ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

in cotutela con KATHOLIEKE UNIVERSITEIT LEUVEN

DOTTORATO DI RICERCA IN

PATRIMONIO CULTURALE NELL'ECOSISTEMA DIGITALE

Ciclo 38

Settore Concorsuale: 01/B1 - INFORMATICA

Settore Scientifico Disciplinare: INF/01 - INFORMATICA

HERITRACE: ENABLING DOMAIN EXPERT PARTICIPATION IN SEMANTIC DATA CURATION WITH INTEGRATED PROVENANCE AND CHANGE TRACKING

Presentata da: Arcangelo Massari

Coordinatore Dottorato

Francesca Tomasi

Supervisore

Silvio Peroni

Co-supervisore

Anastasia Dimou

Esame finale anno 2026

Abstract

Cultural heritage institutions increasingly adopt Semantic Web technologies for FAIR compliance. In cultural heritage, semantic data is inherently interpretative and requires human curation, yet technical complexity prevents domain experts from contributing. This thesis addresses the usability gap through two research questions: RQ1 asks how to design interfaces enabling domain experts to curate RDF data without requiring technical expertise while maintaining provenance and change tracking; RQ2 asks how technical staff can perform one-time configuration of curation environments for specialized domains.

Two case studies serve distinct roles: OpenCitations Meta validates that barriers exist at scale in systems processing 124 million entities; ParaText serves as testbed for guerrilla testing. These reveal five convergent requirements: provenance management, change tracking, usability for both domain experts and configurators, flexible customization, and integration with existing RDF collections.

HERITRACE addresses these requirements through a framework built on the OpenCitations Data Model for provenance and change tracking, with the Time Agnostic Library enabling reconstruction of past entity states. Technicians use familiar SHACL shapes and YAML rules to configure entity types, validation constraints, and display settings; the framework then generates interfaces from these configurations. End users create, modify, delete, and merge entities and restore previous states through version history, without awareness of the underlying RDF infrastructure. Evaluation with 9 end users and 10 technicians combines quantitative measures and grounded theory analysis. For RQ1, end users completed curation tasks with 67% to 100% success and above-average usability (SUS 78.9). For RQ2, technicians achieved 90% success with excellent usability (SUS 83.8).

Since most institutions maintain collections in relational databases, preliminary research explores extending RML to enable inverse transformations: RDF serves as a lingua franca, with HERITRACE curating semantic data while inverse mappings transfer modifications back to original databases, allowing institutions to adopt FAIR principles without changing their existing infrastructure.

Sintesi

Le istituzioni del patrimonio culturale stanno adottando sempre più le tecnologie Semantic Web per aderire ai principi FAIR. I dati semantici in questo ambito richiedono l'intervento di esperti di dominio, ma la complessità tecnica impedisce loro di contribuire. Questa tesi affronta il problema attraverso due linee di indagine: come progettare interfacce che permettano agli esperti di dominio senza competenze tecniche di curare dati RDF, garantendo tracciabilità delle modifiche e della provenance; e come consentire ai tecnici di configurare ambienti di curatela per domini specializzati.

Due casi di studio hanno guidato lo sviluppo. OpenCitations Meta, con 124 milioni di entità bibliografiche, ha validato l'esistenza di barriere su larga scala; ParaText ha fornito un ambiente per test con utenti reali. Ne emergono cinque requisiti: gestione della provenance, tracciamento delle modifiche, usabilità per esperti e configuratori, flessibilità nella personalizzazione e integrazione con collezioni RDF esistenti.

HERITRACE risponde a queste domande di ricerca con un framework che utilizza l'OpenCitations Data Model per la provenance e il tracciamento delle modifiche, e la Time Agnostic Library per ricostruire gli stati passati delle entità. I tecnici configurano tipi di entità, vincoli e presentazione usando SHACL e YAML, linguaggi standard già consolidati; il framework genera automaticamente le interfacce. Gli utenti possono creare, modificare, eliminare e unire entità, ripristinare versioni precedenti, senza conoscere l'infrastruttura RDF.

La valutazione con 9 utenti e 10 tecnici ha combinato misure quantitative e qualitative. Gli utenti hanno completato i compiti con successo tra il 67% e il 100%, con un punteggio SUS di 78,9 (sopra la media). I tecnici hanno raggiunto il 90% di successo con SUS di 83,8 (eccellente).

Infine, poiché molte istituzioni gestiscono collezioni in database relazionali, la ricerca esplora l'estensione di RML per trasformazioni inverse, permettendo di adottare i principi FAIR senza modificare l'infrastruttura esistente.

Contents

Use of generative AI

I did not use generative AI assistance tools during the research/writing process of my thesis, except for mere language assistance.

The text, code, and images in this thesis are my own and generative AI has only been used in accordance with the KU Leuven and University of Bologna guidelines. I have reviewed and edited the content as needed and I take full responsibility for the content of the thesis.

Chapter 1

Introduction

Cultural heritage digitization has transformed how galleries, libraries, archives, and museums (GLAM) manage and disseminate their collections. Semantic Web technologies, particularly Resource Description Framework (RDF) (Manola & Miller, 2004) and Linked Open Data (LOD) principles (Bizer, Heath, & Berners-Lee, 2009), have enabled institutions to represent metadata in machine-readable formats that support the FAIR principles: findability, accessibility, interoperability, and reusability (Wilkinson et al., 2016). These principles address the challenge of ensuring cultural heritage data remains discoverable and reusable across institutional boundaries, maximizing the value of digitization investments through long-term preservation and cross-collection integration.

Numerous national libraries have adopted LOD for their collections. The Österreichische Nationalbibliothek provides access to over 1.8 million objects (Petz, 2023; Rachinger, 2008), the Bibliothèque nationale de France aggregates cultural heritage data spanning medieval manuscripts to contemporary publications (da Silveira, da Silva, Zilio, & Cordeiro, 2020), and the Library of Congress publishes authority data that serves as reference points for institutions worldwide (Laurence, 2013). Similar initiatives have been undertaken by the Biblioteca Nacional de España (Wulff, 2024), the Deutsche National Bibliothek (Hannemann & Kett, 2010), the National Library of the Netherlands (Beek, Hoekstra, Maas, Meroño-Peñuela, & Leemans, 2014), the British National Bibliography (Corine, Neil, Luca, & Pierre-Yves, 2017), the Biblioteca Virtual Miguel de Cervantes (Candela, Escobar, Carrasco, & Marco-Such, 2018), and the Bibliothèque nationale du Luxembourg (Kremer, 2021).

Beyond individual institutions, some countries have developed LOD infrastructure at the national level. Finland provides a notable example (Hyvönen, 2020): Finna.fi (National Library of Finland, 2026), the national search service, aggregates content from over 450 museums, libraries, archives, and research institutions with

over 19 million records[1], integrating with the Finto ontology service for semantic enrichment (Suominen et al., 2015). The Semantic Computing Research Group at Aalto University and the Helsinki Centre for Digital Humanities at the University of Helsinki have developed "Sampo" semantic portals (CultureSampo, BookSampo, WarSampo, LetterSampo) built on RDF and SPARQL (Hyvönen, 2023), covering domains from literary fiction to war history and biographical data. In Italy, ARCo (Carriero et al., 2019), developed through collaboration between the Ministry of Culture and the National Research Council, transforms the General Catalogue into a knowledge graph of 820 thousand cultural entities. At the European scale, Europeana aggregates and semantically enriches cultural heritage data from institutions across member countries (Isaac & Haslhofer, 2013).

However, this technological transformation has created a fundamental tension. While Semantic Web technologies enable FAIR data principles, they require technical expertise in RDF modeling, querying with the SPARQL Protocol and RDF Query Language (SPARQL) (W3C, 2013c), and designing ontologies with the Web Ontology Language (OWL) (W3C, 2012a) that typically lies outside traditional curatorial competencies. Two cases from the University of Bologna illustrate this dichotomy: the FICLIT Digital Library, which provides access to the collections of the Department of Classical Philology and Italian Studies (ADLab - Laboratorio Analogico Digitale & /DH.arc - Digital Humanities Advanced Research Centre, 2022), employs OmekaS (Salarelli, 2016) to offer user-friendly interfaces for curators but lacks complete provenance documentation, prevents tracking changes over time, cannot represent complex semantic relationships in RDF, and does not support importing existing RDF collections. OpenCitations, an infrastructure publishing open scholarly citation data and bibliographic metadata (Peroni & Shotton, 2020), implements complete RDF representation with integrated provenance and change-tracking but requires technical expertise for data quality management, excluding domain experts such as librarians who could identify errors in bibliographic metadata.

These cases illustrate a recurring tension in semantic data curation: data quality depends on domain expertise for entity disambiguation, relationship validation, and metadata enrichment, yet the systems that manage this data require technical skills that domain experts typically lack. This challenge motivates two research questions addressing distinct operational phases:

> **RQ1**: How can we design interfaces that enable domain experts to curate RDF data without requiring technical expertise, while maintaining

[1]Data retrieved on February 7, 2026 by querying the Finna API (https://api.finna.fi).

provenance documentation and change tracking?

RQ2: How can technical staff perform one-time system configuration to adapt the curation environment to specialized domains, ensuring integration with existing RDF collections and flexible customization?

RQ1 addresses the continuous curation workflow where domain experts interact with semantic data daily. RQ2 addresses the initial configuration phase, performed once by technical staff, that prepares the system for domain-specific use.

This thesis presents HERITRACE (Heritage Enhanced Repository Interface for Tracing, Research, Archival Curation, and Engagement) (Massari & Peroni, 2025a), a web-based system that enables domain experts to curate semantic data through user-friendly interfaces while automatically maintaining complete provenance documentation and change tracking, to address these research questions.

The research follows a Design Science Research methodology (Hevner, March, Park, & Ram, 2004), where the construction and evaluation of an artifact (HERITRACE) constitutes the primary contribution. Two case studies serve distinct methodological roles. OpenCitations Meta (Massari, Mariani, Heibi, Peroni, & Shotton, 2024), where the author participates as a contributor, provides independent validation that the identified barriers exist at scale in production systems. The ParaText Bibliographical Database was developed as an application case for HERITRACE, serving as a testbed for guerrilla testing (Nielsen, 1993) and iterative refinement throughout the development process. This distinction is methodologically significant: OpenCitations Meta demonstrates the problem exists independently, while ParaText enables solution validation through guerrilla testing in a real-world scholarly context.

The rest of this thesis is organized as follows. Chapter 2 examines existing semantic data management systems for cultural heritage, evaluating their capabilities for user-friendliness, provenance management, change tracking, and direct RDF import. The comparative analysis reveals that no existing system simultaneously addresses all requirements, establishing the need for an integrated solution.

Chapter 3 analyzes the two aforementioned case studies, OpenCitations Meta and the ParaText Bibliographical Database, documenting the barriers that prevent domain experts from participating in semantic data curation. The chapter identifies five convergent requirements that inform the design of subsequent technical solutions.

Chapter 4 examines methods for representing provenance and change tracking in RDF. The comparative analysis evaluates approaches including named graphs,

RDF-star, PROV-O, and specialized domain OWL ontologies, concluding that the OpenCitations Data Model emerges as the optimal solution by integrating PROV-O, the *de facto* standard for provenance documentation (responsible agents, primary sources) with snapshot-based versioning.

Chapter 5 addresses temporal querying methodologies for reconstructing entity states across time. The chapter presents the Time-Agnostic Library, which implements version materialization, single-version queries, and cross-version queries over OCDM datasets, enabling the version restoration requirement identified in the use cases.

Chapter 6 presents HERITRACE as an integrated framework that synthesizes these technical foundations into a system where domain experts interact with semantic data through form-based interfaces while the system handles RDF operations, provenance documentation, and change tracking transparently.

Chapter 7 presents an empirical evaluation of HERITRACE through a mixed-methods study involving domain experts and technical staff. The evaluation combines quantitative performance measures with grounded theory analysis of user experiences to assess whether the system addresses both research questions.

Chapter 8 explores preliminary experimentation on extending HERITRACE's applicability beyond RDF-native institutions through an algorithm for inverting RML mappings, enabling bidirectional transformation between relational databases and RDF graphs.

Chapter 9 synthesizes the evaluation findings, discusses limitations, and outlines future research directions.

Chapter 2

Semantic management systems: recent developments

The widespread adoption of Semantic Web technologies in cultural heritage institutions has generated the need for user-friendly data management tools. Several solutions address this requirement by providing graphical interfaces that enable domain experts to work with RDF collections without requiring technical expertise in SPARQL querying or OWL ontology engineering.

This chapter presents a comparative evaluation of existing semantic data management systems, building upon and extending the discussion by Massari and Peroni (2025a). The examined systems include OmekaS (Salarelli, 2016), Semantic MediaWiki (Krötzsch, Vrandečić, & Völkel, 2006), Research Space (Oldman & Tanase, 2018), CLEF (Daquino, Wigham, Daga, Giagnolini, & Tomasi, 2023; Giacomini, Daquino, Tomasi, & Fintoni, 2025), and Wikibase (Diefenbach, Wilde, & Alipio, 2021). The evaluation criteria draw inspiration from those originally proposed for assessing CLEF (Daquino et al., 2023). The original criterion referred to workflows for data management, with data versioning provided as an example (Giacomini et al., 2025), but without explicitly defining what such workflows entailed. This has been refined to change tracking to more precisely refer to mechanisms for recording modifications to data entities and enabling restoration of previous versions. The original reusability criterion has been refined and made more stringent as direct RDF import capability, specifically evaluating the ability to integrate pre-existing RDF collections without requiring data transformation. Moreover, one system identified in the original CLEF evaluation, LED (Adamou, Brown, Barlow, Allocca, & d'Aquin, 2019), has been excluded from this comparison: it represented an ad-hoc implementation not adaptable to different data models; additionally, the project is no longer maintained and all pages that previously linked to it now redirect to an

archival notice[1]. Table 2.1 summarizes the assessment across these criteria.

Table 2.1: Comparison of data management system features for the GLAM sector.

System	User-friendly	Admin-friendly	Prov. mgmt.	Change track.	Direct RDF import
OmekaS	✓	✓			
Semantic MediaWiki	✓	✓	✓	✓	
Research Space	✓		✓		✓
CLEF	✓	✓	✓	✓	
Wikibase	✓	✓	✓	✓	

OmekaS, recognized for its user-friendly interface, primarily serves museums and educational institutions with its intuitive web-publishing platform. OmekaS does not track provenance or maintain change histories. To import pre-existing data in bulk, users must rely on the CSV Import plugin (Berthereau, 2024), which requires restructuring the original data to fit its specific format with mandatory field names. This requirement for data formatting reduces the platform's direct RDF import capability.

Semantic MediaWiki enhances the MediaWiki platform by integrating semantic capabilities. The widespread adoption of MediaWiki through Wikipedia has made its interface familiar to a broad audience, providing an approachable entry point for end-users accustomed to Wikipedia's editing environment, as well as for configurators and system administrators who can leverage existing MediaWiki expertise. Semantic MediaWiki inherits MediaWiki's native page revision history, which records who edited each page and when, providing basic provenance documentation at the page level. However, this wiki-level revision tracking does not extend to semantic change tracking: monitoring modifications to individual RDF properties requires external plugins such as Semantic Watchlist (WikiTeq, 2023), which stores change records in a relational database rather than in RDF format. This storage approach creates a discrepancy in adherence to FAIR principles: while the semantic data itself is published as RDF, the change tracking metadata remains in a non-FAIR relational format, limiting its interoperability and reusability. Semantic MediaWiki initially focused solely on importing OWL ontologies for direct RDF import. The RDFIO extension (Lampa et al., 2017) broadened this support by implementing a PHP/MySQL-based triplestore through the ARC2 library (Nowack, 2024), storing imported RDF triples in relational database tables rather than in a native RDF store. The extension supports importing RDF data in Turtle format via web form and N-Triples format via command-line tools. However, it only supports

[1] https://research-archive.stem.open.ac.uk/led/

the import of triples, not quadruples, which means it cannot handle named graphs.

ResearchSpace, tailored for the academic and research community, provides diverse data visualization options such as graphs and temporal maps for end-users. For administrators, the platform requires a solid understanding of HTML, handlebars and other ResearchSpace-specific components for creating templates, which may be cumbersome even for those with technical expertise. ResearchSpace records when modifications occur and which agents are responsible for them. While this approach documents who made changes and when, it does not constitute a change-tracking system in the sense adopted in this analysis: ResearchSpace does not record what modifications were made to data entities, nor does it maintain version histories that would enable restoration of previous states. The platform allows uploading RDF data directly, which is advantageous for projects involving such formats. However, after data upload, an administrator's intervention is required to adapt the interface appropriately to display the items correctly, involving a significant level of programming complexity.

CLEF is designed to manage complex digital libraries, archives, and research data, particularly in the humanities. It offers an administrator-friendly interface and focuses on user-friendliness for end-users, making it suitable for a wide range of audiences within its domain. Provenance management is implemented through named graphs, and change tracking capabilities include synchronization with GitHub (Giacomini et al., 2025), though the system lacks a graphical interface to restore previous versions. CLEF is not designed to upload and manage pre-existing RDF data as-is: the software is structured to add items one by one from scratch directly through the user interface. This approach, while potentially beneficial for building new databases, limits the platform's ability to integrate and manage existing large-scale datasets. Furthermore, even though CLEF does not impose a specific data model, it organizes data in a format akin to nanopublications for managing provenance. This structure means that if a pre-existing triple store is connected, the system is not ready to explore the data without a prior reorganization to make it compatible with CLEF's framework.

Wikibase, the software powering Wikidata, combines user-friendliness with administrative ease of use through an interface familiar to Wikipedia users. Like Semantic MediaWiki, Wikibase inherits MediaWiki's revision history system, which provides both provenance documentation and change tracking at the page level. However, Wikibase presents a fundamental limitation in integrating pre-existing RDF data. The system only allows elements from its internal Property namespace (P) to be used in the predicate position, whereas standard RDF allows arbitrary

IRIs to be reused in any triple position. This restriction prevents the direct use of predicates from external OWL ontologies. Consequently, any term from an imported ontology must be redefined as a Wikibase entity with proprietary identifiers (QIDs for items, PIDs for properties), and mappings must then be established to indicate equivalence with the original external vocabulary terms. While tools such as O2WB (Dobriy & Polleres, 2023) can automate this conversion process, the requirement to transform and redefine all OWL ontology terms rather than reuse them directly renders direct RDF import a complex undertaking.

As summarized in Table 2.1, no existing system simultaneously addresses user-friendliness, admin-friendliness, provenance management, change tracking, and direct RDF import capability.

Chapter 3

Use cases: data curation challenges in cultural heritage

This chapter examines two case studies that document challenges encountered across different cultural heritage domains. The analysis provides empirical evidence for the systematic barriers in semantic data curation and validates the requirements that inform the technical solutions explored in subsequent chapters.

Two case studies demonstrate convergent barriers that affect semantic data curation. The first examines OpenCitations Meta, a large-scale bibliographic metadata index of scholarly publications. The second analyzes the ParaText Bibliographical Database in Classical Philology, where specialized scholarly terminology and contested interpretations create unique semantic representation challenges. Both cases confirm a recurring pattern: Semantic Web technologies enable complex data representation and computational analysis, while simultaneously creating access barriers for domain experts without specialized technical expertise.

3.1 OpenCitations Meta: bibliographic metadata curation at scale

OpenCitations Meta represents a large-scale bibliographic metadata database that exemplifies both the potential and the challenges of semantic data curation in scholarly publishing contexts. As a repository incorporating metadata from Crossref (Hendricks, Tkaczyk, Lin, & Feeney, 2020), DataCite (Brase, 2009), the National Institutes of Health Open Citation Collection (ICite, Hutchins, & Santangelo, 2022), OpenAlex (Priem, Piwowar, & Orr, 2022), OpenAIRE (Manghi, Manola, Horstmann, & Peters, 2010), and Japan Link Center (Moretti et al., 2024), OpenCi-

tations Meta demonstrates an approach to automated data processing while simultaneously revealing the persistent barriers that prevent effective human-mediated data curation at scale.

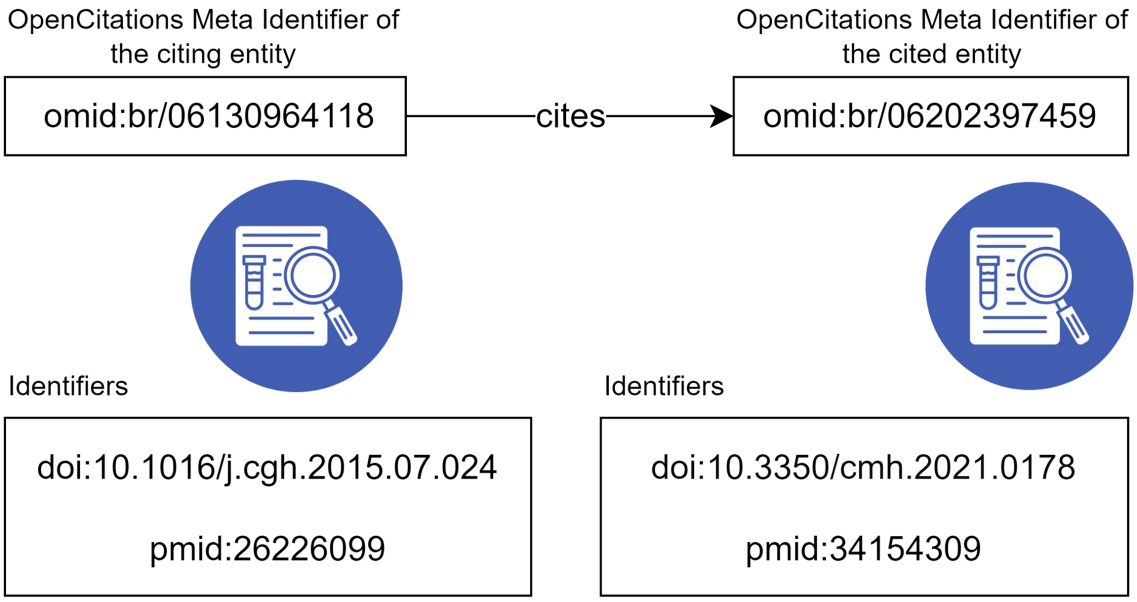
The project addresses two fundamental challenges in bibliographic metadata management: citation disambiguation across multiple identifier schemes, and the inclusion of publications lacking persistent identifiers. Through the introduction of OpenCitations Meta Identifiers (OMIDs), the system enables unified representation of citations as OMID-to-OMID relationships, solving disambiguation problems that arise when the same publication appears with different identifiers across multiple sources (Figure 3.1).

The technical architecture of OpenCitations Meta implements automated curation through a two-phase process. The first phase concerns data cleaning, including deduplication based strictly on identifiers, error correction following documented patterns, and metadata enrichment from existing database content. The deduplication approach favors precision over recall: entities are merged only when they share identical persistent identifiers, avoiding heuristic-based matching that might introduce false positives.

The automated error correction system addresses common metadata inconsistencies through pattern recognition and standardization. Volume and issue number corrections follow specific rules to handle prefix errors, suffix errors, encoding problems, and misclassified fields. For example, the system recognizes patterns like "Vol. 35 N° spécial 1" and correctly separates volume and issue information. Similarly, malformed identifiers are corrected based on their respective schemes, with syntactic validation for ISBNs, ISSNs, and ORCIDs, and semantic verification for ORCIDs and DOIs through API calls to verify actual existence.

The second phase converts the curated data into RDF format following the OCDM (Daquino et al., 2020), ensuring semantic interoperability and enabling complex queries through SPARQL endpoints. This conversion process implements the provenance and change tracking mechanisms described in Section 4.5, utilizing the OCDM framework to record data modifications with temporal boundaries and agent attribution.

However, OpenCitations Meta reveals significant limitations in addressing data quality challenges that require human expertise. The system's reliance on automated processes, while enabling scale, cannot address semantic errors or contextual inconsistencies that require domain knowledge. For instance, while the system can detect syntactically invalid dates or misplaced volume information, it cannot identify factually incorrect publication dates or resolve ambiguous author attributions



Without the OpenCitations Meta Identifier	
Citing ID	**Cited ID**
doi:10.1016/j.cgh.2015.07.024 pmid:26226099	doi:10.3350/cmh.2021.0178 pmid:34154309
doi:10.1016/j.cgh.2015.07.024 pmid:26226099	pmid:34154309 doi:10.3350/cmh.2021.0178

With the OpenCitations Meta Identifier	
Citing ID	**Cited ID**
omid:br/06130964118	omid:br/06202397459

Figure 3.1: Citation disambiguation through OpenCitations Meta Identifiers. When a document is described by multiple identifiers (e.g., a DOI from Crossref and a PMID from PubMed), citations involving it may be described in multiple ways, creating ambiguity and deduplication problems. The OMID acts as a proxy between different external identifiers, enabling unified citation representation.

without additional identifier information.

The deduplication process illustrates this tension between automated efficiency and accuracy requirements. The decision tree implemented in OpenCitations Meta handles six distinct scenarios based on identifier availability and conflicts (Fig-

ure 3.2), but manual intervention becomes necessary when external identifiers connect multiple entities that should remain distinct or when conflicting information cannot be resolved automatically. For example, when two initially distinct entities in the database are later linked by a shared identifier through new data ingestion, the automated approach would merge them into a single entity. However, this merge may introduce errors rather than correct them: the shared identifier itself could be erroneous, assigned incorrectly to one or both records. The system cannot automatically determine whether a shared identifier indicates true entity equivalence or metadata corruption. A concrete example illustrates this problem: two unrelated articles published in the Scandinavian Journal of Infectious Diseases share the identical DOI 10.1080/00365540410020884, despite having different authors, publication years, and entirely distinct subject matter: one addresses respiratory infections in children (Maltezou et al., 2004), while the other concerns pediatric vaccination schedules (Nilsson et al., 2005). An automated deduplication system relying on DOI matching would incorrectly merge these records into a single entity, corrupting both bibliographic entries. Only human review can recognize that the shared DOI represents a data entry error rather than entity equivalence. Beyond identifier errors, author disambiguation represents the most problematic case, where ORCIDs become fundamental for accurate identification. Without ORCIDs, the risk of homonymy remains substantial, a challenge extensively documented in the disambiguation literature that employs various technical approaches beyond the scope of this analysis (Cappelli, Colavizza, & Peroni, 2025).

