FAIR Learning Technologies with Web Components and Packages

Frederic Salmen [1], Martin Breuer [2], Sergej Görzen [1], Malte Persike [2], and Ulrik Schroeder [1]

Abstract: Making the diverse software artifacts of the learning technologies community findable, accessible, interoperable, and reusable (FAIR) can be a technical challenge. We introduce a concept informed by our research involving packages and components to achieve FAIRness for web-based artifacts. This result is presented as a guideline to make FAIR technology choices when creating web-based learning technologies. The guideline compares classic choices with new paths afforded by technological innovation of the web platform. Supported by practical examples (learning analytics dashboards, e-assessment, and explorables) we discuss practical applications of our result.

Keywords: FAIR principles, Web Components, Packages, Reusable Software

1 Motivation

Current learning technologies like Learning Management Systems (LMS), multimodal learning resources, and e-assessment systems are used at many educational institutions such as universities or schools. Due to different requirements and contexts, there is a broad range of different platforms for individual needs. Recent discussions in the learning technology community question the current practices and lack of FAIRness related to developing software artifacts [KS22]. The authors favor fostering community collaboration, enhancing software quality and interoperability, and ensuring that software and IT infrastructure are acknowledged as critical scholarly research outputs, creating a more open and efficient scientific ecosystem. The FAIR principles were first proposed for research data in 2016 [Wi16]. Since that pivotal publication on this topic in the learning technologies community, progress has been made across other communities. In 2022, an adaptation of the FAIR model specifically for research software [Ch22] was proposed, which is the result of a three-year standardization process by an international and interdisciplinary working group of over 250 members, including researchers, developers, policymakers, and users. The model retains the FAIR acronym with four foundational principles, each supported by some guiding principles. The principles as stated in [Ch22] are:

1 RWTH Aachen University, Learning Technologies, Aachen, Germany,
salmen@cs.rwth-aachen.de, https://orcid.org/0009-0007-5206-1146;
goerzen@cs.rwth-aachen.de, https://orcid.org/0000-0003-3853-2435;
schroeder@cs.rwth-aachen.de, https://orcid.org/0000-0002-5178-8497
2 RWTH Aachen University, Learning Technologies, Aachen, Germany,
breuer@medien.rwth-aachen.de, https://orcid.org/0009-0008-0749-5110;
persike@cls.rwth-aachen.de, https://orcid.org/0000-0002-7825-089X

doi:10.18420/delfi2024_23

- F Software and its associated metadata are easy for humans and machines to find.
  - F_1 Software is assigned a globally unique and persistent identifier (ID).
  - F_2 Software is described with rich metadata.
  - F_3 Metadata clearly and explicitly include the identifier of the software described.
  - F_4 Metadata are FAIR, searchable and indexable.
- A Software and its metadata are retrievable via standardized protocols.
  - A_1 Software is retrievable by its identifier using a standardized comm. protocol.
  - A_2 Metadata are accessible, even when the software is no longer available.
- I Software interoperates with other software by exchanging data and/or meta-data and/or through interaction via application programming interfaces (APIs) described through standards.
  - I_1 Software reads/writes/exchanges data with domain-rel. community standards.
  - I_2 Software includes qualified references to other objects.
- R Software is both usable (can be executed) and reusable (can be understood, modified, built upon, or incorporated into other software).
  - R_1 Software is described with a plurality of accurate and relevant attributes.
  - R_2 Software includes qualified references to other software.
  - R_3 Software meets domain-relevant community standards.

In this practical report, we outline a concept for applying these principles to a major subset of artifacts in the learning technologies research community: Web-based artifacts, which are implemented using web technologies, are prevalent in the community today (see next section). Their specific properties may be exploited to improve FAIRness. To this end, we pose the following research question:

RQ: How can the FAIRness (findability, accessibility, interoperability, and reusability) of web-based artifacts in learning technologies be achieved practically?

