and FAIR Checker, *i.e.*, they are tied to the implementation and do not teach users about FAIR. 2BFAIR, FAIR Evaluator, and FAIR EVA teach about FAIR when the user executes them and goes through the assessment results.

Recomendations (R20): FAIR-Checker and 2BFAIR are the only ones that give explicit recommendations for FAIRness improvements.

Natural language (R21): The reports generated by 2BFAIR, FAIR Checker, and FAIR EVA are fully readable by non-experts.

Searchable (R22): None of the tools store the results even in a searchable engine.

Versioning (R23): The tools do not support results versioning, which would help to understand how a digital object's FAIRness evolves.

Overall, considering values of 1, 0.5, and 0 to complete (✓), partial (◗), and no (✗) fulfillment, respectively, to present the coverage of the requirements by the tools, they are very similar regarding requirement fulfillment: 2BFAIR (87%), F-UJI (74%), FAIR EVA (72%), FAIR-Checker (70%), FAIR-Evaluator (63%, and FAIR Enough (63%). They all follow good software development practices and use state-of-the-art technologies in Software Engineering and the Semantic Web. The reporting features should be improved, considering user-experience techniques and user evaluations. The storage of results and versioning are complex features to be implemented, and no tool appraised attempts to do it. Although 2BFAIR has the highest percentage, other tools can be evolved to improve the coverage of their requirements. However, it requires technical expertise since their implementation does not follow a framework architecture, one of the main characteristics of 2BFAIR, which aims to allow for customization.

5. Conclusion

This work presented 2BFAIR, a framework for automated FAIRness assessment. 2BFAIR was developed as a framework to support customization of the FAIRness evaluation according to community characteristics. It provides ready-to-use pieces of code (*frozen-spot*) that encapsulate complex logic common to any FAIRness assessment process that the user does not have to change. On the other hand, 2BFAIR provides semi-finished building blocks (*hot spots*) that users may adjust to their particular needs. The framework architecture allows creating tools based on 2BFAIR tailored to specific domains.

We also provided a default implementation of 2BFAIR, *i.e.*, an automated tool for FAIRness assessment created with the 2BFAIR framework code. This implementation consists of evaluators for all the principles, a structured configuration, and an implementation of a (meta)data collector. 2BFAIR users can use this implementation directly to start executing FAIRness assessments. Technical users can use it as an example of 2BFAIR instantiation to develop their tool based on our framework.

We analyzed this default implementation of 2BFAIR against the state-of-the-practice tools based on a set of desirable requirements that tools for automated FAIRness evaluation should fulfill. 2BFAIR reached the highest percentage of fulfillment (87%), while other tools reached at most 74%. No tool meets all the requirements, so none stands out as the state-of-the-art. 2BFAIR is a good starting point since it was implemented as a framework and provides a default implementation for direct use. However, we emphasize that to make a tool choice, one should consider not only the percentage of fulfillment but also understand the more critical requirements for their specific scenario, the characteristics of each tool, and the difficulties in improving each tool's implementations in case it becomes vital to support requirements not adequately addressed.

As future work, we will evolve the 2BFAIR framework to support the three missing requirements, *i.e.*, implement functionalities to handle authentication (R11), to store the evaluation results in a searchable resource (R22), and support FAIRness assessments versioning (R23). We will evaluate the 2BFAIR framework with users to generate applications for real scenarios from different domains. Another proposal of work is to evaluate how we can automatically generate code for the hot spots, *e.g.*, based on the configuration file, templates, or using code assistants (like the IBM Watsonx Code Assistant[25]).

In this work, we evaluated how the tools comply with requirements that automated tools for FAIRness assessment of digital objects should meet. From another perspective, we could analyze how the tools themselves abide by the FAIR principles, *e.g.*, by examining how they meet the FAIR for research software (FAIR4RS) [45].

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

[25]https://www.ibm.com/products/watsonx-code-assistant

References

[1] M. D. Wilkinson, M. Dumontier, et al., The fair guiding principles for scientific data management and stewardship, Scientific data 3 (2016) 1–9. doi:10.1038/sdata.2016.18.

[2] A. Jacobsen, R. de Miranda Azevedo, N. Juty, B. *et al.*, FAIR Principles: Interpretations and Implementation Considerations, Data Intelligence 2 (2020) 10–29. doi:10.1162/dint_r_00024.

[3] C. Trojahn, M. Kamel, A. Annane, et al., A FAIR core semantic metadata model for FAIR multidimensional tabular datasets, in: Knowledge Engineering and Knowledge Management, Springer, 2022, pp. 174–181. doi:10.1007/978-3-031-17105-5_13.