More fundamentally, OpenCitations Meta's quality depends on the quality of its input sources. Crossref (Hendricks et al., 2020), as a primary source, does not systematically verify metadata provided by publishers, resulting in preserved errors throughout the system. Examples include publications with impossible future publication dates or inconsistent author name formatting. While some errors can be addressed through pattern recognition and standardization rules, others require either web crawling or manual intervention by domain experts.

A systematic analysis of invalid DOIs in Crossref data exemplifies the magnitude of source data quality issues (Cioffi et al., 2022). The study identified over 1.2 million invalid DOI-to-DOI citations. The investigation revealed that only a few publishers were responsible for the majority of invalid citations, with Ovid Technologies alone accounting for over 370,000 invalid outgoing citations. Through pattern recognition and regular expression-based cleaning methods, approximately 141,809 citations (11.6% of invalid citations) could be automatically corrected, primarily addressing suffix-type errors where extraneous characters appeared after valid DOI

strings. However, the remaining 88.4% of invalid citations could not be resolved through automated processes, requiring either publisher intervention or acceptance of permanent data loss.

The scale of the challenge becomes apparent when considering the quantitative scope of OpenCitations Meta. At its June 2025 release (OpenCitations, 2025), the database contained over 124 million bibliographic entities with more than 376 million authors, over 1 million publication venues, approximately 2.8 million editor roles and 104 million publisher roles (each counted per record rather than as distinct individuals or organizations). This scale makes manual verification impractical for any significant portion of the dataset, yet automated approaches cannot address all quality issues.

The project's commitment to openness and Semantic Web standards provides significant advantages for interoperability and reuse, with data available under CC0 license through SPARQL endpoints, REST APIs, and complete data dumps. However, these technical capabilities do not address the fundamental challenge of enabling domain experts to participate effectively in data curation processes. The current system offers no mechanism for librarians, researchers, or other qualified users to contribute corrections or improvements to the metadata.

3.2 ParaText bibliographical database in Classical Philology

While OpenCitations Meta demonstrates usability barriers at scale in bibliographic metadata management, the ParaText Bibliographical Database illustrates how specialized scholarly domains present compounded challenges when semantic data needs intersect with domain-specific complexity. ParaText was developed as an application case for HERITRACE, providing the environment for guerrilla testing with domain experts throughout the development process.

ParaText represents a two-year research project funded through the European Union's Next Generation EU program and the Italian Ministry of University and Research. The project investigates relationships between literary texts and exegetical paratexts in manuscript transmission of ancient Greek poetry, producing a Repertoire of Hypertext Transcriptions, a Lexicon of Ancient Greek Exegesis, and a Bibliographical Database[1]. In this context, exegetical paratexts refer to interpretive materials such as commentaries, annotations, and glosses that accompany primary literary texts in manuscript traditions, providing explanations and interpretations

[1] https://paratext.unipv.it/

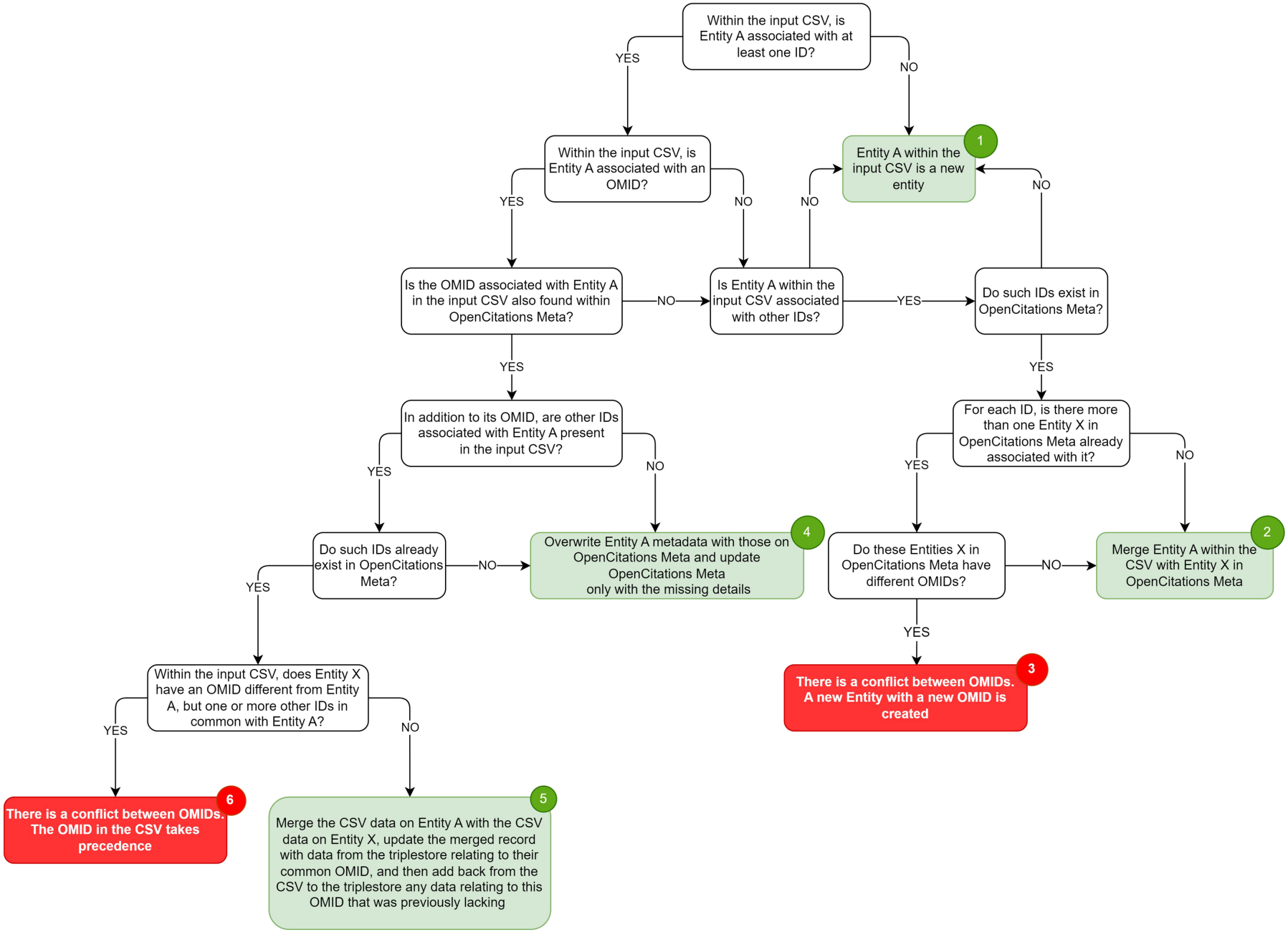


Figure 3.2: Deduplication decision tree showing the six possible outcomes when processing entities with various identifier configurations. The process prioritizes existing database information while creating new OMIDs for entities lacking identifiers or experiencing conflicts that cannot be resolved automatically.

of the original works. The Bibliographical Database adopts RDF for semantic data representation to enable interoperability with other Linked Open Data resources in cultural heritage, support FAIR data principles, and express complex relationships between texts, commentaries, and manuscript traditions.

Classical Philology presents an extreme case where ongoing data curation demands particularly deep subject matter knowledge. The field's terminological precision, hierarchical classifications, and contested scholarly interpretations create requirements for continuous expert intervention in data quality management. Philologists must make informed decisions about terminological distinctions specific to ancient Greek exegesis, recognize when contested interpretations warrant multiple keyword assignments, and identify when metadata must evolve to reflect emerging scholarly consensus. This expertise proves necessary for maintaining data quality in ways that technical staff cannot replicate.

Classical Philology's terminological specialization demonstrates domain-specific customization demands. The Année Philologique, founded by Jules Marouzeau in 1926 (Hilbold, 2018), serves as the primary bibliographical reference for classical studies with over 900,000 records. While this generalist database serves broad disciplinary needs, specialized research in ancient Greek exegesis requires more granular terminological distinctions.

The subject "Scholia, commentaries" in Année Philologique encompasses 521 bibliographic records representing a broad category. Scholia are marginal or interlinear annotations found in manuscript copies of ancient texts, typically providing explanations, grammatical notes, or interpretive commentary. ParaText requires far more specific terminology: while both databases use "scholia", ParaText distinguishes it from "hypomnema" (systematic commentary organized as continuous prose) and employs granular classifications including "Dscholia", "VMK-scholia", "scholia exegetica", "h-scholia", and "Ge-scholia" for different Homeric scholiastic classes based on manuscript traditions and content characteristics. Temporal categories like "scholia vetera" denote ancient scholia, while typological specifications such as "short scholia", "frame-scholia", and "scholia à recueil" indicate length and placement in manuscripts.

Among approximately 400 records pertaining to ancient Greek poetic text exegesis in ParaText Bibliographical Database, 179 employ the "scholia" keyword. Considering that among Année Philologique's 521 "Scholia, commentaries" records, only a fraction pertain to ancient Greek poetry, this terminological precision enables research depth that generalist categorizations cannot provide. However, maintaining such granular distinctions requires continuous data curation by domain experts

who can correctly assign keywords to bibliographic records.

ParaText Bibliographical Database organizes this complexity through four hierarchical macro-categories: "exegetical cultures and activities", "exegetical products", "exegetical signs and layout", and "ancient tradition". Each macro-category contains specialized terms, and when any term is entered as a keyword, the corresponding macro-category must also be included, enabling research at both general and detailed levels while maintaining controlled vocabulary standards.

The scholarly tradition of questioning transmitted texts directly influences bibliographical databases in Classical Philology. This critical approach manifests in specialized record types like "Review Article" (modeled as `fabio:ReviewArticle` from the FaBiO ontology (Peroni & Shotton, 2012)) alongside the standard "Review" category.

The case study of Chiara Martis's 2013 article "L'enigma del PLouvre inv. 7733 verso: l'epigramma dell'ostrica" illustrates how semantic precision impacts research effectiveness and creates editorial challenges (Martis, 2013). Comparison of abstracts across three sources reveals terminology selection challenges that affect discoverability. The journal abstract refers to "explanatory commentary" and acknowledges an "elegy or epigram" debate. Elegies and epigrams represent distinct poetic genres in ancient Greek literature: elegies typically consist of longer poems, while epigrams are shorter, often epigrammatic verses originally associated with inscriptions. Année Philologique refers generically to "distici elegiaci" and "commento". ParaText Bibliographical Database initially used "commented edition", the most precise term identifying the exegetical product as text-plus-paratext unity rather than paratext alone, but referred only to "epigram".

ParaText Bibliographical Database's initial choice to use only "epigram" in the abstract and as keyword concealed ongoing scholarly debate and limited discoverability for researchers studying paratexts related to elegy. This necessitated scholarly discussion, ultimately leading to abstract revision and introduction of the "elegy" keyword to maintain research discoverability while preserving interpretive debate. The resolution required recognizing the discoverability problem, modifying metadata records appropriately, and balancing precision with discoverability.

This example illustrates the philological knowledge necessary for data curation: recognizing that the "elegy or epigram" scholarly debate necessitated adding the "elegy" keyword, or deciding when to use "scholia" versus "hypomnema", or assigning keywords from hierarchical macro-categories all require continuous expert intervention. In traditional RDF-based systems, implementing these curatorial decisions requires technical intermediaries who can perform RDF data manipulation.

The technical complexity of conventional RDF management tools prevents philologists from performing these curation operations directly. Such tasks typically require manipulating RDF triples through SPARQL queries or interfaces that demand technical skills in query syntax, data structures, and triplestore interaction that remain beyond the capabilities of most humanities scholars, whose expertise lies in textual analysis, manuscript traditions, and scholarly interpretation rather than Semantic Web technologies.

This usability barrier traditionally creates a dependency on technical intermediaries for routine curation tasks. Domain experts must communicate desired modifications to technical staff, who then translate curatorial decisions into SPARQL operations. This workflow introduces delays, increases miscommunication risks, and creates dependencies that limit curation sustainability. The problem proves particularly acute for specialized domains like Classical Philology, where the frequency of required curatorial interventions exceeds typical technical support capacity. ParaText's role as an application case for HERITRACE enabled validation that user-friendly interfaces can bridge this gap while maintaining semantic integrity.

3.3 Synthesis: requirements for semantic data curation systems

The analysis of OpenCitations Meta and ParaText Bibliographical Database confirms convergent requirements that semantic data curation systems must address to enable effective domain expert participation while maintaining semantic integrity.

Provenance management emerges as the first requirement. OpenCitations Meta processes metadata from six external sources (Section 3.1) where conflicts require tracking which source provided specific information. ParaText requires attributing scholarly interpretations to specific philologists who make curatorial decisions about contested terminology. A curation system must therefore record who made changes, when modifications occurred, and what primary sources informed curatorial decisions, supporting institutional accountability, scholarly attribution, and quality assessment.

The second requirement, **change tracking and version restoration**, reflects a shared need across both case studies. The latest version of metadata does not always represent the most accurate understanding: OpenCitations Meta encounters situations where automated corrections introduce errors that require reverting to previous states, while in ParaText, scholarly consensus evolves and keyword modifications may later prove incorrect when new manuscript evidence emerges. Complete

version histories that enable temporal reconstruction of entity states and restoration to previous versions are therefore necessary.

Both case studies converge most strongly on **domain expert usability**, the third and most pervasive requirement. OpenCitations Meta cannot integrate corrections from librarians and researchers who identify errors in bibliographic metadata but lack RDF expertise. ParaText philologists possess the specialized knowledge necessary for terminological distinctions ("scholia" versus "hypomnema", hierarchical macro-categories, contested interpretations) but cannot manipulate RDF triples directly. A curation system must abstract RDF complexity behind interfaces that correspond to domain experts' existing mental models, enabling metadata creation, modification, and relationship management without requiring SPARQL query construction or RDF syntax knowledge.

ParaText reveals a fourth requirement: **flexible customization for specialized domains**. The granular terminological distinctions and hierarchical macro-categories required by ParaText (Section 3.2) exceed what generalist bibliographic systems can accommodate. Customization to specialized terminologies, controlled vocabularies, and domain-specific validation rules must be achievable without extensive programming knowledge or proprietary configuration languages.

Finally, **integration with existing RDF collections** constitutes the fifth requirement. Cultural heritage institutions have invested substantial resources in creating RDF datasets that comply with Semantic Web standards and FAIR principles. OpenCitations Meta maintains its entire bibliographic collection in RDF format, available through SPARQL endpoints. Institutions cannot abandon these investments to adopt new systems: solutions must apply curation capabilities to established data stored in triplestores and other semantic repositories without requiring migration or format conversion.

The comparative analysis in Chapter 2 demonstrates that no existing semantic data management system simultaneously addresses all five requirements. This systematic gap motivates the development of HERITRACE as an integrated framework. The following chapters examine the technical mechanisms necessary to satisfy these requirements. Chapter 4 reviews methods for representing provenance and change tracking in RDF, evaluating their capabilities for documenting modifications and maintaining version histories. Chapter 5 addresses temporal query mechanisms that enable reconstruction of entity states across time. These technical foundations inform the design of HERITRACE (Chapter 6), which integrates provenance management, change tracking, and domain expert usability into a unified framework for semantic data curation.

Chapter 4

Provenance and change tracking representation in RDF

Documenting provenance and tracking changes over time are practices that predate digital technologies and the Semantic Web. Several disciplines involved in cultural heritage care have independently developed methodologies that converge on the same requirement: recording not only the current state of knowledge but also who contributed it, from what sources, and through what reasoning processes. This chapter first examines these disciplinary perspectives before turning to the technical challenges of representing provenance and change tracking information in RDF.

4.1 Provenance and change tracking across cultural heritage disciplines

In museology, the concept of "object biography" (Gosden & Marshall, 1999) describes how cultural objects accumulate meanings, attributions, and interpretations throughout their social lives: from creation, through changes in ownership and conservation interventions, to musealisation. Each phase of an object's biography involves different agents who attribute different values and meanings to the same artefact, and the documentation of these successive layers of interpretation constitutes the object's provenance record. The International Council of Museums (ICOM), in section 2.3 of its Code of Ethics (International Council of Museums, 2017), requires its member museums to document the complete provenance of acquired objects, including full ownership history from discovery or production. This requirement serves not only scholarly purposes but also the prevention of illicit trafficking in cultural objects (Immonen, Bonnie, Dixon, Tervahauta, & Thomas, 2020): gaps in provenance

documentation can conceal illicit transfers and facilitate forgeries persisting undetected in museum collections for decades (La Niece, 2009), making the traceability of custodial chains a matter of both scholarly integrity and legal compliance.

Conservation theory has undergone a parallel shift that reinforces the same provenance requirements. The classical paradigm, articulated by Cesare Brandi in his *Teoria del restauro* (Brandi, 2000), focused on the dual identity of the artwork (aesthetic instance and historical instance) and on the restoration of its potential unity without producing falsification or erasing historical traces. This object-centered approach has been progressively expanded: Roland May proposed a shift toward communication-centered conservation (May, 2009), understanding heritage as a social phenomenon (*fait sociétal*) constituted through ongoing processes of meaning attribution. In this framework, conservation is not the preservation of material state but the maintenance of communicative capacity across time. The debate surrounding the Saint-Denis cathedral spire in Paris exemplifies this dynamic (Lejeune, 2021). The spire was dismantled in 1846, and scholars remain divided on whether it should be reconstructed: some argue that the absence has become part of the monument's biography, while others advocate reconstruction for symbolic, identity, or educational reasons. Multiple stakeholders (administration, academic experts, local residents, citizen associations) hold competing value frameworks, and the decision-making process itself requires documentation: not only the technical interventions but also the motivations, debates, and criteria that shaped each choice. If conservation is a form of observing and documenting how meaning has been attributed to heritage over time, then provenance records must capture not only factual changes but also the reasoning processes and competing interpretations that informed each decision.

Historical research methods arrive at the same requirement from a different direction. The structured interview method developed by Kurkowska-Budzan, Soroko, and Stasiak (2022) for historians working with witnesses to historical events establishes rigorous procedures for "evoking the historical source": the researcher's prepared and scientifically rigorous participation in the creation by a respondent of recollective material that can be subjected to historical analysis. The research contract that opens each interview establishes transparent rules about data collection, storage, and attribution: the historian documents who the respondent is, what relationship they have to the events, under what circumstances the interview took place, and how the recorded material may be used. This procedural transparency ensures that subsequent researchers can evaluate not only the content of the testimony but also the conditions of its production. The method also addresses the psychological

phenomenon of source monitoring: research in cognitive psychology shows that people progressively lose track of the origin of their information over time (Johnson, Hashtroudi, & Lindsay, 1993), and may no longer distinguish whether they know a fact from personal experience, from another witness's account, or from media coverage. The structured interview method addresses this through direct questions about information sources, asking respondents to distinguish between what they personally witnessed and what they learned from other sources. This distinction between "I remember" (episodic memory, grounded in personal experience) and "I know" (semantic memory, potentially acquired from secondary sources) constitutes a form of provenance attribution applied to human testimony. The method further links personal chronology with objective chronology, connecting the respondent's life events to verifiable external dates to create a temporal framework that allows subsequent verification against other sources.

These three disciplines have independently developed provenance practices that share a common structure: documenting not only current knowledge but also the agents responsible for it, the sources from which it derives, the temporal context in which it was produced, and the reasoning processes that informed decisions. However, representing such provenance and change tracking information in RDF presents significant technical challenges. The foundational Semantic Web technologies (RDF, SPARQL, and OWL) did not initially provide mechanisms to annotate statements with contextual metadata. This limitation has led to numerous metadata representation models, none of which have achieved widespread acceptance as standards. The remainder of this chapter offers a detailed examination of existing models and approaches for representing provenance and tracking changes in RDF, building upon and extending the discussion by Massari, Peroni, Tomasi, and Heibi (2025), with the objective of identifying solutions that satisfy the requirements established through empirical case study analysis.

4.2 Provenance management foundations in the Semantic Web

In 2010, the Provenance Incubator Group was established by the W3C to review the state of the art and develop a roadmap for provenance in Semantic Web technologies (Gil et al., 2010). One of the first challenges faced by the group was identifying a shared and universal definition of "provenance," a task that proved impossible given its broad and multisectoral nature. Consequently, a working definition was adopted, limited to the context of the Web: "Provenance of a resource is a record

that describes entities and processes involved in producing and delivering or otherwise influencing that resource. Provenance provides a critical foundation for assessing authenticity, enabling trust, and allowing reproducibility. Provenance assertions are a form of contextual metadata and can themselves become records with their own provenance."

According to the W3C Provenance Incubator Group, provenance can be evaluated under three broad categories: content, management, and usage. These categories encompass various dimensions that prove necessary for understanding the requirements and challenges associated with provenance representation in the Semantic Web, as summarized in Table 4.1.

Many data models, annotation frameworks, vocabularies, and OWL ontologies have been introduced to meet these requirements. In order to conduct this review, I adopted a citation-based strategy (Figure 4.1), also known as "snowballing" (Wohlin, 2014), which consists of exploding the bibliography from a seed paper (Lecy & Beatty, 2012).

The seed paper for the snowballing procedure was the survey by Sikos and Philp (2020), which examines the advantages and disadvantages of existing approaches for attaching provenance information to RDF data.

4.3 W3C standard proposals for provenance representation

To address the challenge of representing provenance in RDF, the W3C has proposed several models, notably **RDF Reification** (Manola & Miller, 2004) and **n-ary relations** (W3C, 2006). RDF Reification represents the earliest and most fundamental approach proposed by the W3C for annotating RDF statements. In RDF Reification, a statement (or triple) is converted into a resource, allowing metadata to be attached to it. This process involves creating four additional triples for each original statement, specifying the subject, predicate, object, and the statement itself.

Consider an example where RDF Reification transforms a simple RDF statement asserting the date of a manuscript. The original statement `ex:manuscript10245 ex:hasDate "15th Century"` becomes a set of reified triples. The new resource, representing the original statement, can then serve as the subject of additional provenance triples, such as those documenting the curator responsible for recording the information, as shown in Listing 1.

This methodology has a considerable disadvantage: the size of the dataset is at least quadrupled since subject, predicate, and object must be repeated to add at least

Table 4.1: Dimensions of provenance described and classified under three broad categories: content, management, and usage (Gil et al., 2010).

Category	Dimension	Description
Content	Object	The artifact that a provenance statement is about.
	Attribution	The sources or entities that contributed to creating the artifact in question.
	Process	The activities (or steps) that were carried out to generate or access the artifact at hand.
	Versioning	Records of changes to an artifact over time and what entities and processes were associated with those changes.
	Justification	Documentation recording why and how a particular decision is made.
	Entailment	Explanations showing how facts were derived from other facts.
Management	Publication	Making provenance available on the Web.
	Access	The ability to find the provenance for a particular artifact.
	Dissemination	Defining provenance distribution and access control mechanisms.
	Scale	Dealing with large amounts of provenance.
Use	Understanding	How to enable the end-user consumption of provenance.
	Interoperability	Combining provenance produced by multiple different systems.
	Comparison	Comparing artifacts through their provenance.
	Accountability	Using provenance to assign credit or blame.
	Trust	Using provenance to make trust judgments.
	Imperfections	Dealing with imperfections in provenance records.
	Debugging	Using provenance to detect bugs or failures of processes.

```
ex:manuscript10245 ex:hasDate "15th Century".
ex:triple12345 rdf:type rdf:Statement ;
    rdf:subject ex:manuscript10245 ;
    rdf:predicate ex:hasDate ;
    rdf:object "15th Century" .
ex:triple12345 dc:creator ex:curator56789 .
```

Listing 1: Representation of provenance using RDF Reification, where a triple is transformed into a statement resource to which metadata can be attached.

Figure 4.1: The snowballing procedure adopted for conducting a systematic review.

one provenance information. There is a shorthand notation, the `rdf:ID` attribute in RDF/XML, but it is not present in other serializations. Additionally, composing SPARQL queries to obtain provenance annotated through RDF Reification proves cumbersome: to identify the URI of the reification, it becomes necessary to explicitly match the entire reference triple.

For these reasons, several deprecation proposals exist for this syntax, including that by David Beckett, one of the editors of RDF in 2004 (Beckett, 2010): "There are a few RDF model parts that should be deprecated (or removed if that seems possible), in particular reification which turned out not to be widely used, understood or implemented even in the RDF 2004 update".

After RDF Reification, in 2006, the W3C published a note that suggested a new

approach to express provenance, called n-ary relations (W3C, 2006). In RDF and OWL, properties are always binary relationships between two URIs or a URI and a value. However, sometimes it proves convenient to connect a URI to more than one other URI or value, such as expressing the provenance of a certain relationship.

The n-ary relations approach achieves this by introducing a new node, a blank node, that serves as an intermediary between the subject and the object, allowing additional metadata to be attached. Using the same example, a statement such as `ex:manuscript10245 ex:hasDate "15th Century"` can be transformed into an n-ary relation by introducing a blank node to encapsulate the provenance information. Here, a blank node acts as an intermediary, linking the manuscript entity to both the date value and the provenance metadata, as illustrated in Listing 2.

```
@prefix ex: <http://example.org/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
ex:manuscript10245 ex:hasDate _:date .
_:date ex:hasDateValue "15th Century" ;
    dc:creator ex:curator56789 .
```

Listing 2: Representation of provenance using the n-ary relation approach, where a blank node encapsulates both the date value and its associated provenance metadata.

From a structural perspective, n-ary relations share similarities with RDF Reification, but there is a key difference in their approach. RDF Reification (Listing 1) converts the entire triple into an entity that can be annotated with additional metadata, whereas n-ary relations (Listing 2) introduce an intermediary node that connects the subject to both the object value and its associated metadata. This method avoids repeating all three elements of the original triple, as required in RDF Reification, and instead only introduces a new node as the object of the original statement. However, a drawback of this approach is that the blank node introduced cannot be globally dereferenced, which can limit interoperability in certain contexts.