2 Related Work

In the 1990s and 2000s, learning objects (LO) emerged as an initial endeavor to address the challenge of blending educational materials [Wi02]. Through the establishment of standards, the aim was to make learning resources easily identifiable, reusable, and compatible, essentially to make them FAIR. Yet, a key drawback of LO surfaced when viewed through the lens of remix: “Learning objects can be aggregated but not adopted“ [Wi08].

A necessary first step is providing access to the source code of the artifact. Open sourcing makes the implementation of the artifact transparent and allows for fine-grained reuse of parts of the implementation. This total control over the granularity of the reuse is both an advantage and a disadvantage: Any part of the implementation, no matter how small, can be extracted. However, this is essentially looking inside the black box. Usually, there

is no standard abstraction to allow the reuse of something specific, forcing researchers to understand the whole implementation and to re-implement the part they need.

What learning technology artifacts have in common is that they are likely to be web-based, using the browser as a platform. The browser platform has seen many innovations in the last 10 years. Starting with the development of the HTML5 standard by the W3C and WHATWG, today's web marks a shift from the static, document-oriented web of the 1990s and 2000s towards a modern, app-oriented web. Previously, a lot of a web app's functionality had to be implemented on the server side (e.g., storage or routing) or integrated by plugins (e.g., Flash for multimedia). Today's web unites all these features in a single platform as a static document that can be called from everywhere (e.g., by using the browser on a smartphone). [Ka17]

Recently, the learning technologies research community has started adopting some of the new concepts of the modern web platform, such as components [SS23; St23]. Still, wide adoption is lagging behind compared to the software industry (React pioneered components in 2013 [Co15], and cross-browser support for Web Components was achieved in 2018 [WH24]) and is a promising path towards more FAIRness in research software.

3 FAIR Research Web Technology Flowchart

As an intermediary tool between the rather abstract FAIR-RS principles and the practical implementation of web-based learning technologies, we present a schema that can help make web-based artifacts more FAIR (compare Fig. 1), which we call the "FAIR Research Web Technology Flowchart". The FAIRness of the source code as research data, e.g. by forking an open source repository, is also encouraged but not addressed here.

Our flowchart (see Fig. 1) has two main paths, and each path leads to a different degree of FAIRness, as technology choices may or may not fulfill certain principles. We will go through two possible paths: the classic path (static file) and the FAIR path (modularized technology). In both paths, we start with a web application, achieving R_3 (domain-relevant community standards) by default since the web is heavily standardized.

Classic path: In the first case, we may produce a website as an artifact (e.g., an 'index.html' with related assets). Then, that website is served on any web hosting solution (for example, a self-hosted web server such as nginx). This covers both F_1 (globally unique, persistent ID) and A_1 (retrievable by ID) - the artifact can be retrieved using the URL as a globally unique identifier. It is worth noting that this identifier has limited persistency: If the site is restructured or the resource becomes otherwise unavailable, both principles no longer apply. Finally, the served website may be embedded into other web applications, for example, with an 'iframe.' While this path does not provide much FAIRness, it is very easy to follow: Since the software needs to be published as a whole anyway to be used normally, this almost enables reuse by default. It also provides security benefits as websites have strong and well-testing sandboxing protections.

FAIR path: In this path, a web application (R_3 applies) forks into two options for modularization: Component or Library. With Components, we mean encapsulated, reusable user interface elements (for example, a performance graph for learning analytics). Established libraries such as React and standards such as Web Components facilitate this common pattern to enable reuse across applications. Similarly, parts of the model layer or application logic can be extracted as an API or Library. With a Library, we mean related functionality intended for reuse (for example, jQuery [BRK15]). Almost all programming languages facilitate this pattern with a module system (e.g., ES Modules [Ec23]). Both options can fulfill multiple principles, namely I_1 (read/write/exchange data) with attributes, events, function calls, etc., I_2 (qualified references to objects) with the language's built-in reference system, R_1 (described with license and provenance) with associated docstrings and comments, and R_2 (qualified references to software) with composition of components or libraries through imports. Of course, components and modules are not exclusive: Any web application may have components and libraries worth modularizing. At this step, the researcher must deal with the fundamental challenges of modularization, for example, decoupling the component or library from other application parts or finding the right level of abstraction.