[4] E. Amdouni, C. Jonquet, FAIR or FAIRer? an integrated quantitative fairness assessment grid for semantic resources and ontologies, in: Communications in Computer and Information Science, volume 1537 CCIS, 2022, pp. 67 – 80. doi:10.1007/978-3-030-98876-0_6.

[5] J. van Soest, A. Choudhury, N. Gaikwad, M. Sloep, A. Dekker, Annotation of existing databases using Semantic Web technologies: making data more FAIR, in: 12th International Conference on Semantic Web Applications and Tools for Health Care and Life Sciences, 2019, pp. 94–101.

[6] E. Amdouni, S. Bouazzouni, C. Jonquet, O'FAIRe: Ontology fairness evaluator, https://github.com/agroportal/fairness, Accessed on 2025–04-10.

[7] M. D. Wilkinson, M. Dumontier, S.-A. Sansone, L. O. Bonino da Silva Santos, et al., Evaluating FAIR maturity through a scalable, automated, community-governed framework, Scientific data 6 (2019) 1–12. doi:10.1038/s41597-019-0184-5.

[8] F. Aguilar Gómez, I. Bernal, FAIR EVA: Bringing institutional multidisciplinary repositories into the FAIR picture, Scientific Data 10 (2023) 1–19. doi:10.1038/s41597-023-02652-8.

[9] A. Gaignard, T. Rosnet, F. De Lamotte, V. Lefort, M.-D. Devignes, FAIR-Checker: supporting digital resource findability and reuse with Knowledge Graphs and Semantic Web standards, Journal of Biomedical Semantics 14 (2023) 1–7. doi:10.1186/s13326-023-00289-5.

[10] M. E. Markiewicz, C. J. P. de Lucena, Object oriented framework development, XRDS 7 (2001) 3–9. URL: https://doi.org/10.1145/372765.372771. doi:10.1145/372765.372771.

[11] W. Pree, Meta patterns — A means for capturing the essentials of reusable object-oriented design, in: M. Tokoro, R. Pareschi (Eds.), Object-Oriented Programming, Springer, Berlin, Heidelberg, 1994, pp. 150–162. doi:10.1007/BFb0052181.

[12] L. G. Azevedo, G. Banaggia, J. Tesolin, R. Cerqueira, Analysis of automated tools for FAIRness evaluation: A literature perspective, in: The Semantic Web: ESWC 2024 Satellite Events, Springer Nature Switzerland, 2025, pp. 149–166. doi:10.1007/978-3-031-78955-7_15.

[13] A. Carrera-Rivera, W. Ochoa, F. Larrinaga, G. Lasa, How-to conduct a systematic literature review: A quick guide for computer science research, MethodsX 9 (2022) 1–15. doi:10.1016/j.mex.2022.101895.

[14] A. Devaraju, R. Huber, F-UJI - An Automated FAIR Data Assessment Tool, Zenodo, 2020. doi:10.5281/zenodo.4063720.

[15] A. Devaraju, R. Huber, An automated solution for measuring the progress toward FAIR research data, Patterns 2 (2021) 1–16. doi:10.1016/j.patter.2021.100370.

[16] fairsharing, Fairassist.org, https://fairassist.org/, Accessed on 2025–04-10.

[17] Institute of Data Science, FAIR Enough, https://fair-enough.semanticscience.org/, Accessed on 2025–04-10.

[18] F. Aguilar Gómez, I. Bernal, FAIR EVA (Evaluator, Validator & Advisor), https://github.com/EOSC-synergy/FAIR_eva, Accessed on 2024–10-29.

[19] E. Amdouni, S. Bouazzouni, C. Jonquet, O'FAIRe: Ontology FAIRness evaluator in the agroportal semantic resource repository, in: The Semantic Web: ESWC 2022 Satellite Events, Springer, Cham, 2022, pp. 89–94. doi:10.1007/978-3-031-11609-4_17.

[20] HowFAIRis, HowFAIRis, https://github.com/fair-software/howfairis, Accessed on 2025–04-10.

[21] D. Garijo, O. Corcho, M. Poveda-Villalón, OEG FAIR Ontologies Assessment (FOOPS!), https://github.com/oeg-upm/fair_ontologies, Accessed on 2025–04-10.