RDF Reification and n-ary relations represent the only standard alternatives recommended by W3C to describe provenance in RDF and have considerable design flaws. For this reason, different approaches have been proposed. Among these, the concepts originally proposed as RDF-star (Gschwend & Lassila, 2022) have gained significant attention and are being standardized as part of the RDF 1.2 specification (Kellogg, Hartig, Champin, & Seaborne, 2025), which introduces triple terms and a new reification mechanism that addresses many of the limitations of traditional reification techniques.

4.4 Systematic review of provenance representation systems

Since 2005, numerous approaches have been proposed to represent provenance in RDF datasets (Sikos & Philp, 2020). These approaches can be categorized based on multiple dimensions: semantics, tuple typology (the number of elements in an RDF tuple, such as triples, quadruples, or quintuples), standard compliance, reliance on external vocabularies, blank node management, granularity, and scalability. These approaches can be broadly categorized into three main types: encapsulating provenance within RDF triples, associating provenance through RDF quadruples, and extending the RDF data model itself.

4.4.1 Encapsulating provenance within RDF triples

The **Provenance Context Entity (PaCE)** approach (Sahoo, Bodenreider, Hitzler, Sheth, & Thirunarayan, 2010) proposes a method for attaching provenance to RDF data without using standard reification or blank nodes. Instead, each element (subject, predicate, object) can be "minted" as a separate URI that includes the source information, thereby embedding provenance directly in the resource names. This approach creates a new instance representing the original entity within the context of its provenance source. The property `provenir:derives_from` from the Provenir Ontology (Sahoo & Sheth, 2009) was originally proposed to associate provenance metadata.

For example, a statement such as `ex:manuscript10245 ex:hasDate "15th Century"` can be represented using PaCE by creating a new instance that embeds the provenance source directly in the resource name, as illustrated in Listing 3.

```
@prefix provenir: <http://knoesis.wright.edu/provenir/provenir.owl#> .
ex:manuscript10245_ArchiveA a ex:Manuscript .
ex:manuscript10245_ArchiveA ex:hasDate "15th Century" .
ex:manuscript10245_ArchiveA provenir:derives_from ex:ArchiveA .
```

Listing 3: Representation of provenance using PaCE, where a new instance embeds the provenance source in the resource name.

This approach creates a new instance (`ex:manuscript10245_ArchiveA`) representing the original entity (`ex:manuscript10245`) within the context of its provenance source (`ex:ArchiveA`). The use of `provenir:derives_from` in PaCE introduces a meta-rule that extends RDF(S) semantics while maintaining compatibility with standard RDF 1.1. Specifically, when two RDF triples share the same value

for `provenir:derives_from`, they are considered to belong to the same provenance context.

Unlike RDF reification, which requires multiple joins to reconstruct triples, PaCE offers a more compact representation. However, its adoption has been hindered by significant limitations. The most restrictive is that PaCE can only indicate the primary source of an entity, without supporting any additional provenance details.

Another approach that encapsulates provenance within triples is the **singleton property** method (Nguyen, Bodenreider, & Sheth, 2014). Inspired by set theory, where a singleton set contains exactly one element, the singleton property approach creates "a unique property instance representing a newly established relationship between two existing entities in one particular context." In practice, subjects are connected to objects using these unique properties, which are linked to a generic predicate via a dedicated predicate (commonly `rdf:singletonPropertyOf`). This unique property instance then serves as the carrier for additional provenance metadata.

The singleton property approach was introduced to address the limitations of PaCE, particularly its inability to represent provenance beyond identifying a primary source. While PaCE required minting new URIs that embedded provenance information, singleton properties instead generate unique predicates for each assertion, allowing metadata to be attached without altering the identity of the entities involved. This key difference enables more flexible provenance tracking while maintaining compatibility with standard RDF structures.

For example, a statement such as `ex:manuscript10245 ex:hasDate "15th Century"` can be transformed using singleton properties, where a unique property (`ex:hasDate_1`) is introduced to encapsulate provenance information, as shown in Listing 4.

```
@prefix ex: <http://example.org/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
ex:manuscript10245 ex:hasDate_1 "15th Century".
ex:hasDate_1 rdf:singletonPropertyOf ex:hasDate ;
    dc:creator ex:curator56789 .
```

Listing 4: Representation of provenance using the singleton property approach, where a unique property instance encapsulates additional metadata.

In this example, the singleton property is explicitly linked to the generic predicate (`ex:hasDate`) and further annotated with provenance metadata (`dc:creator ex:curator56789`).

While this strategy has demonstrated advantages in terms of reducing query size and improving query execution time when compared to PaCE (Nguyen et al., 2014), it also suffers from disadvantages in scenarios where multiple predications share the same source. Additionally, singleton properties, like PaCE, rely on non-standard terms and face scalability challenges, yet they remain fully compliant with the RDF data model and SPARQL and are serializable in any RDF format.

The **Wikibase ontology**, which underlies Wikidata (Vrandečić & Krötzsch, 2014), presents another approach for encapsulating provenance within triples. In Wikibase, each statement is treated as a separate entity, commonly referred to as a statement node, designed to encapsulate the primary assertion along with its associated provenance metadata. For instance, consider the snippet from the Wikidata RDF dump for Codex Leicester (`wd:Q683814`) in Listing 5. In this example, the item "Codex Leicester" is connected to a statement via property `p:P127` ("owned by"). The statement itself, of type `wikibase:Statement` (and marked with `wikibase:BestRank`), contains the main assertion indicating that Codex Leicester is owned by Bill Gates (`wd:Q5284`) and a qualifier (`pq:P580`) that specifies the start time of this ownership.

```
@prefix wd: <http://www.wikidata.org/entity/> .
@prefix wdt: <http://www.wikidata.org/prop/direct/> .
@prefix p: <http://www.wikidata.org/prop/> .
@prefix ps: <http://www.wikidata.org/prop/statement/> .
@prefix pq: <http://www.wikidata.org/prop/qualifier/> .
@prefix wikibase: <http://wikiba.se/ontology#> .
wd:Q683814 p:P127 s:Q683814-d5054028-4345-88f5-c931-b20e3241efbb .
s:Q683814-d5054028-4345-88f5-c931-b20e3241efbb
    a wikibase:Statement,
        wikibase:BestRank ;
    ps:P127 wd:Q5284 ;
    pq:P580 "1994-01-01T00:00:00Z" .
```

Listing 5: Wikibase RDF representation for Codex Leicester demonstrating how the item is linked via property `p:P127` to its ownership statement, which asserts that Bill Gates is the owner, with a qualifier indicating the start time of ownership.

This design is fully RDF 1.1 compliant since it uses only triples and standard serialization formats (such as Turtle, RDF/XML, or JSON-LD), though it leverages a specialized Wikibase vocabulary.

Within the cultural heritage domain, CIDOC CRM (Doerr, 2003) employs its **E13 Attribute Assignment** class to record the action of making assertions about an object's property or the relationship between two items or concepts. Conceptually similar to the Wikibase mechanism, the E13 Attribute Assignment class permits the attachment of provenance metadata such as the asserting agent, the date of

assertion, and the contextual conditions directly to the assertion. This approach not only captures the initial attribution but also supports the documentation of subsequent or even conflicting assertions over time, reflecting the evolving nature of cultural heritage interpretations.

4.4.2 Associating provenance through RDF quadruples

Named graphs (Carroll, Bizer, Hayes, & Stickler, 2005) extend the basic RDF model, traditionally composed of triples, by adding a fourth element to form quadruples. This fourth element is the graph URI, which serves as a contextual identifier. This structure allows RDF statements to not only describe resources but also to encapsulate metadata about the graphs themselves.

The serialization of named graphs can be achieved through extensions of existing RDF formats. RDF/XML, Turtle, and NTriples have been extended to TriX (Carroll & Stickler, 2004), TriG (W3C, 2024b), and N-Quads (W3C, 2024a) respectively. These formats are standardized and compatible with SPARQL. The W3C has proposed named graphs as a foundational element for expressing the provenance of scientific statements in a model known as "nanopublications" (Groth, Gibson, & Velterop, 2010). A nanopublication typically consists of three interrelated named graphs: one for the core data, one for provenance, and one for publication metadata. This structure provides a framework for ensuring the credibility and reproducibility of scientific information.

Named graphs enhance RDF by allowing triples to be grouped into distinct graphs identified by URIs. However, a limitation arises when handling implicit triples generated through inference. Consider RDF Schema (RDF/S) (W3C, 2014), which introduces inference rules that generate implicit triples not explicitly declared. When a named graph is deleted, the associated inference rules and hence the derived triples are also lost because named graphs do not inherently distinguish between explicit and inferred triples.

To overcome these limitations, **RDF/S Graphsets** (Pediaditis, Flouris, Fundulaki, & Christophides, 2009) and **RDF Triple Coloring** (Flouris, Fundulaki, Pediaditis, Theoharis, & Christophides, 2009) extend named graphs' capabilities. RDF/S Graphsets (Pediaditis, 2008) extend named graphs by allowing co-owned inferred triples to reside in a graphset URI rather than a single named graph. In the example in Listing 6, the triple `ex:manuscript10245 rdf:type ex:CulturalArtefact` results from combining two named graphs: `ex:namedGraph1`, which states that `ex:manuscript10245` is of type `ex:Manuscript` and that `ex:Manuscript` is a subclass of `ex:WrittenWork`, and `ex:namedGraph2`, which states that `ex:WrittenWork`

is a subclass of `ex:CulturalArtefact`.

By applying transitive reasoning over `rdfs:subClassOf`, it follows that `ex:manuscript10245` is of type `ex:CulturalArtefact`. Since this conclusion does not originate from a single named graph but rather from their combination, it is explicitly recorded under `ex:graphset1` to preserve its provenance and co-ownership, ensuring that the inferred knowledge remains intact even if one contributing named graph is altered or deleted.

```
@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

GRAPH ex:namedGraph1 {
    ex:manuscript10245 rdf:type ex:Manuscript .
    ex:Manuscript rdfs:subClassOf ex:WrittenWork .
}
GRAPH ex:namedGraph2 {
    ex:WrittenWork rdfs:subClassOf ex:CulturalArtefact .
}
GRAPH ex:graphset1 {
    ex:graphset1 rdf:type ex:Graphset ;
        ex:hasNamedGraph ex:namedGraph1, ex:namedGraph2 .
    ex:manuscript10245 rdf:type ex:CulturalArtefact .
}
```

Listing 6: An example of RDF/S graphsets represented in TriG syntax, illustrating how the triple `ex:manuscript10245 rdf:type ex:CulturalArtefact` is recorded under `ex:graphset1` to indicate that it is inferred from both `ex:namedGraph1` and `ex:namedGraph2`.

In contrast, RDF Triple Coloring (Flouris et al., 2009) provides a way to track the provenance of both explicit and inferred statements using standard RDF constructs. As with named graphs, each triple is stored as a quadruple (`s`, `p`, `o`, `c`), where `c`, the "color," is a URI denoting the source graph. However, when an implicit triple follows from two or more explicit statements, the inferred statement goes into a new color, conceptually "color1 + color2." Listing 7 shows a TriG example of this approach.

Despite the presence of named graphs in both approaches, RDF triple coloring and RDF/S graphsets diverge in how they extend provenance tracking. In the triple-coloring model, each color is simply a named graph within the scope of the RDF 1.1 Recommendation, and it fully aligns with SPARQL 1.1's native support for `GRAPH`, `FROM NAMED`, and other relevant keywords. By contrast, RDF/S graphsets formalize an additional concept of "graphsets" that group named graphs, collectively assign ownership to their derived statements, and define custom entailment rules about

```
@prefix ex: <http://example.org/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
ex:Color1 {
    ex:manuscript10245 rdf:type ex:Manuscript .
    ex:Manuscript rdfs:subClassOf ex:WrittenWork .
}
ex:Color2 {
    ex:WrittenWork rdfs:subClassOf ex:CulturalArtefact .
}
ex:Color1_2 {
    ex:manuscript10245 rdf:type ex:CulturalArtefact .
}
```

Listing 7: Representation of explicit and inferred statements using RDF Triple Coloring in TriG.

how these sets interact. Standard SPARQL 1.1 operations cannot directly address a "graphset" as a single resource unless the engine is extended to interpret this extra layer of grouping.

Building upon the named graphs foundation, **conjectural graphs** (Vitali & Pasqual, 2025) address the need to express uncertain or evolving claims in RDF. In the Arts, scholars often debate claims with no clear consensus. Standard RDF semantics treat all triples as asserted facts: including a triple in a graph implies accepting it as true within that graph. However, provenance documentation and historical scholarship require mechanisms to record statements without committing to their truth value. Conjectural graphs allow the representation of such uncertain claims without asserting their truth, using a unique predicate for each conjectured triple. This approach fulfills the need to Express Without Asserting (EWA): representing information content separately from truth commitment.

Consider the painting "Girl reading a letter at an open window," which has been attributed to Rembrandt, Hooch, and Vermeer (Figure 4.2). Using conjectural graphs, these competing attributions can be represented without asserting any of them as absolute truth. Each competing attribution is represented by a unique predicate, ensuring that these statements are treated as conjectural forms rather than asserted facts. These unique predicates are linked to the original predicate to maintain clarity and traceability, as shown in Listing 8.

In this example, each attribution (Rembrandt, Hooch, Vermeer) is represented by a unique conjectural predicate (`ex:cp1`, `ex:cp2`, `ex:cp3`). These conjectural predicates are mapped to the original predicate `crm:P14_carried_out_by` using the `conj:isAConjecturalFormOf` property. This mapping ensures that the original statements are not asserted, providing a clear distinction between conjectures and established facts.

Figure 4.2: Johannes Vermeer's "Girl Reading a Letter at an Open Window" (c. 1657-1659), exemplifying artworks with disputed attributions that benefit from conjectural graph representation.

The contested attribution of this painting illustrates the broader scope of provenance documentation discussed in Section 4.1: provenance extends beyond ownership records to encompass the evolving scholarly interpretations that constitute an object's biography, the reasoning processes behind decisions, and the competing value frameworks that different stakeholders bring to the same heritage entity.

4.4.3 Extending the RDF data model

Quadruples are not the only strategy to attach provenance information to RDF triples. The RDF data model can be extended to achieve this goal. The first proposal of this kind was **Notation3 Logic** (Berners-Lee, 2005), which introduced *formulae*. *Formulae* allow producing statements on N3 sentences, which are encapsulated by the syntax `{...}`. This approach enables attaching metadata to groups of triples without asserting their truth, making it particularly suitable for provenance representation.

```
@prefix ex: <http://example.org/> .
@prefix conj: <http://w3id.org/conjectures/> .
@prefix crm: <http://www.cidoc-crm.org/cidoc-crm/> .
ex:painting a crm:E22_Human-Made_Object .
ex:painting_activity a crm:E14_Activity ;
    crm:P108_has_produced ex:painting .
GRAPH <http://example.org/conjecture/rembrandt> {
    ex:painting_activity ex:cp1 ex:Rembrandt .
    ex:cp1 conj:isAConjecturalFormOf crm:P14_carried_out_by .
}
GRAPH <http://example.org/conjecture/hooch> {
    ex:painting_activity ex:cp2 ex:Hooch .
    ex:cp2 conj:isAConjecturalFormOf crm:P14_carried_out_by .
}
GRAPH <http://example.org/conjecture/vermeer> {
    ex:painting_activity ex:cp3 ex:Vermeer .
    ex:cp3 conj:isAConjecturalFormOf crm:P14_carried_out_by .
}
```

Listing 8: Example of using conjectural graphs to represent conflicting attributions of a painting.

For example, using the same manuscript case, N3 Logic can represent provenance information by quoting the original statement within *formulae* and attaching metadata to describe its source and confidence level, as shown in Listing 9.

```
@prefix ex: <http://example.org/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix prov: <http://www.w3.org/ns/prov#> .

{ex:manuscript10245 ex:hasDate "15th Century" .}
    prov:hadPrimarySource ex:archive123 ;
    dc:creator ex:curator56789 ;
    ex:confidence 0.85 .
```

Listing 9: Representation of provenance using N3 Logic *formulae*, where a statement is quoted within curly braces and annotated with source and confidence metadata.

In this example, the original statement about the manuscript's date is enclosed within curly braces, creating a *formula* that does not assert the statement as true but allows it to be referenced and annotated with provenance metadata. This mechanism provides a way to express statements without asserting them, which proves valuable for representing uncertain or disputed information.

Beyond *formulae* for provenance representation, Berners-Lee and Connolly proposed an additional mechanism within N3 Logic for tracking changes over time through a patch file format for RDF deltas (Berners-Lee & Connolly, 2004). This approach introduces three specialized terms: `diff:replacement`, which allows expressing any change between graph versions; `diff:deletion`, which serves as a

shortcut to express removed triples; and `diff:insertion`, which provides a shortcut for added triples. This delta representation offers significant storage efficiency: given two graph versions G1 and G2, the storage cost becomes directly proportional to the actual differences between the graphs rather than storing complete duplicates.

However, while N3 Logic provides high expressiveness for both provenance representation through *formulae* and change tracking through delta patches, it faces adoption limitations. Although N3 conforms to the SPARQL algebra, it does not comply with the standard RDF data model and relies on the specialized N3 Logic Vocabulary, which restricts interoperability with standard RDF tools and serialization formats.

Adopting a different perspective, **RDF+** (Dividino, Sizov, Staab, & Schueler, 2009) solves the problem by attaching a provenance property and its value to each triple, forming a quintuple, as shown in Table 4.2. Additionally, it extends SPARQL with the expression `WITH META MetaList`, which includes graphs specified in `MetaList`, containing RDF+ meta knowledge statements (Dividino et al., 2009). To date, RDF+ is not compliant with any standard, neither the RDF data model, nor SPARQL, nor any serialization formats.

Table 4.2: An RDF+ quintuple.

Subject	**Predicate**	**Object**	**Meta-property**	**Meta-value**
`ex:manuscript10245`	`ex:hasDate`	"15th Century"	`ex:accordingTo`	`ex:archive123`

SPOTL(X) (Hoffart, Suchanek, Berberich, & Weikum, 2013) allows expressing triple provenance through quintuples. The framework's name means Subject Predicate Object Time Location. Optionally, it is possible to create sextuples that add context to the previous elements. SPOTL(X) is concretely implemented in YAGO (Hoffart et al., 2013), a knowledge base automatically built from Wikipedia, given the need to specify which time, space, and context a specific statement is true. Outside of YAGO, SPOTL(X) does not follow either the RDF data model or the SPARQL algebra, and there is no standard serialization format.

Also, **annotated RDF (aRDF)** (Udrea, Recupero, & Subrahmanian, 2010) lacks standardization. A triple annotation has the form s p:λ o, where λ is the annotation linked to the property. Annotated RDF Schema extends this pattern by annotating an entire triple. It presents a SPARQL extension to query annotations, called AnQL (Zimmermann, Lopes, Polleres, & Straccia, 2012). Additionally, it specifies three application domains: the temporal, fuzzy, and provenance domains, as shown in Table 4.3.

The most recent proposal for extending the RDF data model to handle provenance information originated as **RDF-star** (Hartig & Thompson, 2019), which in-

Table 4.3: Annotated RDF Schema application examples.

Domain	**Annotated triple**	**Meaning**
Temporal	(`ex:manuscript10245`, `dct:created`, "1450-01-01"): [1450]	Manuscript was created in the year 1450
Provenance	(`ex:manuscript10245`, `dct:creator`, `ex:author123`): Archive	Manuscript was created by Author123 according to the Archive
Fuzzy	(`ex:manuscript10245`, `dct:condition`, `ex:good`): 0.8	Manuscript is in good condition to a degree of 0.8
Combined	(`ex:manuscript10245`, `dct:creator`, `ex:author123`): `<[1450], 1, Archive>`	Manuscript was created by Author123 in 1450, according to the Archive

troduced the concept of embedding triples within other triples to replace RDF Reification with less verbose semantics. The original proposal, developed as a W3C Community Group specification (Gschwend & Lassila, 2022), defined "quoted triples" delimited by `«` and `»` and introduced extensions to Turtle and SPARQL (Turtle-star and SPARQL-star). RDF-star was adopted in YAGO4 to attach temporal information to facts through `schema:startDate` and `schema:endDate` (Suchanek, Lajus, Boschin, & Weikum, 2019), and was proposed for enabling statement-level annotations in RDF streams through RSP-QL-star (Keskisärkkä, Blomqvist, Lind, & Hartig, 2019).

The W3C has since incorporated and refined the RDF-star concepts into the RDF 1.2 working draft (Kellogg et al., 2025), introducing new terminology and a more precise formal model. The central construct is the **triple term**, written as `«( s p o )»`: an RDF triple used as a term in the object position of another triple. A triple term denotes a proposition, that is, the abstract content of a statement. It does not by itself assert anything: including `«( ex:manuscript10245 ex:hasDate "15th Century" )»` as the object of a triple does not imply that the manuscript has date "15th Century." The proposition is asserted only if the corresponding triple `ex:manuscript10245 ex:hasDate "15th Century"` also appears as a separate element of the graph (Lindström & Champin, 2025).

RDF 1.2 uses triple terms through a mechanism called reification, based on the new predicate `rdf:reifies`. A triple whose predicate is `rdf:reifies` and whose object is a triple term is called a reifying triple. Its subject, called a reifier, represents a concrete occurrence of the proposition denoted by the triple term. Once created, the reifier can serve as the subject of additional triples that attach prove-

nance metadata, temporal boundaries, or other contextual information. Listing 10 illustrates this mechanism using the same manuscript example.

```
@prefix ex: <http://example.org/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
ex:manuscript10245 ex:hasDate "15th Century" .
_:r1 rdf:reifies <<( ex:manuscript10245 ex:hasDate "15th Century" )>> .
_:r1 dc:creator ex:curator56789 .
```

Listing 10: Representation of provenance using RDF 1.2 triple terms and reification. The first triple asserts the date. The second triple links a reifier (`_:r1`) to the triple term denoting the proposition. The third triple attaches provenance metadata to the reifier.

In Listing 10, the first triple asserts that the manuscript has date "15th Century." The second triple creates a reifier (`_:r1`) linked through `rdf:reifies` to the triple term `«( ex:manuscript10245 ex:hasDate "15th Century" )»`, which denotes the proposition expressed by the first triple. The third triple attaches provenance information to the reifier, recording that `ex:curator56789` is responsible for this assertion. Compared to RDF Reification (Listing 1), which requires five triples to decompose the original statement into its subject, predicate, and object components, RDF 1.2 reification achieves the same result with three triples by referencing the proposition directly as a triple term.

This mechanism natively supports expressing statements without asserting them (EWA). If the first triple in Listing 10 were removed, the reifier would still reference the proposition through the triple term, but the proposition itself would not be part of the asserted graph. This enables the representation of uncertain, disputed, or hypothetical claims without committing to their truth value.

4.5 RDF provenance ontologies

A wide range of OWL ontologies and vocabularies represent provenance information, categorized as upper ontologies, domain ontologies, or provenance-related ontologies. Upper ontologies include the Proof Markup Language (da Silva, McGuinness, & Fikes, 2006), PROV ontology (Lebo, Sahoo, & McGuinness, 2013), and the Open Provenance Model (Moreau et al., 2011). Domain ontologies encompass the SWAN Ontology for Neuromedicine (Ciccarese et al., 2008), Provenir ontology for eScience (Sahoo & Sheth, 2009), and PREMIS for archived digital objects (Caplan, 2017). Provenance-related ontologies include Dublin Core Metadata Terms (Board, 2020) and the OpenCitations Data Model (Daquino et al., 2020).

Among the upper ontologies, the **Open Provenance Model** (Moreau et al., 2011) stands out because of its interoperability. It describes the history of an entity in terms of processes, artifacts, and agents (Moreau et al., 2011), a pattern later incorporated into the PROV Data Model (W3C, 2013b).

Among domain-relevant models, several OWL ontologies have been developed to address the specific needs of their respective fields. The **Provenir ontology** (Sahoo & Sheth, 2009) is tailored to capture the dynamic processes and data evolution in scientific research. **PREMIS** (Caplan, 2017) is designed to document the provenance of archived digital objects such as files and bitstreams. The **Semantic Web Applications in Neuromedicine (SWAN) ontology** (Ciccarese et al., 2008) supports the modeling of scientific discourse in the biomedical research context.

Dublin Core Metadata Terms (Board, 2020) provides a dedicated vocabulary for provenance tracking. The specification includes a `dct:provenance` property that captures statements about changes in ownership and custody since resource creation. Agent-related properties such as `dct:creator`, `dct:contributor`, and `dct:publisher` track the entities responsible for resource creation, contribution, and publication, with all values constrained to instances of `dct:Agent`. Temporal provenance is captured through date properties including `dct:created`, `dct:dateAccepted`, `dct:dateSubmitted`, `dct:issued`, and `dct:modified`. The Dublin Core Metadata Initiative has established a dedicated Metadata Provenance Task Group to define application profiles for making assertions about metadata statements, and maintains official mappings between Dublin Core terms and the PROV ontology to ensure interoperability across provenance frameworks (W3C, 2013a).

All these requirements and OWL ontologies have been merged into a single data model, the **PROV Data Model** (W3C, 2013b), translated into the **PROV Ontology** (Lebo et al., 2013) using the OWL 2 Web Ontology Language. It provides several classes, properties, and restrictions, representing provenance information in different systems and contexts. Its level of genericity makes it possible to create new classes and data model-compatible properties for new applications and domains. PROV-DM captures provenance under three complementary perspectives:

- Agent-centered provenance entails people, organizations, software, inanimate objects, or other entities involved in generating, manipulating, or influencing a resource. PROV-O maps the responsible agent with `prov:Agent`, the relationship between an activity and the agent with `prov:wasAssociatedWith`, and an entity's attribution to an agent with `prov:wasAttributedTo`.

- Object-centered provenance represents the origin of a document's portion from

other documents. PROV-O maps a resource with `prov:Entity`, whether physical, digital, or conceptual, while the predicate `prov:wasDerivedFrom` expresses a derivation relationship.

- Process-centered provenance encompasses the actions and processes necessary to generate a resource. PROV-O expresses the concept of action with `prov:Activity`, the creation of an entity with `prov:wasGeneratedBy`, and the use of another entity to complete a process with `prov:used`.