After that, two choices supplement each other to produce an artifact. An asset may be produced, such as a single file bundle using a bundle (e.g., Vite), so the user of the library or component needs only load a single file. This is optional since some libraries and components can be used directly (e.g., a small library in a single JS file). To achieve more FAIRness, the assets may be wrapped in a package. In some use cases, this may be a Node style 'package.json' or a container (e.g., Docker). Creating a package fulfills principles F_2 and F_3 as packages include the identifier and metadata such as the author, license, etc.

Next, there are two choices again. Assets may be served directly like a website, fulfilling F_1 and A_1 (see classic path). Alternatively, a package can be published in a registry such as NPM, Github, or Docker Hub (alongside version control to associate the code, the artifact, and their version). This fulfills F_1 (globally unique, persistent ID) with guaranteed immutable, versioned package names, F_4 (metadata are FAIR, searchable, and indexable) with supported metadata fields, A_1 (retrievable by ID) with package managers such as yarn, and A_2 (metadata available when software is not anymore) with deprecated packages remaining registered. While registered packages may also become unavailable like any web resource, dedicated registries such as NPM are both stable and easy to mirror, should the registry provider ever become defunct. Ultimately, this provides guarantees similar to the DOI system regarding stable, unique identifiers.

Finally, the package or served resource can be integrated into another web application. Compared to the classic path, reuse is more granular: The other application does not need to embed a whole website but can reference the URL of the served resource for a script or stylesheet or use a package manager to install a registered package. An even smaller granularity of reuse is possible by importing only the needed parts of a library. As a disadvantage, this kind of reuse requires more trust as code from the original web application is directly executed in the other web application.