[22] D. Garijo, O. Corcho, M. Poveda-Villalón, FOOPS!: An Ontology Pitfall Scanner for the FAIR

Principles, in: International Semantic Web Conference (ISWC) 2021: Posters, Demos, and Industry Tracks, volume 2980 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 1–4.
[23] E. González, A. Benítez, D. Garijo, FAIROs: Towards FAIR assessment in research objects, in: Linking Theory and Practice of Digital Libraries, Springer, 2022, pp. 68–80. doi:10.1007/978-3-031-16802-4_6.
[24] CLARIN, Clarin curation dashboard, https://github.com/clarin-eric/curation-dashboard, 2025.
[25] CLARIN, Curation dashboard 7.0.0, https://curation.clarin.eu/, Accessed on 2025–04-10.
[26] A. Czerniak, Lightweight FAIR assessment in the OpenAIRE Validator, 2021. doi:10.5281/zenodo.5541133.
[27] OpenAIRE, OpenAIRE's Repository Manager, https://provide.openaire.eu/home, Accessed on 2025-04-10.
[28] J. Wang, S. Gesing, R. Johnson, N. Meyers, D. Minor, PresQT, https://github.com/Lucy-Family-Institute/presqt, Accessed on 2025–04-10.
[29] J. Wang, S. Gesing, R. Johnson, N. Meyers, D. Minor, PresQT Data and Software Preservation Quality Tool Project, https://osf.io/d3jx7/, 2022. doi:10.17605/OSF.IO/D3JX7.
[30] J. Menke, P. Eckmann, I. B. Ozyurt, M. Roelandse, N. Anderson, J. Grethe, A. Gamst, A. Bandrowski, Establishing Institutional Scores With the Rigor and Transparency Index: Large-scale Analysis of Scientific Reporting Quality, Journal of Medical Internet Research 24 (2022). doi:10.2196/37324.
[31] Sciscore, SciScore: Enhance rigor and reproducibility in scientific research, https://sciscore.com, Accessed on 2025–04-10.
[32] J. Wang, S. Gesing, R. Johnson, N. Meyers, D. Minor, Presqt, https://presqt.readthedocs.io/en/latest/services.html, Accessed on 2025–04-10.
[33] C. Sun, V. Emonet, M. Dumontier, A comprehensive comparison of automated FAIRness evaluation tools, in: CEUR Workshop Proceedings, volume 3127, 2022, pp. 1–10.
[34] N. Krans, A. Ammar, P. Nymark, E. Willighagen, M. Bakker, J. Quik, FAIR assessment tools: evaluating use and performance, NanoImpact 27 (2022). doi:10.1016/j.impact.2022.100402.
[35] K. Peters-Von Gehlen, H. Höck, A. Fast, D. Heydebreck, A. Lammert, H. Thiemann, Recommendations for Discipline-Specific FAIRness Evaluation Derived from Applying an Ensemble of Evaluation Tools, Data Science Journal 21 (2022). doi:10.5334/dsj-2022-007.
[36] D. Slamkov, V. Stojanov, B. Koteska, A. Mishev, A comparison of data FAIRness evaluation tools, in: CEUR Workshop Proceedings, volume 3237, CEUR-WS, 2022, pp. 1–12.
[37] L. Richardson, M. Amundsen, S. Ruby, RESTful web APIs: services for a changing world, O'Reilly Media Inc.„ 2013.
[38] J. M. Aronsen, O. Beyan, N. Harrower, A. Holl, et al., Recommendations on FAIR metrics for EOSC, Publications Office of the European Union, LU, 2021. doi:10.2777/70791.
[39] A. Devaraju, R. Huber, M. Mokrane, P. Herterich, L. Cepinskas, J. de Vries, H. L'Hours, J. Davidson, A. White, FAIRsFAIR Data Object Assessment Metrics, Technical Report, Zenodo, 2022. doi:10.5281/zenodo.6461229.
[40] M. Wilkinson, et al., The FAIR Maturity Evaluation Service, https://fairsharing.github.io/FAIR-Evaluator-FrontEnd, Accessed on 2025–04-10.
[41] L. Ehrlinger, W. Wöß, Towards a definition of knowledge graph, SEMANTICS 2016: Posters and Demos Track 48 (2016) 1–2.
[42] T. Rosnet, A. Gaignard, M.-D. Devignes, FAIR-checker, https://fair-checker.france-bioinformatique.fr/, Accessed on 2025–04-10.
[43] RDA, FAIR Data Maturity Model: specification and guidelines, 2020. doi:10.15497/rda00050.
[44] G. Mosconi, Q. Li, D. Randall, H. Karasti, P. Tolmie, J. Barutzky, M. Korn, V. Pipek, Three Gaps in Opening Science, Computer Supported Cooperative Work (CSCW) 28 (2019) 749–789. doi:10.1007/s10606-019-09354-z.
[45] M. Barker, N. P. Chue Hong, D. S. Katz, A.-L. Lamprecht, C. Martinez-Ortiz, F. Psomopoulos, J. Harrow, L. J. Castro, M. Gruenpeter, P. A. Martinez, T. Honeyman, Introducing the FAIR Principles for research software, Scientific Data 9 (2022) 622. doi:10.1038/s41597-022-01710-x.