The Graffoo diagram in Figure 4.3 (Falco, Gangemi, Peroni, Shotton, & Vitali, 2014) provides a high-level view of the discussed concepts' structure, constituting the so-called "starting point terms".

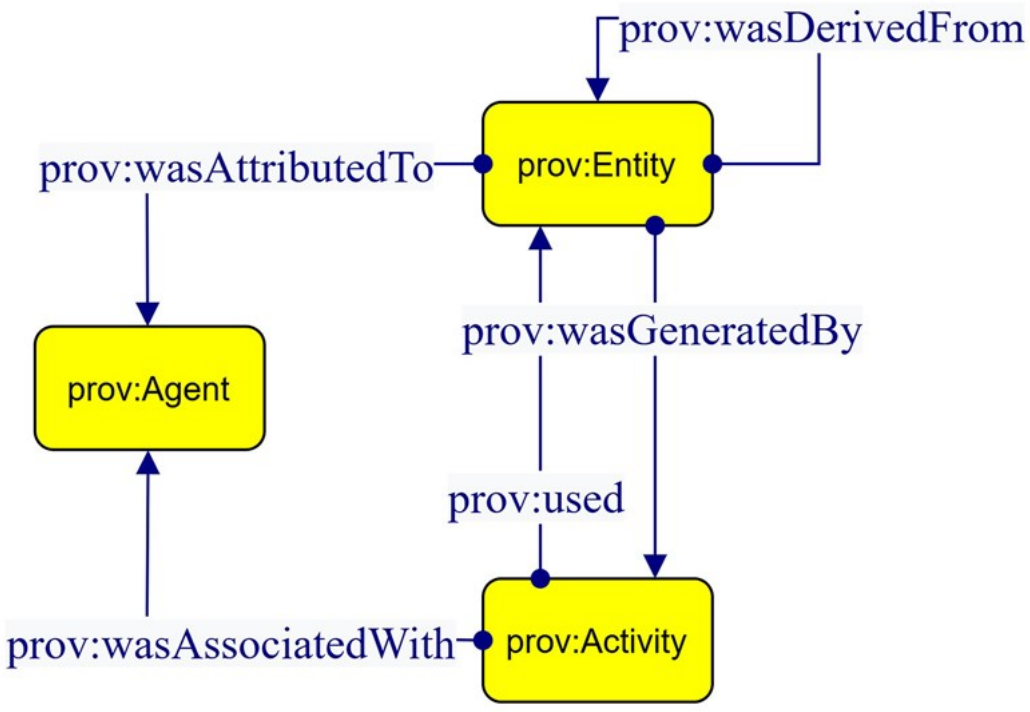
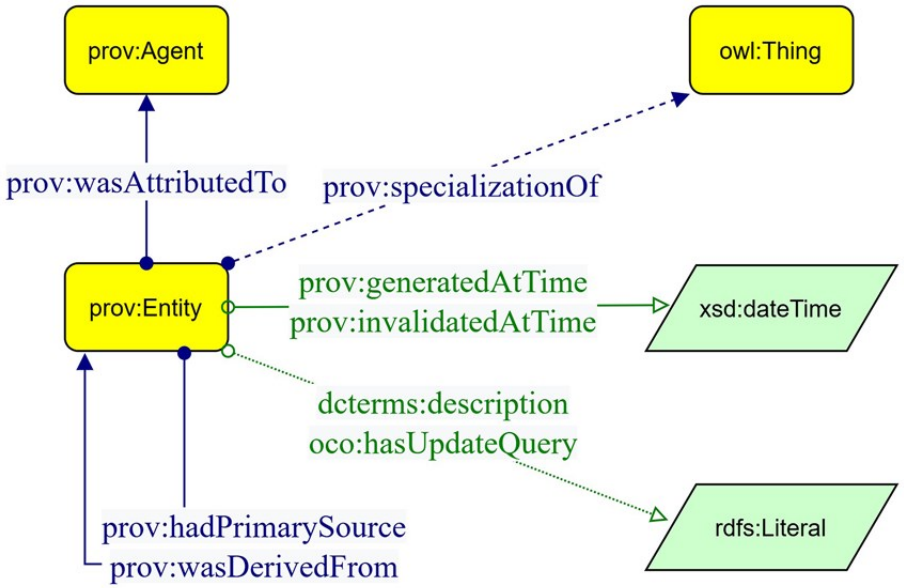


Figure 4.3: High level overview of PROV records.

Finally, the **OpenCitations Data Model (OCDM)** (Daquino et al., 2020) represents provenance and tracks changes in a way that complies with RDF 1.1. It relies on well-known and widely adopted standards such as PROV-O (Lebo et

al., 2013), named graphs, and Dublin Core (Daquino et al., 2020). Each entity described by the OCDM is annotated with one or more snapshots of provenance. The snapshots are of type `prov:Entity` and are connected to the entity described through `prov:specializationOf`, a predicate present in the PROV-O "expanded terms". Being the specialization of another entity means sharing every aspect of the latter and, in addition, presenting more specific aspects, such as an abstraction, a context, or, in this case, a time.

Each snapshot records temporal boundaries: `prov:generatedAtTime` indicates when the snapshot became valid by marking the moment the entity entered its current state, while `prov:invalidatedAtTime` specifies when the snapshot ceased to be valid due to subsequent modifications. Additionally, snapshots capture the agents responsible for both creation and modification of the data (`prov:wasAttributedTo`), the primary sources (`prov:hadPrimarySource`) and a link to the previous snapshot in time (`prov:wasDerivedFrom`). The model is summarized in Figure 4.4.



Prefixes	
dcterms:	http://purl.org/dc/terms/
oco:	https://w3id.org/oc/ontology/
owl:	http://www.w3.org/2002/07/owl#
prov:	http://www.w3.org/ns/prov#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
xsd:	http://www.w3.org/2001/XMLSchema#

Figure 4.4: Provenance in the OpenCitations Data Model.

Furthermore, OCDM extends PROV-O by introducing a new property called `hasUpdateQuery`, a mechanism to record additions and deletions from an RDF graph

with a SPARQL `INSERT DATA` and `DELETE DATA` query string (Daquino et al., 2020; Peroni, Shotton, & Vitali, 2016). The snapshot-oriented structure, combined with a system to explicitly indicate how a previous snapshot was modified to reach the current state, makes it easier to restore an entity to a specific snapshot.

Figure 4.5 provides a high-level visualization of this change tracking mechanism. Each snapshot captures the complete state of an entity at a specific point in time, with provenance information (who made the change, when it occurred, and what source was used). The transitions between snapshots are represented as deltas: the green circles indicate triples added, the red circles show triples removed, and the overlapping green-red circles represent the combination of additions and removals that transform one snapshot into the next.

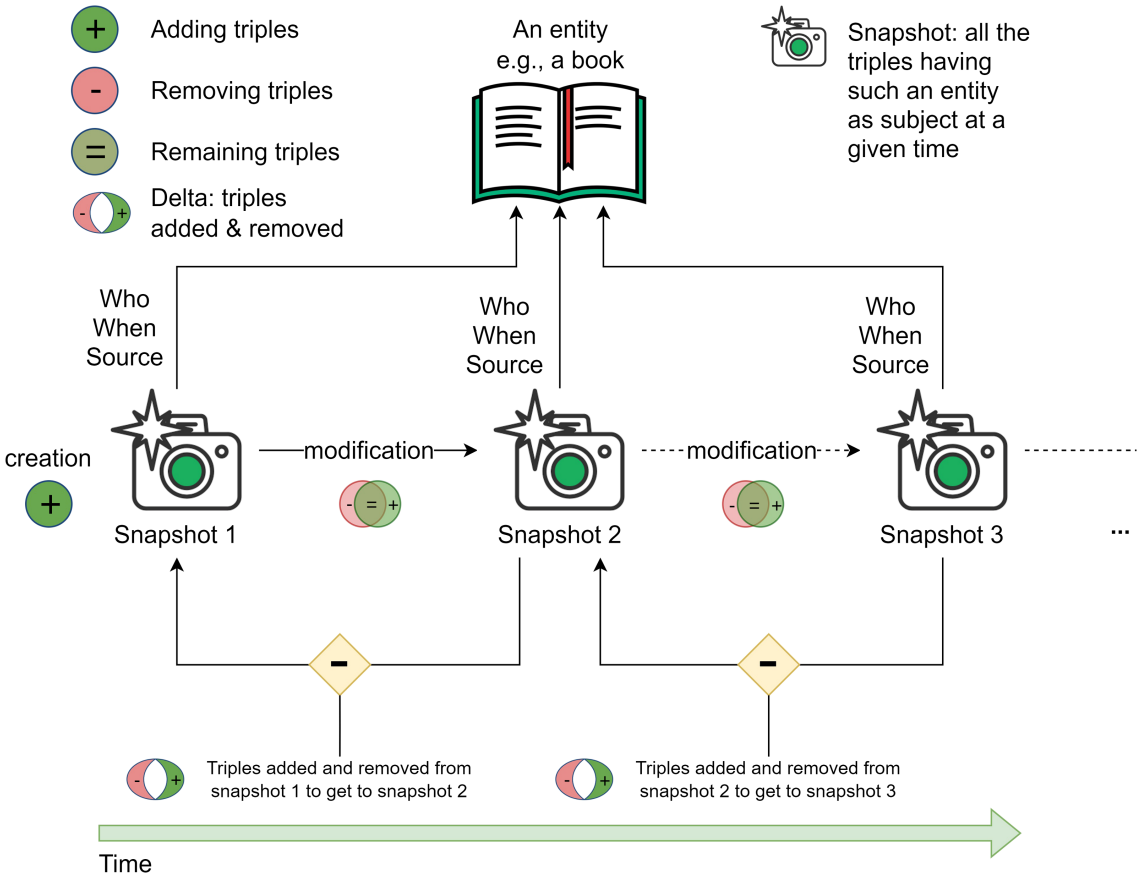


Figure 4.5: High-level overview of the OpenCitations Data Model change tracking mechanism, showing how entity states are captured through snapshots and the transitions between them are recorded as deltas of added and removed triples.

4.6 Analysis and evaluation of provenance models

Understanding the various models for representing provenance in RDF proves necessary for making informed choices about which to implement, particularly in the realm of Digital Humanities and cultural heritage metadata management. The analysis reveals that named graphs, RDF 1.2 triple terms, PROV-O, Dublin Core, conjectural graphs, and the OpenCitations Data Model stand out as the most effective models, as summarized in Figure 4.6 and detailed in Table 4.4.

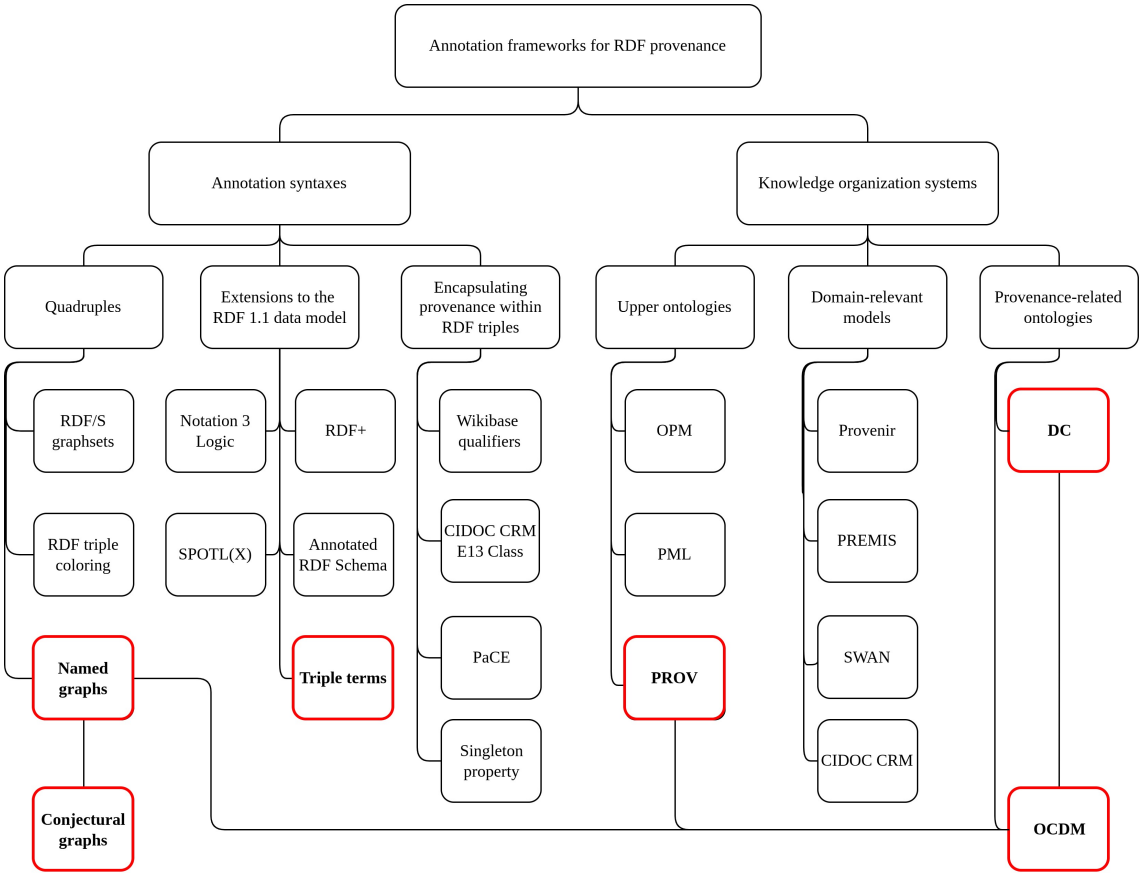


Figure 4.6: Annotation frameworks for RDF provenance. The edges represent a membership relationship, while the bold type and red circling highlight the syntaxes identified as most relevant.

Table 4.4 reveals two primary dividing lines among the surveyed approaches. The first concerns standard compliance: named graphs, RDF triple coloring, and all triple-based approaches (Wikibase, CIDOC CRM E13, PaCE, singleton properties, conjectural graphs) conform to RDF 1.1 and SPARQL 1.1, while RDF/S graphsets, N3Logic, aRDF, RDF+, SPOTL(X), and RDF 1.2 triple terms require extensions beyond the current RDF 1.1 standard. Among these, only RDF 1.2 triple terms are progressing toward W3C standardization as part of the RDF 1.2 and SPARQL

Table 4.4: Advantages and disadvantages of metadata representations models for RDF. The table was expanded from the one in (Sikos & Philp, 2020).

Approach	**Tuple type**	**RDF**	**SPARQL**	**Serialisations**	**Ext. vocabulary**	**Scalable**	**EWA**
Named graphs	Quadruple	✓	✓	[a]	–	✓	†
RDF/S graphsets	Quadruple	–	–	[a]	–	✓	†
RDF triple coloring	Quadruple	✓	✓	[a]	–	✓	†
N3Logic	Triple (N3)	–	✓	N3	N3 Logic Vocab	✓	✓
aRDF & Ann. RDF Schema	Nonstandard	–	–	–	–	✓	✓
RDF+	Quintuple	–	–	–	–	✓	✓
SPOTL(X)	Quint./sextuple	–	–	–	–	†	✓
Triple terms	Triple (triple term)	–	–	[b]	–	✓	✓
Wikibase qualifiers	Triple	✓	✓	[c]	Wikibase ont.	–	–
CIDOC CRM E13 Class	Triple	✓	✓	[c]	CIDOC CRM ont.	–	–
PaCE	Triple	✓	✓	[c]	Provenir ont.	–	–
Singleton property	Triple	✓	✓	[d]	Singleton prop.	–	–
Conjectural graph	Quadruple	✓	✓	[a]	Conjectural prop.	✓	✓

[a] TriG, TriX, N-Quads. [b] Turtle 1.2, TriG 1.2, N-Triples 1.2, N-Quads 1.2 (WD). [c] Turtle, N-Triples, RDF-JSON, JSON-LD, RDFa, HTML5. [d] RDF/XML, N3, Turtle, N-Triples, RDF-JSON, JSON-LD. † Depends on implementation.

1.2 working drafts. The second dividing line concerns scalability: quadruple-based approaches (named graphs, RDF triple coloring) and RDF data model extensions (N3Logic, aRDF, RDF+, RDF 1.2 triple terms) scale to large datasets, whereas triple-based encapsulation methods (Wikibase, CIDOC CRM E13, PaCE, singleton properties) require additional triples for each annotated statement, limiting scalability in large RDF collections.

The approaches also differ in their reliance on external vocabularies. Named graphs and RDF triple coloring operate without external dependencies, while triple-based methods require domain-specific vocabularies (Wikibase ontology, CIDOC CRM, Provenir ontology, singleton property vocabulary). This dependency introduces coupling between the provenance representation and a specific vocabulary, affecting interoperability across systems that adopt different conventions.

The representation models discussed above define how provenance is structurally attached to RDF statements. Orthogonally, the ontologies reviewed in Section 4.5 define what provenance information is recorded. PROV-O, as a W3C recommendation, provides a vocabulary of agents, activities, and entities that can be combined with any of the compliant representation models. Dublin Core contributes agent attribution and temporal properties that are widely recognized in bibliographic contexts. The OpenCitations Data Model builds on both: it adopts named graphs as its representation layer, uses PROV-O and Dublin Core for provenance semantics, and adds snapshot-based change tracking through SPARQL update queries (Section 4.5). This layered design means that OCDM inherits the standard compliance and scalability of named graphs while gaining the provenance expressiveness of PROV-O.

From the perspective of expressing statements without asserting them, different models exhibit varying levels of support. Named graphs present an ambiguous case: while they allow grouping of statements under a graph URI, their semantics depend on the interpretation of the default graph and the conventions adopted by specific implementations. This ambiguity makes them a flexible but not inherently reliable method for expressing without asserting.

N3Logic offers a more explicit approach to expressing without asserting by allowing graphs to be quoted rather than asserted. This mechanism enables the representation of statements within logical constructs without incorporating them into the asserted knowledge base, making it one of the most expressive solutions for non-asserted RDF representation. However, its adoption remains limited due to its reliance on Notation3 (N3), which is not natively supported in standard RDF tools.

RDF 1.2 triple terms provide a lightweight syntax for referring to statements

within other triples, and the reification mechanism through `rdf:reifies` offers a way to describe statements without asserting them. Similar functionality is observed in SPOTL(X) and RDF+, which extend RDF to include additional elements such as context and temporal dimensions. These models share the advantage of reducing verbosity compared to traditional reification, while still allowing for the expression of uncertainty or disputed claims. However, among them, only RDF 1.2 triple terms are progressing toward standardization, with the W3C Working Group developing both the data model and corresponding SPARQL 1.2 query language extensions.

By contrast, Wikibase qualifiers and CIDOC CRM E13 provide mechanisms for enriching statements with additional metadata but do not inherently prevent the base triple from being asserted. Likewise, PaCE and the Singleton Property method create new URIs for contextualizing statements, but the modified triples remain part of the asserted dataset. Conjectural graphs, in contrast, explicitly structure statements as conjectural rather than asserted by introducing new predicate URIs that isolate the conjectured statement from the dataset's asserted portion. This allows competing or uncertain claims to be represented without treating them as established facts, making them particularly suitable for Digital Humanities and cultural heritage research contexts.

Among the models reviewed, the OpenCitations Data Model offers the most complete combination of properties for the purposes of this thesis. OCDM builds on named graphs for standard compliance and scalability, adopts PROV-O and Dublin Core for provenance semantics, and adds snapshot-based change tracking through SPARQL update queries. Data managed through OCDM remains queryable with standard SPARQL and storable in any RDF 1.1 quadstore, while each modification is recorded with the responsible agent, the primary source, and the temporal boundaries of validity. The following chapters build on OCDM as the provenance and change tracking layer for the solutions developed in this thesis.

Chapter 5

Temporal queries and change management in RDF datasets

Chapter 4 established that the OpenCitations Data Model provides mechanisms for representing provenance and change tracking through snapshot-based versioning and SPARQL update queries. However, representation alone does not address the practical requirements identified in Chapter 3. ParaText philologists need to reconstruct how entities appeared at specific points in time to understand the evolution of scholarly interpretations. OpenCitations Meta requires restoration capabilities to revert entities to previous valid states when automated corrections introduce errors. These scenarios demand algorithms that can process OCDM snapshots and delta records to reconstruct historical entity states while maintaining consistency across related entities.

OCDM snapshots document what changed and when, but implementing version reconstruction presents methodological challenges. Restoring an entity to a previous state requires traversing its snapshot history and applying delta operations in reverse. When entities reference other entities, temporal reconstruction must ensure that all related resources align to the same temporal point, preventing inconsistencies where an entity references a version of another entity that did not exist at that time. This chapter examines approaches for performing time-traversal queries on OCDM datasets, presenting algorithms for version materialization, temporal consistency management, and restoration operations that maintain referential integrity across entity relationships.

The discussion begins with an analysis of temporal query typologies, establishing a taxonomy for different categories of time-traversal queries and their specific requirements. Subsequently, the chapter presents the Time-Agnostic Library, a practical implementation that enables temporal querying over versioned RDF datasets.

This chapter builds upon and extends the methodological discussion presented by Massari and Peroni (2022).

5.1 Temporal query typologies for RDF datasets

Fernández, Umbrich, Polleres, and Knuth (2016) established a systematic classification for time-agnostic queries. Their framework introduces a high-level categorization based on "retrieval needs": reconstructing complete entity versions at specific time points, extracting change deltas between versions, and executing queries across multiple temporal states.

Before detailing such queries, definitions of entity, time-aware dataset, and version are required:

Definition 1 (Entity). An entity $\mathcal{E}$ is the set of RDF triples `(s,p,o)` having the same subject `s`.

Definition 2 (Time-aware dataset). A version-annotated entity is an entity $\mathcal{E}$ annotated with a label i representing the version in which this entity holds, denoted by the notation $\mathcal{E}_i$, where $i \in \mathcal{N}$. A time-aware dataset $\mathcal{A}$ is a set of version-annotated entities.

Definition 3 (Version). A version of a time-aware dataset $\mathcal{A}$ at snapshot i is the RDF graph $\mathcal{A}_i = \{\mathcal{E} \mid \mathcal{E}_i \in \mathcal{A}\}$.

In the query definitions, the evaluation of a SPARQL query Q on a graph $\mathcal{G}$ produces a bag of solution mappings $[[Q]]_{\mathcal{G}}$.

The following query types are defined:

Version materialization (VM) retrieves the complete state of the dataset at a specific version i, that is, $\mathcal{A}_i$. For example, "Retrieve the state of the dataset in 2014".

Single-version structured query (SV) retrieves the results of a SPARQL query targeted at any specific version. Formally: $SV(Q, \mathcal{A}_i) = [[Q]]_{\mathcal{A}_i}$, where $i \in \mathcal{N}$ can be any version. For example, "Which David Shotton's papers were featured in the dataset in 2014?".

Cross-version structured query (CV), also called time-traversal query, retrieves the results of a SPARQL query targeted at multiple versions. Formally: $CV(Q, \mathcal{A}_i, \mathcal{A}_j) = SV(Q, \mathcal{A}_i) \bowtie SV(Q, \mathcal{A}_j)$, where $i, j \in \mathcal{N}$ can be any versions (not necessarily consecutive). For example, "Which David Shotton's papers were featured in the dataset in 2013 and in 2014?".

Delta materialization (DM) retrieves the differences between two versions $\mathcal{A}_i$ and $\mathcal{A}_j$: $DM(\mathcal{A}_i, \mathcal{A}_j) = (\Delta^+, \Delta^-)$, with $\Delta^+ = \mathcal{A}_i \setminus \mathcal{A}_j$ and $\Delta^- = \mathcal{A}_j \setminus \mathcal{A}_i$. For

example, "What changed in the dataset between 2013 and 2014?".

Single-delta structured query (SD) retrieves the change-sets of a SPARQL query's results between two versions: $SD\,(Q, \mathcal{A}_i, \mathcal{A}_j) = (\Delta^+, \Delta^-)$, with $\Delta^+ = [[Q]]_{\mathcal{A}_i} \setminus [[Q]]_{\mathcal{A}_j}$ and $\Delta^- = [[Q]]_{\mathcal{A}_j} \setminus [[Q]]_{\mathcal{A}_i}$. For example, "What are the additions and removals of David Shotton's papers between 2013 and 2014?".

Cross-delta structured query (CD) retrieves the change-sets of a SPARQL query's results between more than one consecutive couple of versions. Formally: $CD\,(Q, \mathcal{A}_i, \mathcal{A}_j, \mathcal{A}_m) = SD\,(Q, \mathcal{A}_i, \mathcal{A}_j) \bowtie SD\,(Q, \mathcal{A}_j, \mathcal{A}_m)$. For example, "Track the evolution of David Shotton's papers across multiple version transitions in the dataset".

Extensions of SPARQL exist to support queries on time-aware RDF datasets, but they either require using non-standard languages to map data, such as τ-SPARQL (Tappolet & Bernstein, 2009), T-SPARQL (Grandi, 2010), and AnQL (Zimmermann et al., 2012), or only work on purpose-built databases, *i.e.*, SPARQL^T on the RDF-TX system (Zaniolo, Gao, Atzori, Chen, & Gu, 2018).

SPARQL-LTL (Fionda, Chekol, & Pirrò, 2016) extends SPARQL with an algorithm for rewriting queries into standard SPARQL, requiring triples annotated with revision numbers and available as named graphs. The GiZe demonstration prototype is the only known implementation, and no publicly maintained version is available.

The methodology introduced in this chapter differs from all the above approaches in where temporal logic is handled: rather than extending the query language with temporal constructs or requiring a purpose-built database, the Time Agnostic Library relies on temporal information in data mapped according to the OCDM data model and reconstructs states programmatically, preserving standard SPARQL 1.1 syntax.

5.2 Storage paradigms for dynamic linked open data

Various archiving policies have been developed to store and query the evolution of RDF datasets, namely independent copies, change-based, timestamp-based, and fragment-based policies (Pelgrin, Galárraga, & Hose, 2021).