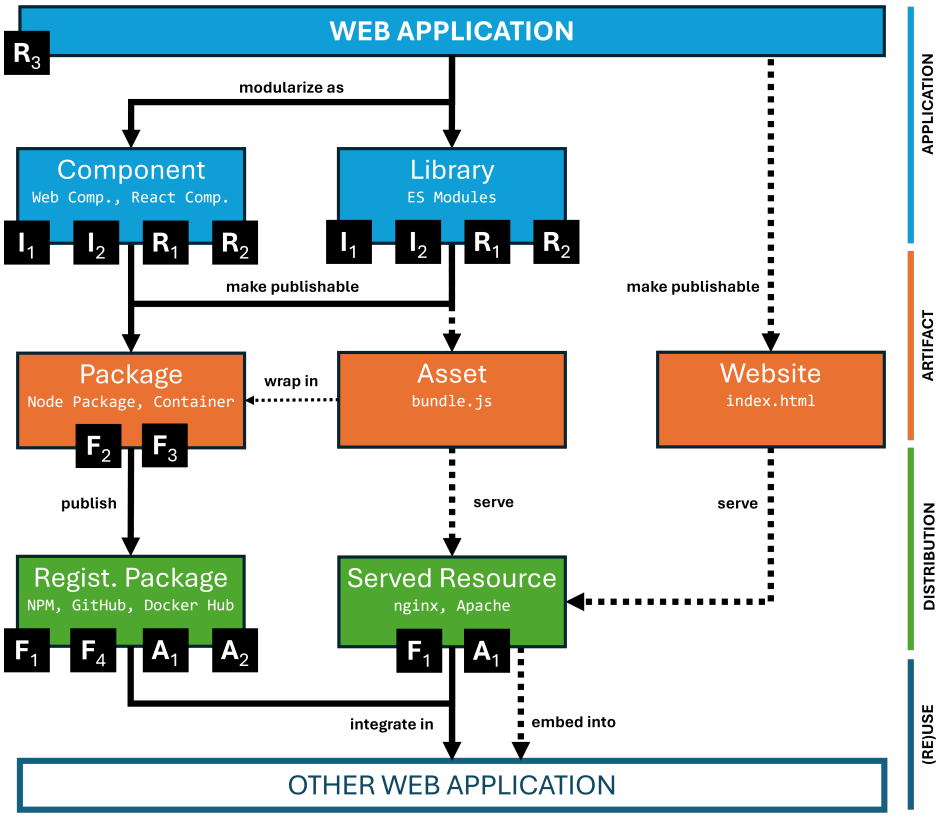

Fig. 1: FAIR Research Web Technology Flowchart. For R_1 etc. see FAIR-RS principles in Sect. 1

4 Practical experiences

One use case for the workflow introduced is an assessment analytics tool to support examiners in improving teaching and assessment practices [Br23]. Current efforts include the development of frontend components with a seamless integration in the assessment system Dynexite. Since the data analysis and result storage take place in a central learning analytics system [Br23], an additional goal is to decouple the development process from the assessment system, also enabling reuse in other systems. Further requirements include displaying multiple instances of the frontend components, such as an information pane for a course on an overview page. For example, that information pane was modularized as a standard web component using the Custom Element API without a library. Using GitLab CI pipelines, the web component and the service were bundled into a single bundle.js file, which was exposed via a Docker image and pushed onto the GitLab Docker Registry. The component can then be reused via the registry or by referencing the served resource.

The authoring tool WebWriter [SRS23][SS24] uses both components and packages. It is intended to help teachers create and remix digital content such as text, images, audio, video, and interactive content (e.g., simulations, interactive maps or charts), bringing them together in multimodal resources called explorables [Ho20; Vi11]. The interactive elements that can be added to explorables are called widgets. Developers may implement their widgets using web components and distribute them as Node packages via NPM. In short, web components and packages become the plugin system of the authoring tool. This encourages developers to create FAIR widgets to be compatible with WebWriter. Widgets are loosely coupled to WebWriter, meaning web applications can reuse them without changing the source code.

5 Conclusion

In this paper, we introduced the FAIR Research Web Technology Flowchart, a tool to apply the FAIR-RS principles to web-based learning technologies. This tool helps researchers by guiding them through the step-by-step implementation of a FAIR web application. While it is specific to modern frontend web technologies, the approach advocated can be used with almost any web-based framework and technology stack, as showcased by the diverse set of practical experiences reported. It also builds upon the best practices of the wider web development community.

While our approach aims to foster modularization, there are some limitations. In terms of scope, the concept is intended for modern single-page applications. Whether it applies to a server-focused architecture where most of the application logic is run server-side is unclear and worth investigating. Also, the flowchart intentionally only covers the reuse of software as an artifact, not the reuse of source code through a public, open-source repository. Regarding assuring software quality, following the "FAIR path" of the flowchart *can* lead to FAIR research software, quality control (including transparent version control) is still needed at every level. Another challenge lies in the validation of FAIRness, for example in the context of a conference where reviewers need to ensure the FAIRness of research software associated with submissions efficiently and reliably. Finally, institutional challenges, such as poor maintenance and a lack of collaboration, cannot be addressed through a technical concept.

For future work, standardized solutions such as Custom Elements Manifest (a JSON file describing Web Components) can be helpful to develop assistive tools for that validation. It may also be valuable to create a more permanent identifier for software artifacts (in case of changing registries), such as a DOI resolving to a registry name. Overall, we assume that our process can be refined further through more and different practical applications using different technology stacks (e.g., Learning Analytics components as Web Components or WebXR) and may also be aided by developing tools specific to the learning technologies community (for example integrating learning analytics with xAPI into learning technology web components). Further, through our flowchart, we plan to explore mechanisms to check the FAIRness of web projects automatically.