Independent copies consist of storing each version separately. This represents the most straightforward model to implement and allows performing version materialization, single-version queries, and cross-version queries easily. However, this approach requires massive amounts of space for storing and time for processing. Fur-

thermore, given the different statements' versions, additional diff mechanisms are required to identify what changed. Nevertheless, this remains the archiving policy adopted by most systems and knowledge bases, such as DBpedia (Lehmann et al., 2015), Wikidata (Vrandečić & Krötzsch, 2014), and YAGO (Suchanek et al., 2024).

Among the earliest version control systems for RDF was SemVersion (Völkel, Winkler, Sure, Kruk, & Synak, 2005), designed for RDF models and RDF-based ontology languages such as RDFS and OWL. It saves each version in a separate snapshot and differences are calculated on the fly. SemVersion supports version materialization and delta materialization but not via SPARQL, because SPARQL became a W3C Recommendation in 2008 and SemVersion has not been updated since 2006.

Change-based policy was introduced to solve scalability problems caused by the independent copies approach. It consists of saving only the deltas between one version and the other. For this reason, delta materialization is costless. The drawback is that additional computational costs for delta propagation are required to support version-focused queries.

The first proposal of this approach relied on a Relational Database Management System (RDBMS) to store the original dataset and the deltas between two consecutive versions (Im, Lee, & Kim, 2012). To improve performance, deltas are preprocessed and duplicated, or unnecessary modifications are deleted. There is no support for SPARQL and queries must be formulated in SQL.

A concrete implementation of a change-based policy is R&Wbase (Sande et al., 2013), a version control system inspired by Git but designed for RDF. Additions and deletions are stored using distinct context identifiers in a quad-store, and SPARQL queries are supported on versioned data. However, this model is not fully semantic, since it requires hash tables to map revisions with change-sets.

R43ples (Graube, Hensel, & Urbas, 2016) is inspired by R&WBase and improves it by adopting a totally semantic model called Revision Management Ontology, which records change-sets and the related provenance metadata in separate graphs using PROV-O and some new properties. R43ples acts as a proxy between the data triplestore and the provenance triplestore. However, R43ples cannot be considered a generic solution, as it extends SPARQL with keywords to simplify queries, and the current implementation mandates using Jena TDB as the provenance triplestore.

A more recent change-based approach uses RDF-star for triple-level provenance tracking (Dibowski, 2024). A provenance engine intercepts SPARQL UPDATE queries and transforms them into SPARQL-star INSERT DATA operations on a separate provenance knowledge graph. Changes and their provenance are repre-

sented using PROV-STAR, an extension of PROV-O that introduces three classes for triple change sets (generation, invalidation, and their abstract parent). Past versions can be restored via a single SPARQL-star CONSTRUCT query that selects all triples generated before a given timestamp and filters out those invalidated before it. The approach requires SPARQL-star support in the underlying triplestore. As far as we know, the provenance engine implementation has not been released as open-source software, preventing independent replication and evaluation of the system.

Timestamp-based policy annotates each triple with its transaction time, that is, the timestamp of the version in which that statement was in the dataset.

x-RDF-3X (Neumann & Weikum, 2010) is a database for RDF designed to manage high-frequency online updates, versioning, time-traversal queries, and transactions. The triples are never deleted but are annotated with two fields: the insertion and deletion timestamp, where the deletion timestamp has zero value for currently living versions. Afterward, updates are saved in a separate workspace and merged into various indexes at occasional savepoints. x-RDF-3X supports version materialization, single-version queries, and cross-version queries.

v-RDFCSA (Cerdeira-Pena, Farina, Fernandez, & Martinez-Prieto, 2016) and its generalization v-RDF-SI (Cerdeira-Pena, De Bernardo, Fariña, Fernández, & Martínez-Prieto, 2024) use a similar strategy but excel in reducing space requirements, compressing both the RDF archive and the timestamps attached to the triples. On BEAR-A, v-RDF-SI stores the full 325 GiB archive in 4.7–9.2 GiB. Both systems support only version materialization, delta materialization, and version queries on single triple patterns, without full SPARQL or dynamic ingestion of new versions. As far as we know, their source code is not publicly available.

Dydra (Anderson, 2019) is also a timestamp-based storage system, which indexes quadruples and associates their creation and addition times to form quintuples. It supports all query types but extends SPARQL with the REVISION keyword.

Fragment-based approach avoids reconstructing versions via deltas by saving only fragments of what changed. Different granularity levels are possible, depending on the requirements: a graph, a subgraph, or an entity.

Quit Store (Arndt, Naumann, Radtke, Martin, & Marx, 2019) inherits from R&Wbase and R43ples the Git-based distributed version control management approach. Modified fragments are saved in named graphs, and metadata are modeled according to PROV-O. The data model complies with RDF 1.1, and all query types are supported in plain SPARQL 1.1. However, the current implementation suffers from high memory requirements, as all queries are run on an in-memory quad-store.

Hybrid storage policies combine change-based and timestamp-based methods. OSTRICH (Taelman, Sande, & Verborgh, 2018) exemplifies this by retaining the first dataset version and subsequent deltas, while merging change-sets based on timestamps to reduce redundancies. OSTRICH natively supports version materialization, delta materialization, and version queries on single triple patterns.

TailR (Meinhardt, Knuth, & Sack, 2015) also preserves the first version and succeeding diffs but reduces the computational effort to reconstruct versions by also saving some intermediate snapshots, adopting an independent copies/change-based approach, and thus increasing space requirements. Queries are not made via SPARQL but via the Memento protocol (Jones, Klein, de Sompel, Nelson, & Weigle, 2021), *i.e.*, HTTP content-negotiation via the Accept-Datetime header.

Finally, the OpenCitations Data Model (Daquino et al., 2020) adopts a hybrid approach combining change-based and timestamp-based methods, representing provenance and changes in RDF.

5.3 The OpenCitations Data Model approach

Building upon the OCDM framework established in Section 4.5, this section demonstrates how the model's change tracking mechanisms enable temporal query processing. The `oco:hasUpdateQuery` property (Daquino & Peroni, 2019; Peroni et al., 2016), which records SPARQL `INSERT DATA` and `DELETE DATA` operations between entity snapshots, provides the foundation for implementing time-traversal queries. This system enables:

- Recovery of the current statements of an entity, as they are those available in the present dataset
- Restoration of an entity to a specific snapshot s_i, by applying the reverse operations of all update queries from the most recent snapshot s_n to s_{i+1}

SPARQL update queries representing deltas must contain only absolute URIs, literals, and blank nodes, while prefixes and variables are not permitted.

Figure 5.1 shows a usage example of the OpenCitations Data Model. The entity `id:062106312420` is an identifier of the bibliographic resource `br:062104388184`, whose title is "OpenCitations Meta". The identifier `id:062106312420` was initially registered with an incorrect DOI, *i.e.*, "https://doi.org/10.1162/qss_a_00292" instead of "10.1162/qss_a_00292", where the error is in the inclusion of the full URL instead of the DOI string. An agent corrected this mistake. Therefore, a new

snapshot was generated, associated with `id:062106312420`, and deriving from the previous snapshot. This example is taken from OpenCitations Meta.

Listing 11 shows the RDF representation in Turtle syntax.

5.4 The Time Agnostic Library

The OpenCitations Data Model provides a representation framework for provenance and change tracking in RDF datasets. To execute temporal queries on data stored according to this model, software capable of interpreting and processing this representation is required. The Time Agnostic Library implements algorithms that use the OCDM provenance specification to support the temporal query types identified in the taxonomy by Fernández et al. (2016).

The distinguishing characteristic of this approach compared to the systems discussed in Section 5.2 is that temporal queries are performed live, without requiring pre-indexing or data restructuring. This capability is a prerequisite for addressing RQ1: an interactive editing tool must allow users to add, modify, and delete triples and immediately reconstruct previous versions of the data. A system that requires stopping and re-indexing after each edit operation cannot support the continuous workflow that interactive curation demands.

5.4.1 Version and delta materializations

Version materialization reconstructs entity states at specific temporal points by inverting SPARQL UPDATE operations stored in the OCDM provenance graph. The implementation employs two distinct strategies depending on the reconstruction target: single point-in-time materialization versus complete history reconstruction.

The materialization process, shown in Listing 12, takes a time interval as a tuple of start and end timestamps, where either bound can be omitted. It retrieves entity snapshots through SPARQL queries against the provenance graph. Each snapshot contains generation time, attribution, and optionally an update query string. The algorithm filters snapshots within the requested interval; then, for each relevant snapshot, it collects all update queries from more recent snapshots and applies them inversely to a copy of the current entity state to reconstruct the historical state at that point.

The inverse transformation mechanism parses SPARQL UPDATE syntax to extract `INSERT DATA` and `DELETE DATA` operations. For each operation, the algorithm inverts the action: `DELETE DATA` operations become additions to the graph, while `INSERT DATA` operations become removals.

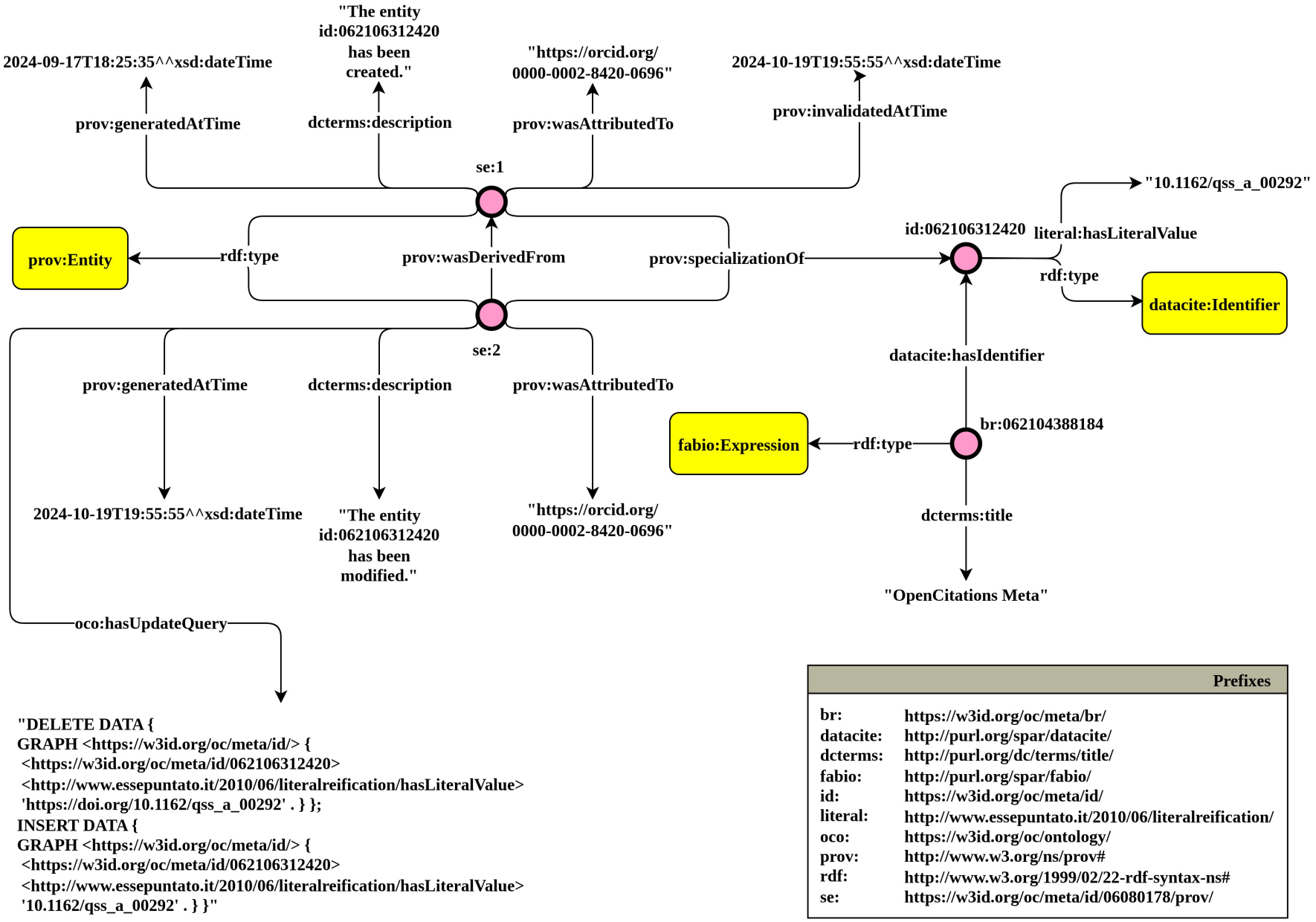


Figure 5.1: Usage example of the OpenCitations Data Model, shown via the graphical framework Graffoo (Falco et al., 2014).

```
@prefix br: <https://w3id.org/oc/meta/br/>.
@prefix datacite: <http://purl.org/spar/datacite/>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix fabio: <http://purl.org/spar/fabio/>.
@prefix id: <https://w3id.org/oc/meta/id/>.
@prefix literal: <http://www.essepuntato.it/2010/06/literalreification/>.
@prefix oco: <https://w3id.org/oc/ontology/>.
@prefix prov: <http://www.w3.org/ns/prov#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix se: <https://w3id.org/oc/meta/id/062106312420/prov/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

br:062104388184 a fabio:Expression;
    dcterms:title "OpenCitations Meta";
    datacite:hasIdentifier id:062106312420.

id:062106312420 a datacite:Identifier;
    datacite:usesIdentifierScheme datacite:doi;
    literal:hasLiteralValue "10.1162/qss_a_00292".

se:2 a prov:Entity;
    oco:hasUpdateQuery """
        DELETE DATA {
            GRAPH <https://w3id.org/oc/meta/id/> {
                <https://w3id.org/oc/meta/id/062106312420>
                    <http://www.essepuntato.it/2010/06/literalreificati⌋
                    on/hasLiteralValue>
                    "https://doi.org/10.1162/qss_a_00292" . } };
        INSERT DATA {
            GRAPH <https://w3id.org/oc/meta/id/> {
                <https://w3id.org/oc/meta/id/062106312420>
                    <http://www.essepuntato.it/2010/06/literalreificati⌋
                    on/hasLiteralValue>
                    "10.1162/qss_a_00292" . } }"""^^xsd:string;
    dcterms:description "The entity
    'https://w3id.org/oc/meta/id/062106312420' has been
    modified."^^xsd:string;
    prov:generatedAtTime "2024-10-19T19:55:55"^^xsd:dateTime;
    prov:specializationOf id:062106312420;
    prov:wasAttributedTo <https://orcid.org/0000-0002-8420-0696>;
    prov:wasDerivedFrom se:1.

se:1 a prov:Entity;
    dcterms:description "The entity
    'https://w3id.org/oc/meta/id/062106312420' has been
    created."^^xsd:string;
    prov:generatedAtTime "2024-10-10T23:44:45"^^xsd:dateTime;
    prov:hadPrimarySource
    <https://api.crossref.org/works/10.1162/qss_a_00292>;
    prov:invalidatedAtTime "2024-10-19T19:55:55"^^xsd:dateTime;
    prov:specializationOf id:062106312420;
    prov:wasAttributedTo <https://orcid.org/0000-0002-8420-0696>.
```

Listing 11: Usage example of the OpenCitations Data Model, translated in RDF Turtle syntax.

```python
def materialize_version(entity_uri, time_interval):
    # time_interval = (start_time, end_time), either may be None

    # Query provenance for all snapshots, sorted by time descending
    all_snapshots = query_provenance_snapshots(entity_uri)

    # Filter snapshots within the requested time interval
    relevant_snapshots = filter_by_time_interval(all_snapshots,
    time_interval)

    # Get current entity state from dataset
    current_quads = query_dataset(entity_uri)

    # Reconstruct state at each relevant snapshot
    entity_graphs = {}
    for snapshot in relevant_snapshots:
        # Collect update queries from snapshots more recent than this
        one
        update_queries = [s.update_query for s in all_snapshots
                          if s.time > snapshot.time and
                          s.has_update_query]

        # Apply inverse transformations to a copy of the current state
        reconstructed = set(current_quads)
        for query in update_queries:
            apply_inverse_update(reconstructed, query)

        entity_graphs[snapshot.time] = reconstructed

    return entity_graphs
```

Listing 12: Version materialization implementation.

Consider entity `id:062106312420` with literal value "10.1162/qss_a_00292" in the latest snapshot. To reconstruct its state at time t_{n-1}, the algorithm retrieves the update query from snapshot t_n (shown in Listing 11) and applies the inversion process:

The inversion process shown in Listing 13 swaps the operations: what was deleted in the original update (the incorrect DOI URL) is re-inserted, and what was inserted (the corrected DOI) is removed. This restores the entity to its previous state at t_{n-1} where the identifier had value "https://doi.org/10.1162/qss_a_00292".

For complete history reconstruction, the implementation optimizes by computing states incrementally. Starting from the current state, each historical state is derived from its successor by applying only the relevant update query, avoiding redundant recomputation.

The implementation also handles related entities through recursive traversal when configured. Three traversal modes are supported: following object relationships from the main entity, tracking entities merged through `prov:wasDerivedFrom`

```
# Original update query from snapshot t_n:
original_query = """
    DELETE DATA {
        GRAPH <https://w3id.org/oc/meta/id/> {
            <https://w3id.org/oc/meta/id/062106312420>
                <http://www.essepuntato.it/2010/06/literalreification/hasLiteralValue>
                "https://doi.org/10.1162/qss_a_00292" .
        }
    }
    INSERT DATA {
        GRAPH <https://w3id.org/oc/meta/id/> {
            <https://w3id.org/oc/meta/id/062106312420>
                <http://www.essepuntato.it/2010/06/literalreification/hasLiteralValue>
                "10.1162/qss_a_00292" .
        }
    }
"""

# Inversion process:
# 1. Parse the original query to extract operations
# 2. Swap DELETE DATA and INSERT DATA operations:
inverted_operations = """
    INSERT DATA {  # Was DELETE DATA
        GRAPH <https://w3id.org/oc/meta/id/> {
            <https://w3id.org/oc/meta/id/062106312420>
                <http://www.essepuntato.it/2010/06/literalreification/hasLiteralValue>
                "https://doi.org/10.1162/qss_a_00292" .
        }
    }
    DELETE DATA {  # Was INSERT DATA
        GRAPH <https://w3id.org/oc/meta/id/> {
            <https://w3id.org/oc/meta/id/062106312420>
                <http://www.essepuntato.it/2010/06/literalreification/hasLiteralValue>
                "10.1162/qss_a_00292" .
        }
    }
"""

# Result: Entity restored to state at t_{n-1} with value
# "https://doi.org/10.1162/qss_a_00292"
```

Listing 13: Update query inversion process.

relations (where according to the OCDM, when two entities are merged, the surviving entity's snapshot is linked to the pre-deletion snapshot of the merged entity via `prov:wasDerivedFrom` to simplify tracking of entities that were deleted as a result of the merge), and discovering entities that reference the main entity as an object.

Delta materialization between consecutive versions requires no additional computation since OCDM stores changes explicitly as SPARQL UPDATE strings. The

```python
def get_history(entity_uri):
    snapshots = sorted(get_snapshots(entity_uri), reverse=True)
    history = {}

    # Start with current state
    current_graph = query_dataset(entity_uri)
    history[snapshots[0].time] = current_graph

    # Compute each past state from its successor
    for i in range(1, len(snapshots)):
        previous_graph = set(history[snapshots[i-1].time])
        if snapshots[i-1].has_update_query:
            apply_inverse_update(previous_graph,
                                 snapshots[i-1].update_query)
        history[snapshots[i].time] = previous_graph

    return history
```

Listing 14: Incremental history reconstruction.

update query itself represents the delta between consecutive versions, immediately available from the provenance graph without reconstruction. For non-consecutive versions, the delta is computed by materializing both target versions and calculating the set difference between them.

5.4.2 Single-version and cross-version structured queries

Temporal SPARQL queries require selective entity reconstruction and query rewriting to operate on historical data states. The process analyzes standard SPARQL SELECT queries, identifies relevant entities through triple pattern analysis, and executes the query against reconstructed temporal graphs.

The algorithm shown in Listing 15 analyzes SPARQL queries to determine which entities require reconstruction. For each triple pattern in the query, the algorithm identifies the relevant entities, reconstructs their temporal states, and merges them into unified temporal graphs for query execution. Entity discovery combines two sources: the current dataset state, which identifies entities that presently match the pattern, and the provenance update queries, which reveal entities whose historical states matched the pattern but have since changed or been deleted.

For single-version queries, the time interval parameter restricts reconstruction to a specific temporal range. The reconstruction process retrieves only snapshots within the specified range, reducing computational overhead:

Cross-version queries omit the temporal filter, reconstructing complete entity histories. The snapshot alignment process merges entity graphs at each timestamp, creating unified temporal graphs for query execution. Entities present at time t_n

```python
def process_temporal_query(sparql_query, time_interval=None):
    # Parse SPARQL query and extract triple patterns
    triple_patterns = parse_query_patterns(sparql_query)

    # Initialize temporal graph reconstruction
    temporal_graphs = {}
    reconstructed_entities = set()

    # Process each triple pattern to identify relevant entities
    for pattern in triple_patterns:
        if is_isolated_pattern(pattern):
            # Query current dataset for entities matching the pattern
            present_entities = query_current_dataset(pattern)
            # Search provenance update queries for historical matches
            provenance_entities = find_entities_in_updates(pattern)
            relevant_entities = present_entities | provenance_entities
        else:
            # Direct entity identification from URI anchors
            relevant_entities = extract_entity_uris(pattern)

        # Reconstruct entity histories
        for entity_uri in relevant_entities:
            if entity_uri not in reconstructed_entities:
                entity_graphs = reconstruct_entity_timeline(entity_uri,
                time_interval)
                merge_entity_graphs(temporal_graphs, entity_graphs)
                reconstructed_entities.add(entity_uri)

    # Align snapshots temporally and resolve variables
    aligned_graphs = align_temporal_snapshots(temporal_graphs)
    complete_graphs = resolve_query_variables(aligned_graphs,
    triple_patterns)

    # Execute query on temporal graphs
    results = {}
    for timestamp, graph in complete_graphs.items():
        query_result = execute_sparql_on_graph(sparql_query, graph)
        results[timestamp] = extract_query_bindings(query_result)

    return results
```

Listing 15: Temporal query processing for both single-version and cross-version queries.

```python
def reconstruct_entity_timeline(entity_uri, time_interval=None):
    if time_interval:
        # Single-version: reconstruct at specific time interval
        return materialize_version(entity_uri, time_interval)
    else:
        # Cross-version: reconstruct complete entity history
        return get_history(entity_uri)
```

Listing 16: Temporal filtering for single-version and cross-version queries.

but absent at t_{n+1} are propagated forward if they remain unchanged, ensuring query completeness across temporal boundaries.

The algorithm distinguishes between joined and isolated triple patterns to optimize entity discovery. Joined patterns contain at least one URI that transitively connects to other patterns, enabling direct entity identification. Isolated patterns lack such connections, requiring analysis of update queries to discover relevant entities:

Definition 4 (Joined and isolated triple pattern). A triple pattern (s, p, o) is joined if a path exists through shared variables between any variable in the pattern and an IRI anchor point in the query.

Assume there are pairwise disjoint infinite sets I, V, and L (IRIs, Variables, and Literals, respectively), and assume $\text{Path}(n, n_1)$ returns True if there is a route through shared variables in the query graph between the nodes n and n_1.

$\text{Joined}(s, p, o) \Leftrightarrow \exists v \in \{s, p, o\} \cap V, \exists i \in I$ appearing in the query : $\text{Path}(v, i)$, with $(s, p, o) \in (I \cup V) \times (I \cup V) \times (I \cup L \cup V)$.

Conversely, $\text{Isolated}(s, p, o) \Leftrightarrow \neg\text{Joined}(s, p, o)$, indicating that no path exists between the subject variable and any subject IRI in the query.

Joined patterns enable variable resolution through previously reconstructed entity graphs, while isolated patterns require searching through update queries to discover relevant entities.

Consider the following example of a joined pattern query:

```
PREFIX literal: <http://www.essepuntato.it/2010/06/literalreification/>
PREFIX datacite: <http://purl.org/spar/datacite/>
SELECT ?br ?id ?value
WHERE {
    <https://w3id.org/oc/meta/br/062104388184> datacite:hasIdentifier
    ?id.
    ?br datacite:hasIdentifier ?id.
    ?id literal:hasLiteralValue ?value.
}
```

Listing 17: Example of a SPARQL query containing only joined triple patterns.

The URI `br:062104388184` provides an anchor point for variable resolution. After materializing all versions of this entity, the algorithm can resolve `?id` from the identifier relationships, then resolve `?br` and `?value` from the discovered identifiers. Each variable may take multiple values across different temporal states.

For entities present in the current dataset, a standard SPARQL query matching the triple pattern identifies them directly. Additionally, isolated patterns require searching through stored update queries to discover entities whose historical states

matched the pattern. The algorithm employs text search capabilities when available, falling back to pattern matching for standard endpoints:

```python
def find_entities_in_updates(triple_pattern):
    # Extract URIs and predicates from the triple pattern
    pattern_elements = extract_pattern_elements(triple_pattern)

    # Search for update queries containing pattern elements
    if has_full_text_search():
        # Use optimized full-text search
        matching_queries = search_updates_fulltext(pattern_elements)
    else:
        # Fallback to pattern-based search
        matching_queries = search_updates_contains(pattern_elements)

    # Parse update queries to extract relevant entity URIs
    relevant_entities = set()
    for update_query_string in matching_queries:
        parsed_query = parse_sparql_update(update_query_string)

        # Extract entities from INSERT/DELETE operations
        for operation in parsed_query.operations:
            operation_triples = extract_operation_triples(operation)
            for op_triple in operation_triples:
                if pattern_matches_triple(triple_pattern, op_triple):
                    entity_uri = extract_subject_uri(op_triple)
                    relevant_entities.add(entity_uri)

    return relevant_entities
```

Listing 18: Entity discovery through update query analysis.

Consider a query to find all identifiers whose literal value has ever contained a DOI URL:

```
PREFIX literal: <http://www.essepuntato.it/2010/06/literalreification/>
SELECT ?literal
WHERE {
    ?id literal:hasLiteralValue ?literal.
    FILTER REGEX(?literal, "^https://doi\\.org/")
}
```

Listing 19: Example of a SPARQL query containing an isolated triple pattern.

The isolated pattern `?id literal:hasLiteralValue ?literal` lacks connection to any URI. The algorithm searches for update queries containing the predicate `literal:hasLiteralValue`, parses them to extract subject-object pairs, and reconstructs all entities that have used this predicate. The FILTER clause is then applied during query execution on the reconstructed temporal graphs.

A query can contain both joined and isolated triple patterns. In this case, the

isolated patterns are processed by conducting textual searches on the stored update queries. In contrast, the joined patterns are resolved by recursively explicating the variables through entity relationships.

After identifying the relevant entities for the query, the reconstruction strategy depends on whether it is a single-version or a cross-version query. For single-version queries, reconstruction focuses only on the specified temporal interval for efficiency. For cross-version queries, complete entity histories must be restored. In both cases, the version materialization process described in the previous section provides the foundation.

However, query execution requires additional preprocessing beyond entity reconstruction. Restored snapshots must be aligned to create a complete temporal picture. Since update queries only record changes, an entity modified at time t_n but unchanged at t_{n+1} will appear in the t_n delta but not in the t_{n+1} snapshot. The t_{n+1} graph would lack that entity despite its continued existence. The alignment process addresses this by propagating unchanged entities from t_n to subsequent snapshots where they remain unmodified. Finally, entity graphs are merged by timestamp to ensure contemporary information appears in unified temporal graphs.

After completing the preprocessing steps, executing the time-traversal query becomes straightforward. The algorithm applies the query to each reconstructed temporal graph, where each graph corresponds to a specific snapshot and contains precisely the information necessary to satisfy the query requirements.

5.4.3 Single-delta and cross-delta structured queries

Delta structured queries track entity modifications across temporal states. These queries analyze entity creation, modification, and deletion events by examining SPARQL update queries stored in the provenance metadata. The approach supports both single-delta queries (changes within a specific time interval) and cross-delta queries (changes across the entire dataset history).

Delta queries operate through a two-phase process: entity discovery and change analysis. The entity discovery phase uses the same mechanisms as version queries to identify relevant entities, while the change analysis phase examines provenance snapshots to extract temporal modification patterns.

The algorithm leverages the OCDM change-based policy where deltas are explicitly stored as SPARQL update query strings. This eliminates the need to compute differences between temporal states, as changes are directly available from the provenance metadata.

The change analysis phase extracts temporal modification information from en-

```python
def process_delta_query(sparql_query, time_interval=None,
changed_properties=set()):
    # Phase 1: Entity Discovery
    triple_patterns = parse_query_patterns(sparql_query)
    reconstructed_entities = set()

    for pattern in triple_patterns:
        if is_isolated_pattern(pattern):
            # Query current dataset and provenance update queries
            present_entities = query_current_dataset(pattern)
            provenance_entities = find_entities_in_updates(pattern)
            reconstructed_entities.update(present_entities |
            provenance_entities)
        else:
            # Direct entity identification from URI anchors
            entity_uri = extract_entity_uri(pattern)
            reconstructed_entities.add(entity_uri)

    # Phase 2: Change Analysis
    delta_results = {}
    for entity_uri in reconstructed_entities:
        entity_changes = identify_changed_entity(entity_uri,
                                                 time_interval,
                                                 changed_properties)
        if entity_changes:
            delta_results.update(entity_changes)

    return delta_results
```

Listing 20: Delta query processing implementation.

tity snapshots. For each identified entity, the algorithm retrieves all associated snapshots and analyzes their temporal sequence to determine creation, modification, and deletion events.

The change identification process distinguishes between three types of temporal events. Creation events correspond to the first snapshot of an entity, identified by the earliest `prov:generatedAtTime` value. Modification events appear in subsequent snapshots containing `oco:hasUpdateQuery` values, representing SPARQL UPDATE operations that altered the entity state. Deletion events are inferred when an entity has snapshots in the provenance graph but no longer exists in the current dataset.

5.5 Implementation and evaluation

This methodology was implemented in a Python package, time-agnostic-library (Massari & Peroni, 2025b), distributed as open-source software under the ISC license. It makes three main classes available: `AgnosticEntity`, `VersionQuery`, and `DeltaQuery`, for materializations, version queries, and delta queries. All three op-

```
def identify_changed_entity(entity_uri, time_interval,
changed_properties):
    # Retrieve all snapshots for the entity
    query = f"""
        SELECT ?time ?updateQuery
        WHERE {{
            ?snapshot prov:specializationOf <{entity_uri}>;
                      prov:generatedAtTime ?time.
            OPTIONAL {{
                ?snapshot oco:hasUpdateQuery ?updateQuery.
            }}
        }}
    """

    snapshots = execute_sparql_query(query)
    if not snapshots:
        return {}

    # Filter by time interval if specified
    if time_interval:
        snapshots = filter_by_time_interval(snapshots, time_interval)

    # Sort snapshots chronologically
    sorted_snapshots = sort_by_timestamp(snapshots)

    # Initialize change structure
    change_info = {
        "created": convert_to_datetime(sorted_snapshots[0]["time"]),
        "modified": {},
        "deleted": None
    }

    # Check current existence
    if not entity_exists_currently(entity_uri):
        change_info["deleted"] =
        convert_to_datetime(sorted_snapshots[-1]["time"])

    # Process modification snapshots
    for snapshot in sorted_snapshots[1:]:  # Skip creation snapshot
        timestamp = convert_to_datetime(snapshot["time"])
        update_query = snapshot.get("updateQuery")

        if update_query and changed_properties:
            # Filter by specific changed properties
            if any(prop in update_query for prop in changed_properties):
                change_info["modified"][timestamp] = update_query
        elif update_query and not changed_properties:
            # Include all modifications
            change_info["modified"][timestamp] = update_query

    return {str(entity_uri): change_info}
```

Listing 21: Entity change identification algorithm.

erations can be performed over the entire history available or by specifying a time interval via a tuple in the form `(START, END)`. In this way, each of the six retrieval functionalities considered in the taxonomy can be accomplished.

The package was tested on Blazegraph, GraphDB, Apache Jena Fuseki, OpenLink Virtuoso, and QLever, and it is fully compatible with these triplestores. In addition, time-agnostic-library provides configuration parameters to take advantage of the full-text indexing systems integrated into each of the mentioned databases to achieve instant text searches.

Test-Driven Development (TDD) (Beck, 2003) was adopted as a software development process. TDD follows a cycle where tests are written before implementation code: developers first create a failing test that specifies desired behavior, then write minimal code to pass the test, and finally refactor while maintaining test coverage. This methodology achieves 100% line coverage, ensuring every code path is exercised. However, line coverage does not guarantee coverage of all possible program states. To address this limitation, integration tests complement the TDD test suite by exercising the system under realistic usage scenarios and stress conditions, testing state combinations that unit tests may not reach. The benchmark evaluation presented in the following section provides an example of this approach, as it exercises the library against a real versioned dataset with diverse query patterns.

5.5.1 Benchmark dataset and setup

The evaluation uses BEAR-B-daily (Fernández et al., 2016), a benchmark for versioned RDF systems adopted by subsequent work on RDF archiving (Cerdeira-Pena et al., 2016; Pelgrin, Taelman, Galárraga, & Hose, 2025; Taelman et al., 2018). BEAR-B-daily consists of 89 consecutive daily snapshots of DBpedia Live, describing the 100 most volatile resources. The dataset grows from 33,502 triples in version 0 to 43,907 triples in version 57, with 83,134 distinct triples across all versions. The benchmark defines 62 query patterns: 49 with a known predicate (`?P?` patterns, such as `?s rdfs:label ?o`) and 13 with a known predicate and object (`?PO` patterns, such as `?s rdf:type dbo:Film`). Three query types are evaluated following the taxonomy defined earlier in this chapter:

- Version materialization (VM): retrieve the triples matching a pattern at a specific version
- Delta materialization (DM): retrieve the triples added or removed between two versions for a given pattern

- Version query (VQ): retrieve the triples matching a pattern across all versions

For the Time Agnostic Library, the BEAR-B-daily data was converted to the OCDM provenance format. Among the four formats distributed by BEAR-B (IC, CB, TB, and CBTB, corresponding to the archiving policies described in Section 5.2), IC and CB were considered as input for the conversion. A reusable conversion module was developed and integrated in the library to produce OCDM-compliant N-Quads from either IC or CB input. CB conversion is faster than IC because it uses the deltas directly instead of computing diffs between consecutive snapshots (0.4 seconds versus 6.6 seconds on BEAR-B-daily). However, the CB deltas distributed by BEAR are inconsistent with the IC snapshots: the expected query results published by BEAR can only be reproduced when computing diffs from the IC files, not when using the CB deltas directly. The IC strategy was therefore used for the OCDM conversion. The resulting N-Quads files were indexed in QLever (Bast & Buchhold, 2017). The total preprocessing time, including OCDM conversion and QLever indexing, was 8.6 seconds with the IC strategy (2.4 seconds with CB). The benchmark executed VM queries at all 89 versions for each of the 62 patterns (5,518 queries), DM queries at 12 version pairs for each pattern (744 queries), where each pair starts from version 0 and ends at versions spaced 5 apart (0–5, 0–10, . . . , 0–55) plus the final version (0–88), and VQ queries once per pattern (62 queries). Each query was repeated 5 times.

For comparison, OSTRICH was executed on the same hardware using its Docker container built from the branch of the source repository that implements multi-snapshot strategies (Pelgrin et al., 2025). The same 89 versions were ingested using the `interval 5` strategy, which creates a new HDT snapshot every 5 versions. This configuration was selected because it yields the fastest ingestion time for BEAR-B-daily among all strategies evaluated by Pelgrin et al. (2025), completing in 10.7 seconds. OSTRICH queries used the same 62 patterns with 5 replications.

All experiments were conducted on a machine with an Intel Core i9-12900K (24 cores), 128 GB DDR RAM, and SSD storage, running Arch Linux with kernel 6.18.3.

5.5.2 Results

Table 5.1 presents the execution times for the Time Agnostic Library. VM queries average 83.89 ms, with `?PO` patterns (30.04 ms) being approximately 3 times faster than `?P?` patterns (98.18 ms), since constraining both predicate and object reduces the number of entities to materialize. DM queries are faster at 29.54 ms on average, as they identify changes between two consecutive versions rather than reconstructing

a full state. VQ queries average 138.48 ms because they materialize all 89 versions for every entity matching the pattern.

Table 5.1: Time Agnostic Library execution times on BEAR-B-daily.

Query type	Pattern	Count	Mean (ms)	Median (ms)
VM	?P?	4,361	98.18	48.46
	?PO	1,157	30.04	23.52
	All	5,518	83.89	42.57
DM	?P?	588	32.90	22.67
	?PO	156	16.88	15.39
	All	744	29.54	21.48
VQ	?P?	49	163.53	86.02
	?PO	13	44.06	36.52
	All	62	138.48	78.47

Table 5.2 compares the Time Agnostic Library with OSTRICH, both measured on the same hardware. The Time Agnostic Library's IC preprocessing completes in 8.6 seconds and the CB strategy in 2.4 seconds, both faster than OSTRICH's ingestion (10.7 seconds). These preprocessing times apply only to the benchmark scenario, where non-OCDM data must be converted: the Time Agnostic Library is designed for live systems that already store provenance in OCDM format, requiring no conversion phase. Query execution shows the opposite pattern: OSTRICH is approximately three orders of magnitude faster across all query types. This gap reflects a difference in architecture. OSTRICH preprocesses all version data into a dedicated C++ index optimized for temporal lookups, resolving queries through direct index access. The Time Agnostic Library operates on data stored in a standard SPARQL endpoint, reconstructing historical states at query time by retrieving provenance snapshots and applying inverse SPARQL UPDATE operations. VQ queries exhibit the largest gap because OSTRICH resolves them through a single index traversal, while the Time Agnostic Library must materialize each of the 89 versions independently. Figures 5.2 and 5.3 show the per-version execution times on a logarithmic scale: both systems exhibit stable performance across versions, with the gap remaining constant at approximately three orders of magnitude. Figure 5.4 shows the same gap for VQ queries.

Table 5.2: Performance comparison on BEAR-B-daily, measured on the same hardware. Query times are mean values.

System	Strategy	Ingestion (s)	VM (ms)	DM (ms)	VQ (ms)
OSTRICH	interval 5	10.7	0.09	0.10	0.12
TAL	IC	8.6	83.89	29.54	138
	CB	2.4		—	

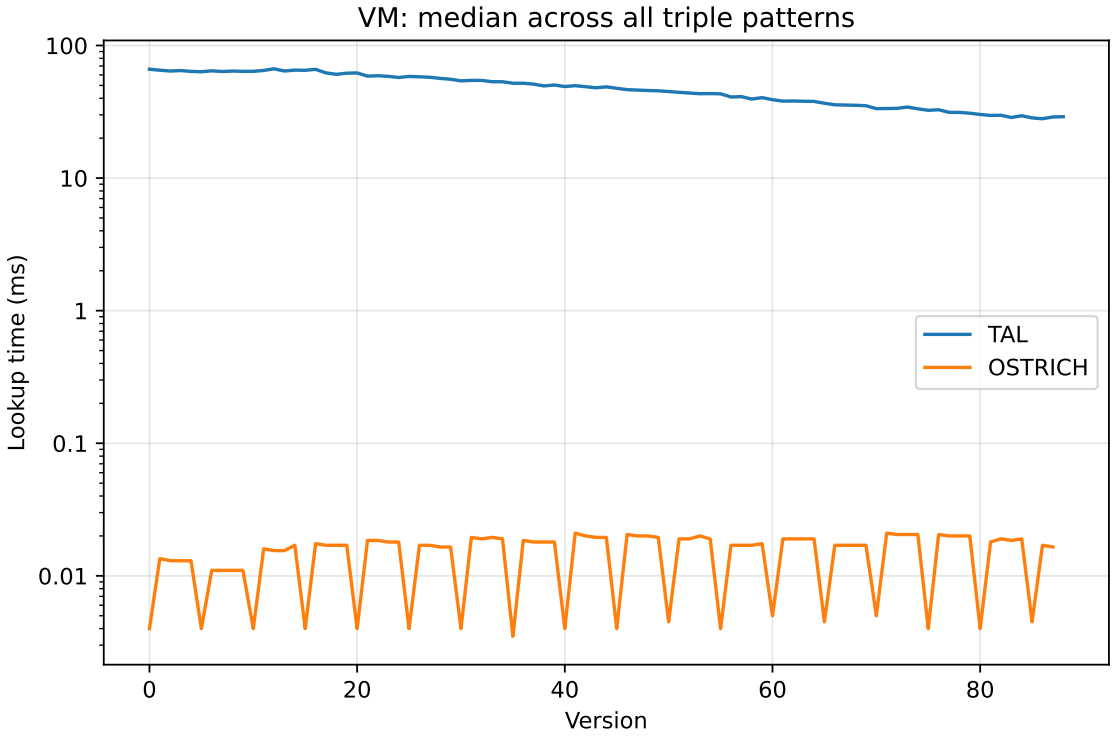


Figure 5.2: Version materialization: median execution time per version across all 62 triple patterns (log scale).

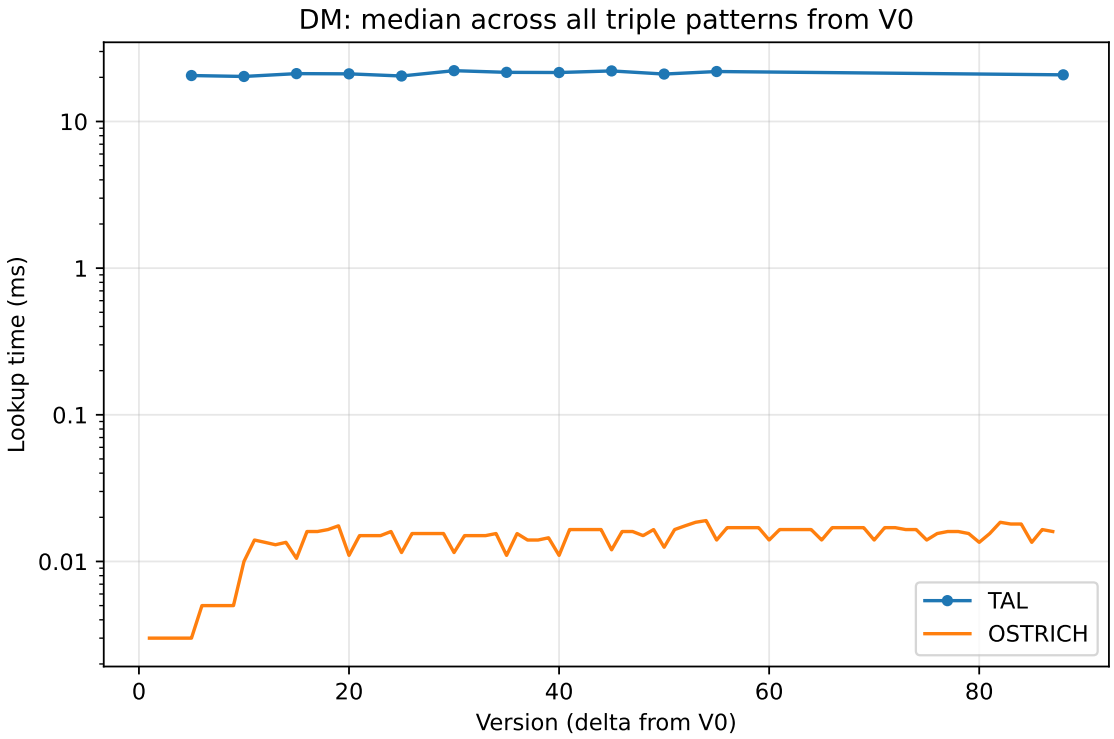


Figure 5.3: Delta materialization: median execution time per version pair from V0 across all 62 triple patterns (log scale).

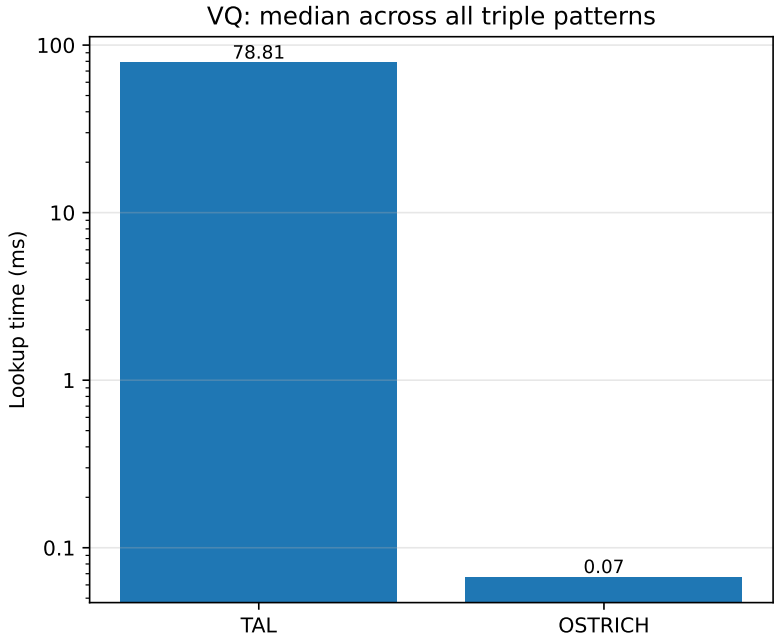


Figure 5.4: Version query: median execution time across all 62 triple patterns (log scale).

5.5.3 Discussion

The two systems address different requirements and operate under different constraints.

OSTRICH requires an offline ingestion phase during which all version data is preprocessed into a dedicated storage engine. As shown in Table 5.2, both of the Time Agnostic Library's conversion strategies are faster than OSTRICH's ingestion on BEAR-B-daily. Furthermore, OSTRICH does not expose a standard SPARQL endpoint and does not support SPARQL UPDATE operations. This is an architectural constraint: snapshots are stored as HDT files, which are immutable binary structures with a fixed dictionary encoding, and intermediate versions are represented as aggregated changesets relative to these snapshots. Ingestion requires all version data to be provided as N-Triples changeset files before processing begins, and the store cannot be queried during ingestion. There is no interface for inserting or deleting individual triples after the store has been built.

As noted in the previous section, the preprocessing times apply only to the benchmark scenario. In production, the Time Agnostic Library operates directly on data already stored in OCDM format in any SPARQL-compliant triplestore, requiring no ingestion phase. Systems that natively produce OCDM-compliant provenance, such as HERITRACE (Massari & Peroni, 2025a), generate queryable temporal data as part of their normal workflow: updates applied through standard SPARQL UPDATE operations produce new provenance snapshots that become immediately queryable. The library is compatible with any triplestore supporting SPARQL 1.1, such as Blazegraph, GraphDB, Fuseki, Virtuoso, and QLever.

The two systems also differ in query coverage. The Time Agnostic Library supports all six temporal retrieval functionalities defined by Fernández et al. (2016): version materialization, single-version queries, cross-version queries, delta materialization, single-delta queries, and cross-delta queries, with full SPARQL SELECT support. OSTRICH natively supports VM, DM, and VQ on single triple patterns.

The choice between these approaches depends on the application requirements. For read-only archives where all data is available before querying and sub-millisecond latency is required, preprocessing-based systems such as OSTRICH are appropriate. For applications requiring live data updates, compatibility with existing SPARQL infrastructure, or full temporal query coverage, the Time Agnostic Library provides these capabilities at the cost of higher query latency. The latencies are nonetheless acceptable for interactive applications: VM queries average 83.89 ms and DM queries average 29.54 ms, which are the operations required to reconstruct entity states and display change histories. VQ queries complete in 138.48 ms on average over the 89

versions of the benchmark dataset. In the context of this thesis, the goal of building an RDF editor that allows domain experts to curate data and inspect its change history in real time makes OSTRICH unsuitable: an editor must be able to insert and delete individual triples through standard SPARQL UPDATE operations, with each change immediately generating queryable provenance metadata. OSTRICH's batch ingestion model, which requires complete changeset files and rebuilds internal index structures for each version, is incompatible with this incremental, interactive workflow. The Time Agnostic Library satisfies these requirements, and its latencies are acceptable for interactive use.

Chapter 6

HERITRACE: a user-centered approach to semantic data curation

The preceding chapters have established the foundation for addressing systematic barriers in semantic data curation. Chapter 3 documented empirical evidence from OpenCitations Meta and ParaText Bibliographical Database, identifying five convergent requirements: provenance management, change tracking and version restoration, domain expert usability, flexible customization, and integration with existing RDF collections. Chapter 4 concluded that the OpenCitations Data Model provides the necessary capabilities for documenting sources, responsible agents, and version histories through snapshot-based mechanisms. Chapter 5 addressed temporal querying methodologies through the Time-Agnostic Library implementation.

This chapter presents HERITRACE (Heritage Enhanced Repository Interface for Tracing, Research, Archival Curation, and Engagement) as an integrated framework that synthesizes these technical foundations to address the empirically identified barriers, building upon and extending the discussion by Massari and Peroni (2025a). The software is released as open source under the ISC license to enable maximum reusability (Massari, 2025c). ParaText provided the environment for guerrilla testing (Nielsen, 1993) with domain experts, whose feedback drove iterative refinement of the system.

6.1 Architecture and design principles

The design of HERITRACE responds directly to the five requirements identified in Chapter 3. The system implements a layer of abstraction that presents semantic

data through form-based interfaces corresponding to domain experts' existing mental models of bibliographic or cultural heritage data, without requiring knowledge of RDF syntax or SPARQL query construction. While the underlying RDF structure is preserved, users interact with metadata through familiar interface patterns that hide the technical complexity of Semantic Web technologies.

Provenance tracking is embedded into every data operation rather than treated as an optional feature. The system automatically records when modifications occur, which agents perform them, and what primary sources inform the changes. This integrated approach eliminates the additional workflow steps that typically discourage provenance documentation in manual curation processes.

Change tracking follows a non-destructive, snapshot-based approach where every modification generates a new version while preserving access to all previous states. This enables both error recovery and historical analysis of curatorial decisions without requiring additional backup procedures.

HERITRACE uses Shapes Constraint Language (SHACL) (Knublauch & Kontokostas, 2017), a W3C standard for data model specification, and YAML Ain't Markup Language (YAML) (Ben-Kiki, Evans, & Ingerson, 2009), a human-readable serialization language, for customization. This enables technical staff to configure the system without learning proprietary syntaxes, reducing deployment barriers while keeping customizations maintainable.

Finally, HERITRACE connects directly to existing RDF collections stored in triple stores or other semantic repositories without requiring data migration or format conversion. The system discovers existing entities through their RDF types and applies its curation capabilities to all subsequent modifications, preserving institutional investments in current infrastructure.

The resulting system is a web-based application that supports cross-platform access through standard browsers. The system authenticates users through ORCID (Haak, Fenner, Paglione, Pentz, & Ratner, 2012) integration, leveraging existing scholarly identity infrastructure for access control.

6.1.1 Technical architecture

HERITRACE implements a layered architecture (Richards & Ford, 2020). Components are grouped by technical capability, providing clear separation of responsibilities and facilitating independent testing and modification.

Figure 6.1 illustrates the container architecture of HERITRACE following the C4 model notation. The architecture distinguishes two categories of users with distinct interaction patterns. GLAM domain experts interact with the system through

HTTPS to edit metadata, representing the primary operational user base. Administrators configure the system through file-based mechanisms, managing SHACL shapes, YAML display rules, and environment variables that control system behavior.

The HERITRACE system boundary contains four container components. The web application container, implemented in Python using the Flask framework, manages semantic data through SHACL validation, provenance recording, and change tracking capabilities. This container serves as the central coordination point, mediating all interactions between users and data stores. The semantic datastore container stores the entities managed by the system and supports both triplestores and quadstores. The provenance datastore container maintains the complete history of changes to entities through snapshot-based versioning; since the provenance data model relies on named graphs, this container requires a quadstore. The two datastores can be configured as separate databases or as a single shared instance. The state store container, implemented using Redis (Sanfilippo, 2009), handles concurrent edit locks to prevent conflicting modifications, initialization state to track system setup, and URI counters to generate unique identifiers for newly created entities.