Acknowledgments We were supported by the research project NERD II (005-2201-0014), sponsored by the state of North Rhine-Westphalia.

References

[Br23] Breuer, M.; Brocker, A.; Persike, M.; Schroeder, U.: AxEL - Eine modulare Softwarekomponente für ein dediziertes E-Prüfungssystem zur Generierung von xAPI-Statements für Assessment Analytics. ISBN: 9783885797326 Publisher: Gesellschaft für Informatik e.V., 2023, DOI: 10.18420/DELFI2023-17.

[BRK15] Bibeault, B.; Rosa, A. D.; Katz, Y.: jQuery in Action. Simon and Schuster, 2015, ISBN: 978-1-63835-337-9.

[Ch22] Chue Hong, N. P.; Katz, D. S.; Barker, M.; Lamprecht, A.-L.; Martinez, C.; Psomopoulos, F. E.; Harrow, J.; Castro, L. J.; Gruenpeter, M.; Martinez, P. A.: FAIR Principles for Research Software (FAIR4RS Principles). Zenodo, 2022.

[Co15] Cory Gackenheimer: Introduction to React. 2015.

[Ec23] Ecma International: ECMA-262, 2023, visited on: 04/11/2024.

[Ho20] Hohman, F.; Conlen, M.; Heer, J.; Chau, D. H. (: Communicating with Interactive Articles. Distill 5 (9), e28, 2020, ISSN: 2476-0757, DOI: 10.23915/distill.00028.

[Ka17] Kaul, M.; Kless, A.; Bonne, T.; Rieke, A.: Game Changer for Online Learning Driven by Advances in Web Technology. 2017.

[KS22] Kiesler, N.; Schiffner, D.: On the Lack of Recognition of Software Artifacts and IT Infrastructure in Educational Technology Research. In: 20. Fachtagung Bildungstechnologien (DELFI). Gesellschaft für Informatik e.V., pp. 201–206, 2022, ISBN: 978-3-88579-716-6.

[SRS23] Salmen, F.; Roepke, R.; Schroeder, U.: WebWriter: A System to Author and Remix Explorables–Requirements & First Prototype. 2023.

[SS23] Selmanagić, A.; Simbeck, K.: Breaking It Down: On the Presentation of Fine-Grained Learning Objects in Virtual Learning Environments. In: 21. Fachtagung Bildungstechnologien (DELFI). Gesellschaft für Informatik e.V., pp. 155–160, 2023, ISBN: 978-3-88579-732-6.

[SS24] Salmen, F.; Schroeder, U.: Evaluating Authoring Tools with the Explorable Authoring Requirements, 2024, arXiv: 2403.17714 [cs], visited on: 04/09/2024.

[St23] Striewe, M.: Strukturformeln für Moleküle zeichnen und differenziertes Feedback erhalten: Eine integrierte Lösung im Rahmen des E-Assessment-Systems JACK. In: 21. Fachtagung Bildungstechnologien (DELFI). Gesellschaft für Informatik e.V., pp. 273–274, 2023, ISBN: 978-3-88579-732-6.

[Vi11] Victor, B.: Explorable Explanations, http://worrydream.com/ExplorableExplanations/, 2011, visited on: 08/23/2022.

[WH24] WHATWG: HTML Living Standard, https://html.spec.whatwg.org/, 2024, visited on: 04/10/2024.

[Wi02] Wiley, D. A.: The Instructional Use of Learning Objects. Agency for instructional technology Bloomington, IN, 2002.

[Wi08] Wiley, D. A.: The Learning Objects Literature. In: Handbook of Research on Educational Communications and Technology. Routledge, pp. 345–353, 2008, visited on: 04/10/2024.

[Wi16] Wilkinson, M. D.; Dumontier, M.; Aalbersberg, I. J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.-W.; da Silva Santos, L. B.; Bourne, P. E.: The FAIR Guiding Principles for Scientific Data Management and Stewardship. Scientific data 3 (1), pp. 1–9, 2016.