The web application container communicates with the semantic and provenance datastores through SPARQL queries, reading and writing entities in the former while recording change history in the latter. The web application authenticates users by integrating with ORCID (Haak et al., 2012), an external system that provides scholarly identity verification through the OAuth 2.0 protocol.

The layered architecture organizes the web application into four distinct layers: presentation, business, persistence, and database. Each layer maintains specific responsibilities and communicates exclusively with adjacent layers, implementing a closed-layer pattern (Richards & Ford, 2020) that enforces isolation. This architectural decision ensures that modifications to one layer do not impact other layers.

The presentation layer handles all HTTP communication and user interface rendering. This layer receives requests from browsers, manages session state, processes form submissions, and generates HTML responses. The presentation layer does not contain business logic or database access code, instead delegating all data operations to the business layer. This separation enables the presentation layer to focus exclusively on user interaction concerns such as input validation, error display, and response formatting.

The business layer implements the logic of the system, containing components responsible for access control, RDF entity management, and SHACL validation. The

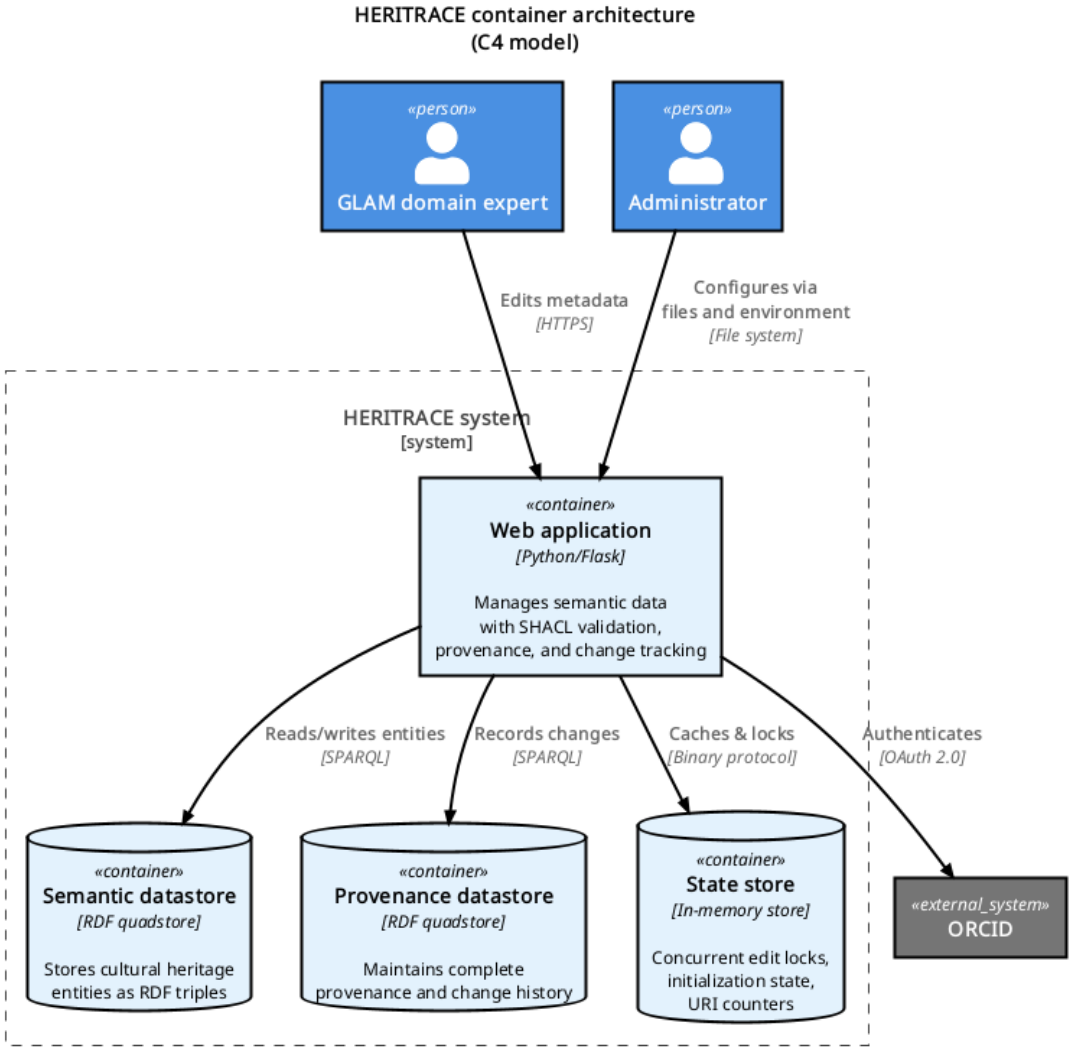


Figure 6.1: HERITRACE container architecture following the C4 model notation, showing the web application and its interactions with data stores and external authentication services.

access control component verifies user permissions before allowing modifications to proceed, ensuring that only authorized agents can edit specific resources. The RDF manager component handles entity creation, modification, and deletion operations, translating user actions into appropriate RDF operations while maintaining semantic consistency. The validator component applies SHACL shapes to incoming data, enforcing structural constraints and data type requirements before persistence occurs. The business layer receives requests from the presentation layer, performs necessary validations and transformations, and delegates data access operations to the persistence layer.

The persistence layer provides an abstraction over database operations, containing the query builder component that constructs SPARQL queries based on high-

level operation requests from the business layer. This layer translates generic data access requests into specific SPARQL queries appropriate for the underlying RDF quadstores, handling details such as named graph selection, triple pattern construction, and result set processing. The persistence layer isolates the business layer from database-specific implementations, enabling potential replacement of the underlying database technology without requiring modifications to business logic.

Figure 6.2 illustrates the flow of an entity creation request through these layers.

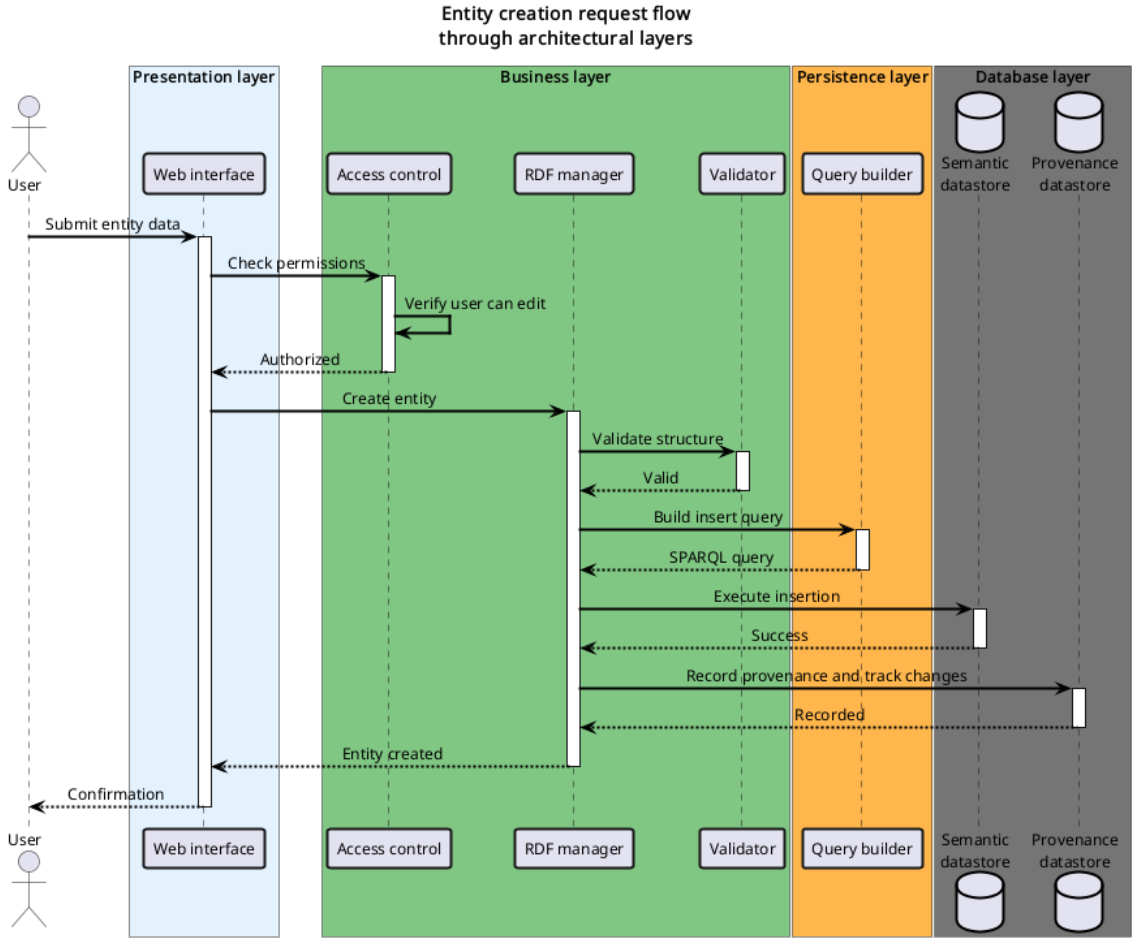


Figure 6.2: Entity creation request flow through HERITRACE architectural layers, demonstrating the closed-layer pattern where requests traverse all layers sequentially.

6.2 User interface design and catalog functionality

The user interface design of HERITRACE prioritizes intuitive navigation and familiar interaction patterns for domain experts in galleries, libraries, archives, and museums. The interface presents semantic data through recognizable organizational structures while maintaining the underlying RDF relationships that enable computational analysis and interoperability. While HERITRACE is not primarily engineered as a search tool, unlike specialized systems such as OSCAR (Heibi, Peroni, & Shotton, 2019), it provides a basic interface for visualizing the contents of a dataset.

The catalog interface implements a two-panel layout. The left panel displays a "Categories" list showing all available entity types in the dataset, with each category indicating the total number of items it contains. This organizational approach allows users to understand the scope and composition of their collection at a glance. For bibliographic collections, categories might include "Journal Article" (114 items), "Issue" (36 items), "Journal" (44 items), and similar publication types, as demonstrated in the ParaText deployment (Figure 6.3).

The right panel presents items within the selected category through a browsable list interface. Users can control the presentation through sorting options that order items by different properties, adjustable pagination that determines the number of displayed results per page, and navigation controls that enable browsing through large collections. Each item appears as a clickable link that provides access to detailed metadata editing capabilities.

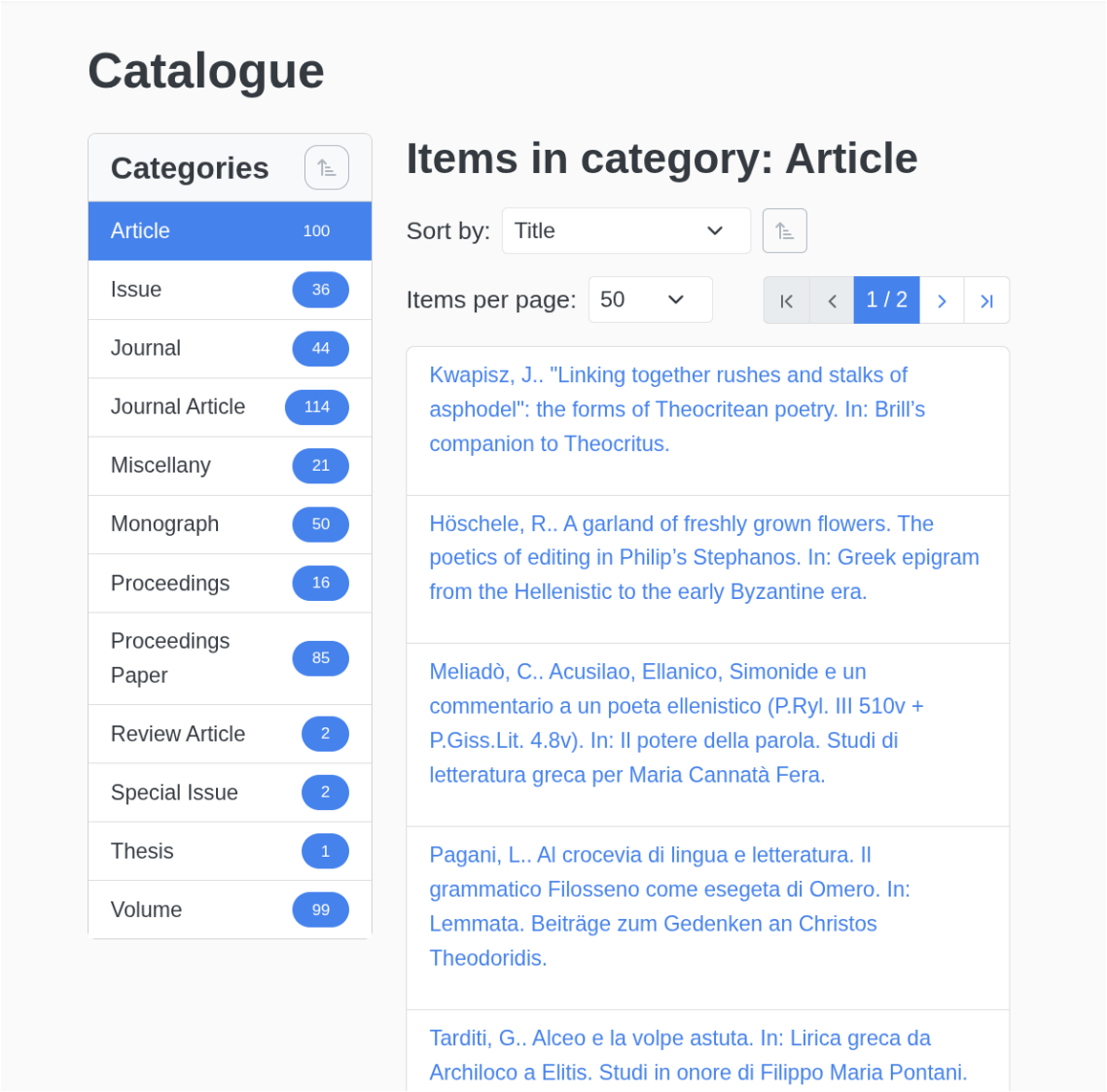


Figure 6.3: HERITRACE catalog interface showing the two-panel layout with categories on the left and browsable item lists on the right.

When users select an item from the catalog, the interface transitions to a detailed editing view that presents all available metadata fields for the resource. This view displays each property with its current value through appropriate input controls: text fields for titles and descriptions, dropdown menus for controlled vocabularies, date selectors for temporal information, and specialized inputs for identifiers and URIs. The interface dynamically adjusts the available fields based on the entity type, ensuring that only relevant properties appear for each kind of resource.

The editing interface implements inline modification capabilities that enable immediate updates to metadata values. Real-time validation provides feedback on data consistency and format requirements without requiring form submission. This approach reduces the cognitive overhead of metadata editing while preserving data quality through immediate error detection and correction guidance.

Action controls at the top of the editing interface provide access to key operations. Users can cancel modifications without saving changes, delete resources when necessary, or access the "Time Machine" functionality for version management. The interface also enables metadata expansion through "add field" capabilities that allow users to include additional properties not initially displayed for the entity.

The system presents technical identifiers and URIs in human-readable formats while preserving the underlying semantic relationships. For example, instead of displaying `http://purl.org/spar/fabio/JournalArticle` as a type designation, the interface shows "Journal Article" as configured through YAML display rules. This translation layer enables domain experts to work with familiar terminology while ensuring that all modifications maintain proper semantic annotations, as shown in Figure 6.4.

The editing interface includes a dedicated section for visualizing incoming relationships through the "Resources Referencing This" functionality. This feature displays all entities in the collection that link to the current resource, enabling users to understand reverse dependencies and navigate the semantic graph from the perspective of referenced entities, as illustrated in Figure 6.5.

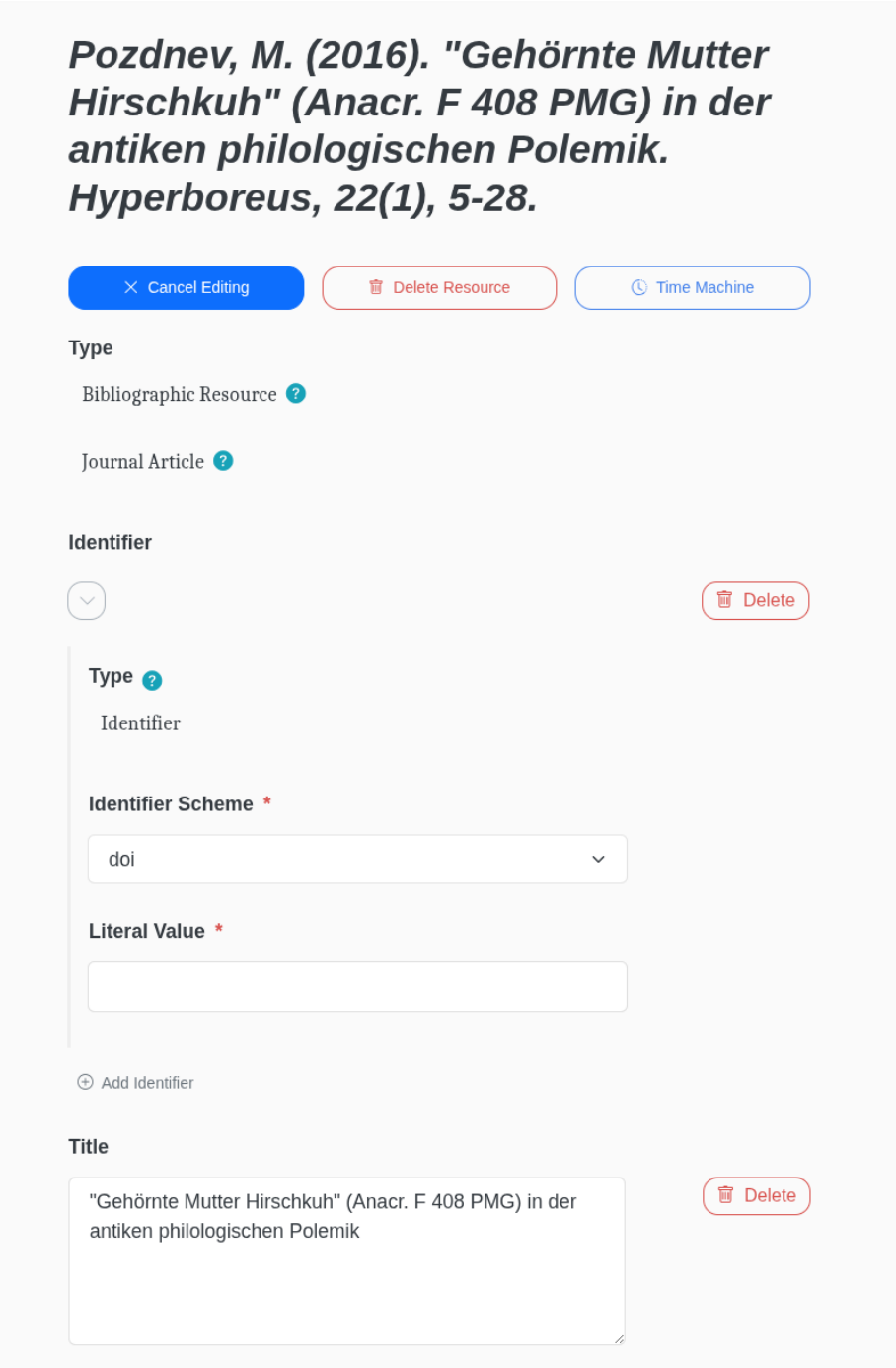


Figure 6.4: HERITRACE editing interface showing metadata fields, input controls, and action buttons for a bibliographic resource.

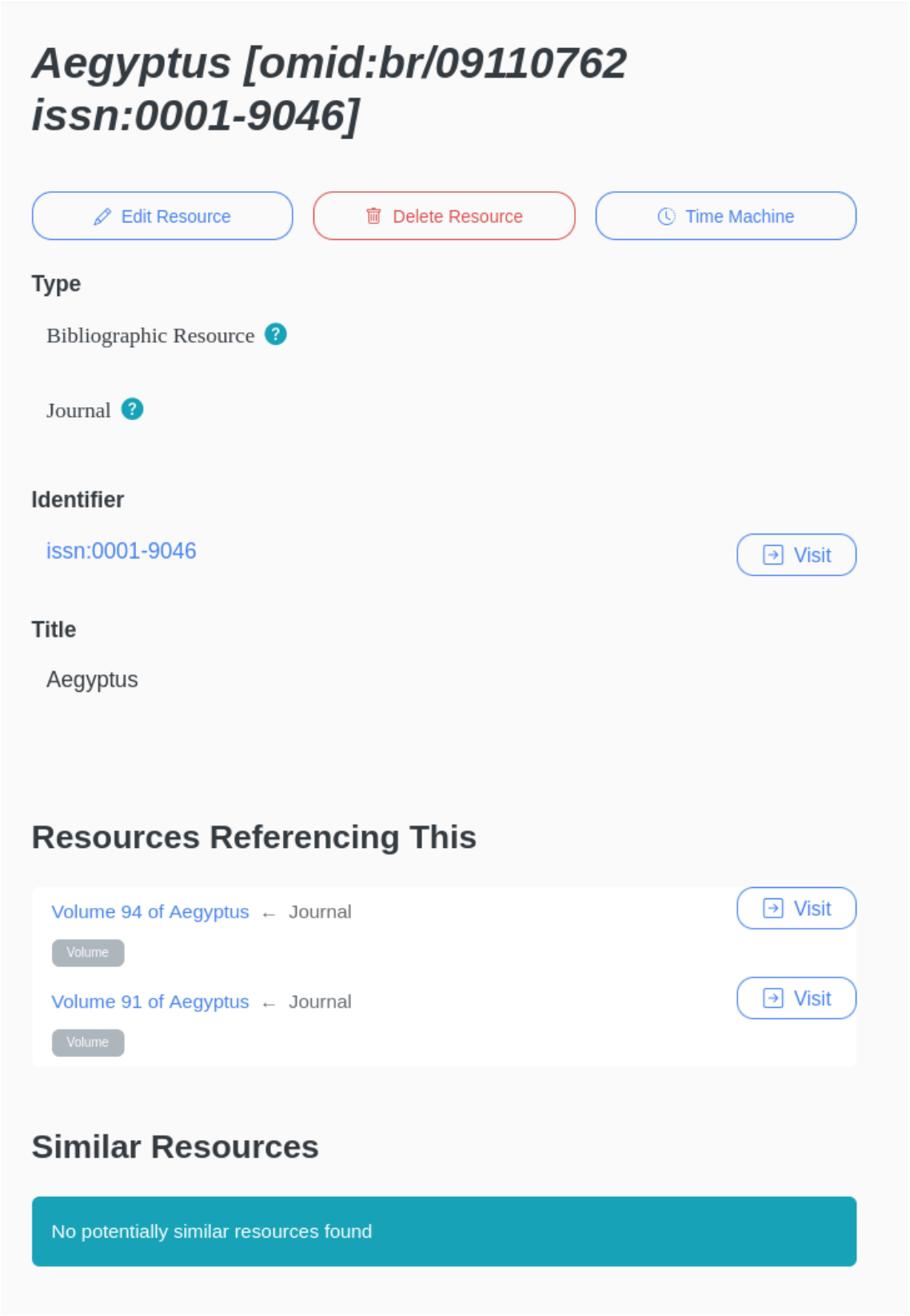


Figure 6.5: HERITRACE interface showing resources that reference the current entity, enabling navigation through reverse relationships in the semantic graph.

6.3 Time machine and change tracking interface

The "Time Machine" functionality provides users with temporal navigation capabilities that make the technical complexity of versioned RDF data available through visual timeline interfaces.

The timeline interface presents the complete modification history of each entity as a chronological sequence of snapshots. Each entry in the timeline captures key information about the state change: the timestamp when the modification occurred, the agent responsible for the change, the primary data source that informed the modification, a description of the entity at that point in time, and a detailed record of specific changes made.

Users can navigate between different versions of an entity through the timeline interface, comparing current and previous states to understand how the metadata has evolved. The system presents changes in human-readable formats, showing additions and deletions in context rather than requiring interpretation of technical SPARQL queries. For bibliographic resources, this might reveal when author information was corrected, how publication dates were refined, or why certain identifiers were added or removed, as demonstrated in Figure 6.6.

The restoration functionality enables users to revert entities to previous states when current versions contain errors or when earlier information proves more accurate. When a user selects a previous snapshot for restoration, HERITRACE automatically updates not only the primary entity but also all related resources to maintain consistency across the dataset. For example, if a bibliographic record is restored to a version with different associated identifiers, the system ensures that identifier records and any dependent relationships are also reverted to compatible states.

The interface enables detailed examination of any historical version, where both the primary entity and all related entities are reconstructed in real-time for that specific temporal point. Users can transition between different version details, facilitating direct comparison of how the entity and its relationships appeared at different moments in time. This reconstruction capability ensures that users see not just isolated changes but the complete contextual state of the data as it existed at each historical point.

The complementary "Time Vault" functionality extends the temporal management capabilities to deleted entities. When resources are removed from the main catalog, they remain available through the Time Vault interface, which serves as a specialized view for deleted items. Users can review information about deleted resources, including deletion timestamps and responsible agents, and can restore

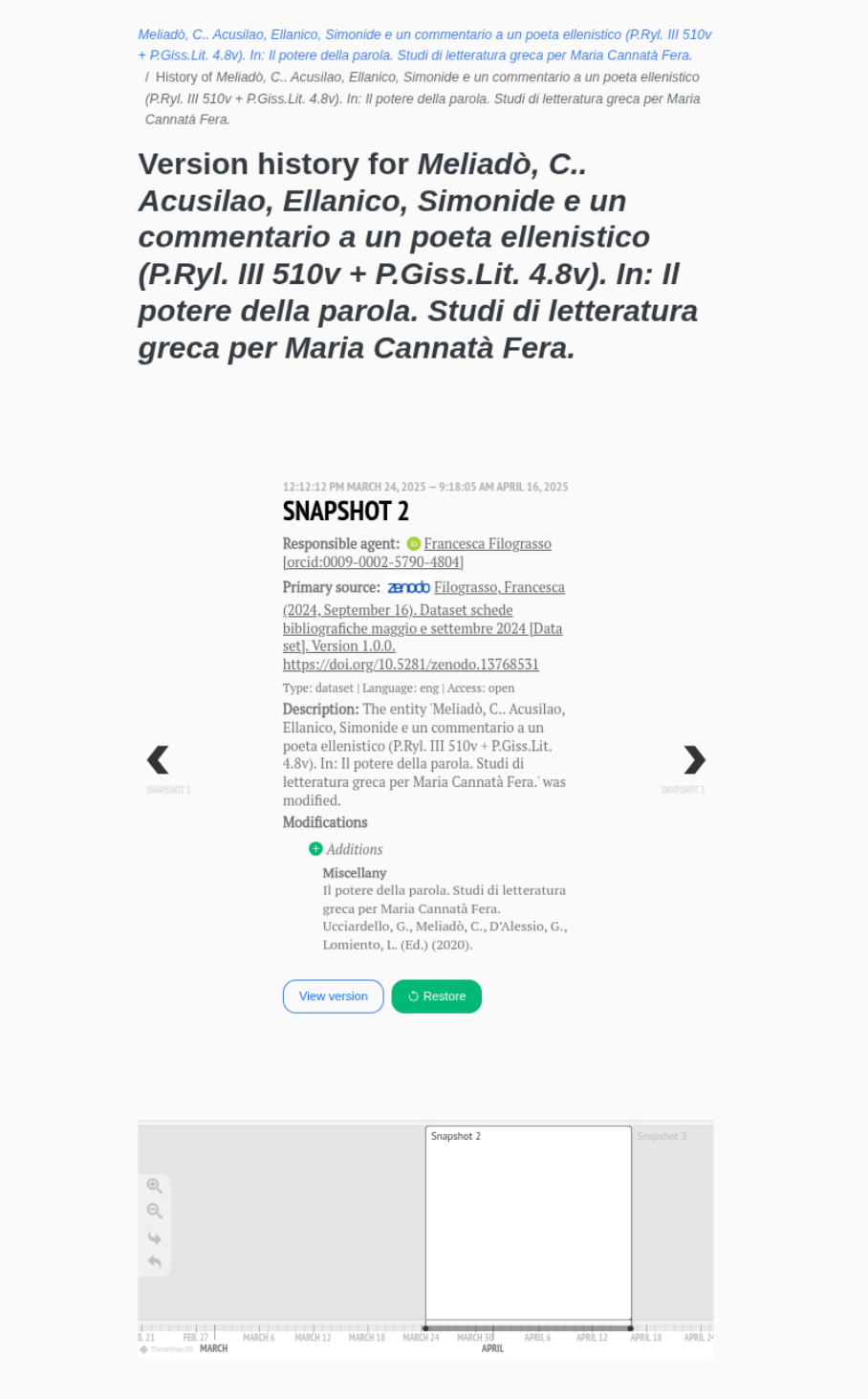


Figure 6.6: HERITRACE Time Machine interface displaying version history with snapshot details, responsible agents, primary sources, and timeline navigation for a bibliographic resource.

deleted entities when necessary. This approach ensures that no data loss occurs due to curatorial errors while maintaining clean primary interfaces that focus on active

collection items. Figure 6.7 illustrates the Time Vault interface, showing deleted resources organized by category with options to view the last valid version or restore items to the active catalog.

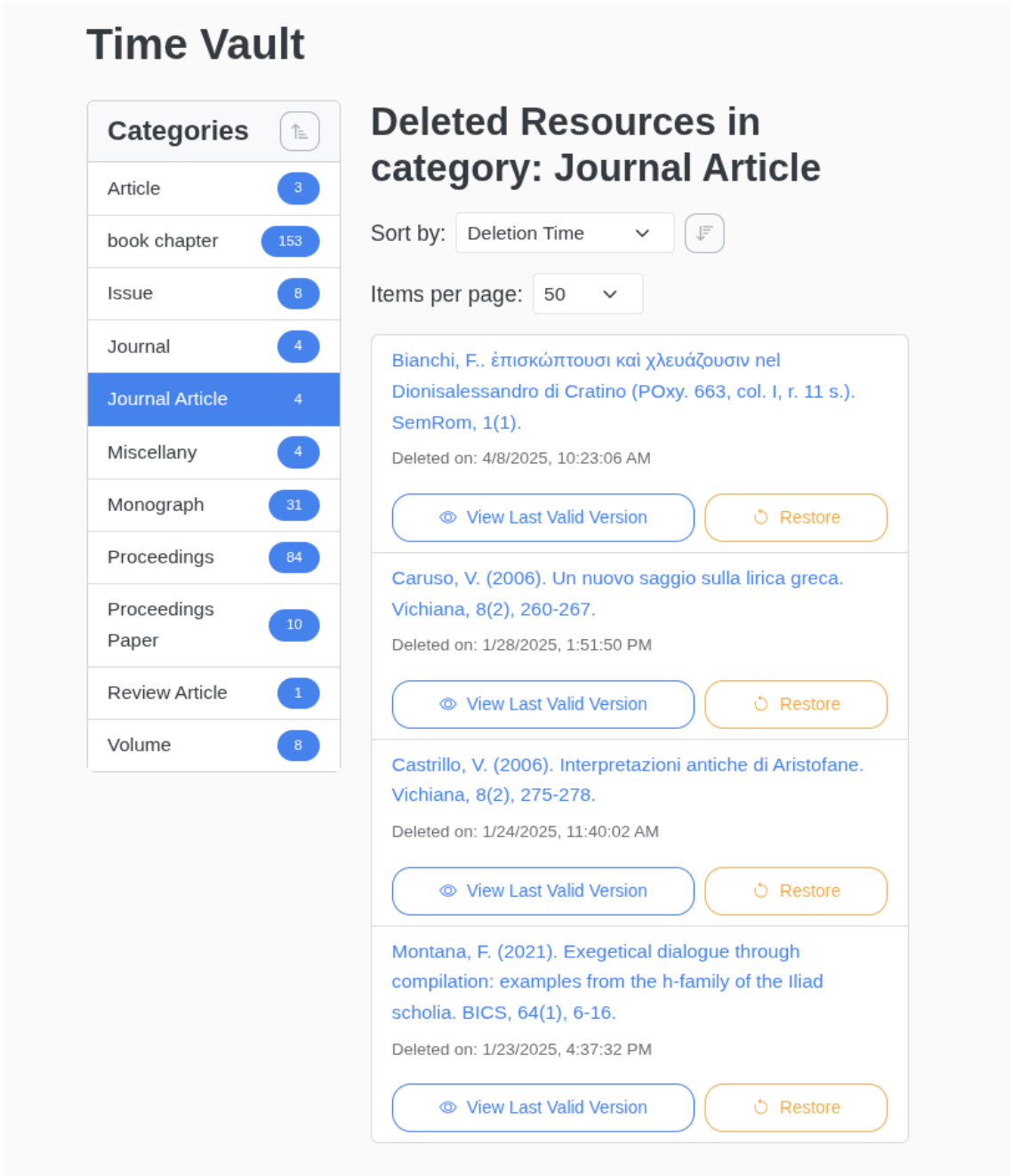


Figure 6.7: HERITRACE Time Vault interface showing deleted resources with restoration options and access to last valid versions.

6.4 Entity creation and validation systems

Entity creation in HERITRACE follows a structured process that guides users through metadata specification while implementing validation rules to maintain semantic consistency and prevent data integrity errors.

The entity creation interface begins with type selection, where users choose the appropriate category for the new resource from the available options in their dataset. The system dynamically adjusts the available metadata fields based on this selection, ensuring that only relevant properties appear for each entity type. For bibliographic collections, selecting "Journal Article" might present fields for identifier, title, description, abstract, keywords, authors, editors, collaborators, and publishers, while selecting "Journal" would display different property sets appropriate to publication venues. As illustrated in Figure 6.8, the interface also performs automatic searches for existing entities to support manual deduplication, preventing the creation of duplicate records.

The validation system operates at multiple levels to ensure data quality. Real-time validation checks field contents as users enter information, providing immediate feedback on format requirements and constraint violations. These validation rules derive from SHACL specifications that define the structure and constraints for each entity type, as detailed in Section 6.6.1. The interface dynamically adapts input controls based on SHACL datatype constraints and enforces cardinality rules by disabling "add" buttons when maximum values are reached and highlighting required fields when minimum counts are not satisfied.

Advanced validation mechanisms include SHACL `sh:pattern` properties that define regular expressions for format validation and `sh:message` properties that provide clear guidance when validation fails. For example, each identifier type may receive specific pattern validation: PMIDs must contain only numbers, PMCIDs must start with "PMC" followed by non-zero-leading numbers, and OpenAlex identifiers must begin with "W" followed by digits. This declarative validation approach catches format errors early in the creation process while providing informative error messages that guide users toward correct identifier formats.

The disambiguation system activates during entity creation to prevent unintentional duplication of existing resources. As users enter metadata such as titles, author names, or identifiers, the system searches for entities with similar attributes and presents potential matches for review.

The disambiguation process can be configured through YAML settings to control which properties trigger searches, what similarity thresholds apply, and how potential matches are presented to users. For example, when entering author information,

the system might search based on family name and given name combinations, present matches with existing ORCID identifiers, and allow users to either select existing entities or confirm the creation of new, distinct entries.

This approach to disambiguation addresses the common problem of entity multiplication in large datasets while respecting curatorial decision-making. Rather than automatically merging similar entries, which could introduce false positives, the system provides information that enables informed choices about whether apparent duplicates represent the same or different real-world entities.

The creation interface supports hierarchical entity creation, where nested entities within the data model can be created alongside the primary entity in a single workflow. As shown in Figure 6.8, when creating a journal article, users can simultaneously add related entities such as authors with their identifiers. The system recognizes that these are separate entities in the underlying data model but presents them as nested components during creation. This allows users to establish a primary bibliographic resource while concurrently creating and linking its associated responsible agents, each with their own identifiers and attributes, through a unified interface that maintains the proper relationships between all entities.

6.5 Entity merging and duplicate management

Cultural heritage collections frequently accumulate duplicate entities. In bibliographic datasets, for instance, authors lacking persistent identifiers often appear as multiple distinct entities. HERITRACE addresses entity duplication through configurable similarity detection mechanisms and structured merge workflows that consolidate duplicate records while preserving complete provenance of the consolidation operation.

When users create or view entities, the system automatically executes similarity searches based on configured criteria (Section 6.6.2). During entity creation, as users enter metadata values, the system queries the dataset for existing entities matching the configured similarity properties. When matches are discovered, the interface presents these potential duplicates alongside the creation form, enabling immediate recognition and preventing duplicate creation. During entity browsing, the system displays potential duplicates in a dedicated "Similar Resources" section of the entity detail view.

The merge workflow implements a three-step process that consolidates duplicate entities while maintaining data integrity and full reversibility. The process begins when users identify duplicate entities through similarity detection results. The in-

terface presents potential duplicates with merge options, allowing users to select which duplicate record should be consolidated into the current entity. This selection determines the directionality of the merge operation, establishing one entity as the primary record that will absorb properties from the duplicate.

Upon selecting entities for merging, the system presents a confirmation interface that displays both records side by side, as illustrated in Figure 6.9. This confirmation screen explicitly communicates the automatic merge operations that will occur: all properties from the duplicate entity will be transferred to the primary entity, all references throughout the dataset pointing to the duplicate entity will be updated to reference the primary entity instead, and the duplicate entity will be marked as deleted. This transparency enables users to verify merge decisions before execution and understand the scope of changes that will result from the consolidation.

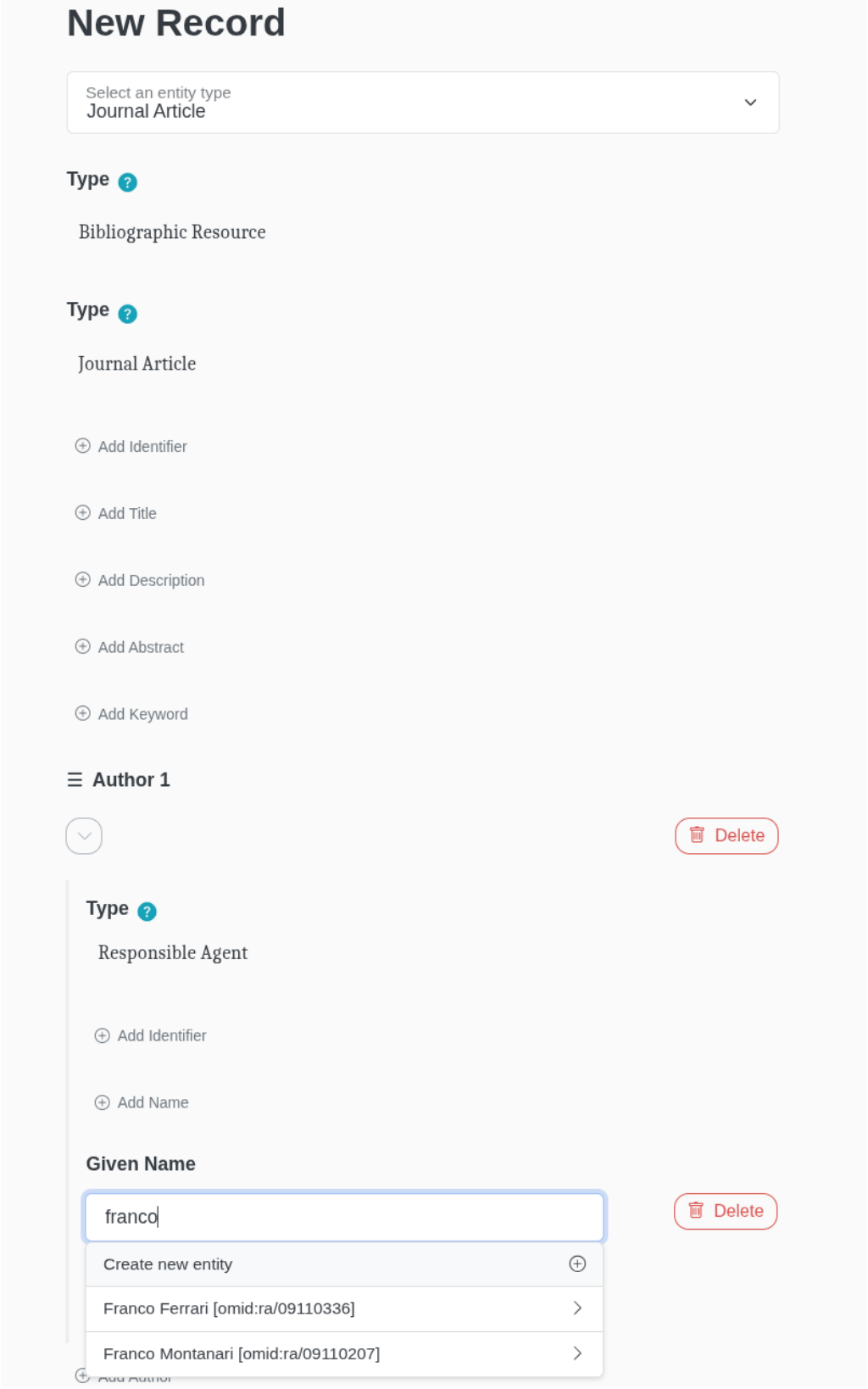


Figure 6.8: HERITRACE entity creation interface showing type selection, available metadata fields, automatic entity search for deduplication, and drag-and-drop handles for manual reordering of authors.

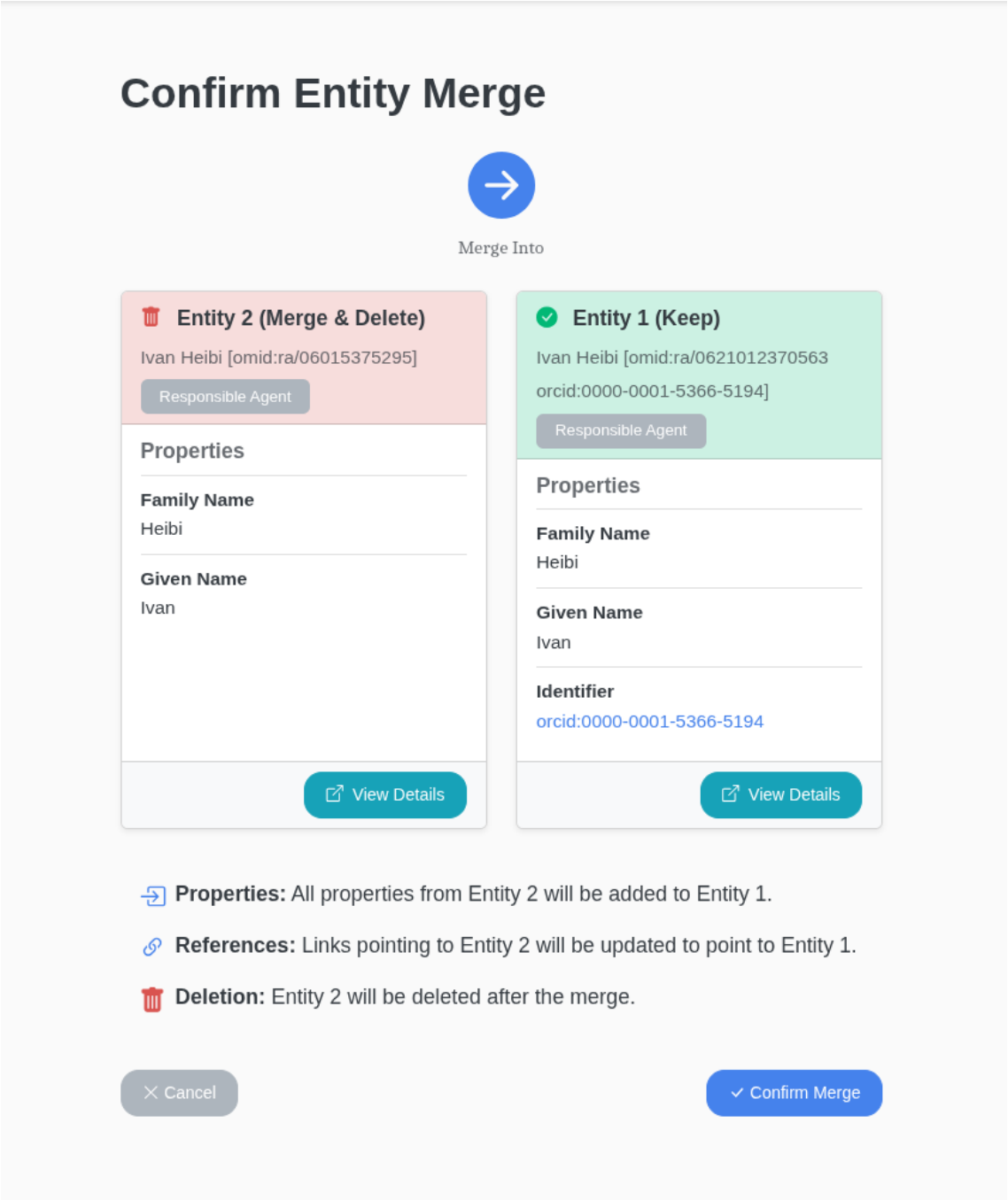


Figure 6.9: HERITRACE merge confirmation interface showing the primary entity and duplicate entity side by side, with explicit indication of automatic merge operations including property transfer, reference updates, and entity deletion.

The deletion of the duplicate entity follows HERITRACE's standard deletion semantics, where the entity is moved to the Time Vault rather than being permanently removed. This approach ensures that merge operations maintain complete reversibility through the time machine functionality described in Section 6.3. If users discover that a merge was performed incorrectly, perhaps because entities appearing similar actually represented distinct real-world resources, they can restore the deleted duplicate entity and reverse the property transfers, returning the dataset to its pre-merge state.

6.6 Configuration through standard technologies

HERITRACE addresses one of the primary barriers to adoption of semantic data management systems by enabling customization through widely-adopted standard technologies rather than proprietary configuration languages. This approach reduces the technical expertise required for system administration while ensuring that customizations remain maintainable and transferable.

The configuration architecture relies on the two technologies introduced in Section 6.1: SHACL for data model specification and constraint definition, and YAML as a declarative format for interface presentation and behavior control. Since both are widely adopted outside of HERITRACE, technical staff can apply existing knowledge from other systems and contexts without learning proprietary configuration languages.

6.6.1 Data model definition through SHACL

HERITRACE reads SHACL shapes at startup and translates them into interface elements, so that the same document that validates RDF data also drives form generation. An administrator writes shapes once; the system derives input controls, cardinality rules, and validation logic from those shapes without additional configuration.

The translation is direct. Datatype declarations determine which widget appears: a shape that lists `xsd:date`, `xsd:gYearMonth`, and `xsd:gYear` as acceptable types for `prism:publicationDate` produces a format selector followed by the matching input control (calendar, month-year picker, or plain numeric field). Cardinality constraints (`sh:minCount`, `sh:maxCount`) determine whether a field is single-valued, repeatable, or required: `dcterms:title` with a maximum of one becomes a single text area, while `prism:keyword` with no upper bound becomes an expandable set. The `sh:in` constraint restricts a property to a fixed list of values and renders as

a dropdown menu; for instance, restricting `datacite:usesIdentifierScheme` to ORCID, VIAF, and Wikidata replaces free-text entry with a selection control.

More expressive SHACL constructs map to correspondingly richer interface behaviour. The `sh:or` construct, which declares that a property may conform to one of several shapes, translates into a type selector: the `frbr:partOf` property of a journal article accepts `fabio:JournalIssue`, `fabio:JournalVolume`, or `fabio:Journal`, and the interface presents these as options so that users can only create valid containment relationships. Node shapes with multiple required properties (*e.g.*, a `fabio:JournalVolume` that must carry both `fabio:hasSequenceIdentifier` and a `frbr:partOf` link to a journal) produce forms where all required fields are present from the start, preventing incomplete hierarchical records.

Conditional constraints add context sensitivity. The `sh:condition` construct applies different validation rules depending on a property's value. For example, when the identifier scheme property equals `datacite:doi`, the system enforces a DOI-specific regular expression (`^10\.\d{4,}/\S+$`), while other schemes trigger their own patterns. This avoids duplicating entity definitions for each identifier type. Shape references via `sh:node` extend the same principle across entities: the property `datacite:hasIdentifier` points to a dedicated identifier shape whose constraints apply recursively, keeping each shape self-contained. Fixed-value constraints (`sh:hasValue`) handle the opposite case, locking a property to a single allowed value. In `AuthorShape`, for instance, `pro:withRole` must equal `pro:author`, a constraint the interface enforces silently rather than exposing to users.

Listing 22 shows the shape that produces the identifier section of Figure 6.4. The `sh:in` list generates the scheme dropdown, `sh:minCount` marks both fields as required, and `sh:maxCount` restricts each to a single value.

6.6.2 Interface presentation through YAML

YAML configuration files govern the presentation and interaction aspects that fall outside SHACL's scope: how entities appear in catalogue listings, which properties are visible in editing forms, how search works, and how values are displayed. Technical staff edit these files directly, without modifying application code. Listing 23 shows the configuration for Journal Article entities in ParaText; the paragraphs that follow walk through its main sections.

Each configuration block begins with a `target` that binds it to an RDF class, a SHACL shape, or both. When both are present the shape takes precedence, which allows distinct interface treatments for entities that share the same ontological class. Regular and special journal issues, for instance, are both typed as

```
schema:JournalArticleIdentifierShape
  a sh:NodeShape ;
  sh:targetClass datacite:Identifier ;
  sh:property [
    sh:path datacite:usesIdentifierScheme ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:in ( datacite:doi datacite:pmid datacite:pmcid datacite:openalex )
    ;
  ] ;
  sh:property [
    sh:path literal:hasLiteralValue ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
  ] .
```

Listing 22: SHACL shape for journal article identifiers. The `sh:in` constraint generates the dropdown menu visible in Figure 6.4, while `sh:minCount` produces the required field indicators.

`fabio:JournalIssue`, but a special issue carries additional properties (title, abstract, thematic description) that warrant a separate editing form. A `priority` attribute resolves ambiguity when multiple configurations could match the same entity, and `displayName` replaces raw URIs with human-readable labels throughout the interface.

Entity URIs also need readable labels. The `fetchUriDisplay` field contains a SPARQL query that assembles a display string from the entity's own data. In Listing 23 the query concatenates author surnames and title, so that catalogue listings show "Peroni & Shotton. OpenCitations, an infrastructure organization for open scholarship" rather than `https://w3id.org/oc/meta/br/062501777134`. A similar mechanism, `fetchValueFromQuery`, formats linked entities that appear as property values: an identifier entity, for example, can be rendered as `doi:10.1234/567` instead of its URI.

The `displayProperties` array determines which fields appear in the editing form, in what order, and with what controls. Each entry can override the default input widget (`textarea` for titles, `date` for dates, `tag` for keywords) and activate search-as-you-type suggestions through `supportsSearch`. The `searchTarget` attribute distinguishes two lookup strategies: `self` returns entities of the same type whose property matches the typed text, while `parent` returns the main entity that owns a matching sub-entity. Searching for an identifier value, for example, should return the bibliographic resource, not the identifier entity itself. The `sortableBy` attribute at the end of the block declares which properties can be used for ordering

```yaml
- target:
    class: "http://purl.org/spar/fabio/JournalArticle"
    shape: "http://schema.org/JournalArticleShape"
  priority: 1
  shouldBeDisplayed: true
  displayName: "Journal Article"
  fetchUriDisplay: |
    PREFIX dcterms: <http://purl.org/dc/terms/>
    PREFIX fabio: <http://purl.org/spar/fabio/>
    PREFIX foaf: <http://xmlns.com/foaf/0.1/>
    PREFIX pro: <http://purl.org/spar/pro/>
    SELECT ?display
    WHERE {
      [[uri]] dcterms:title ?title .
      OPTIONAL {
        SELECT (GROUP_CONCAT(?authorName; SEPARATOR = " & ") AS
        ?authorList)
        WHERE {
          [[uri]] pro:isDocumentContextFor ?authorRole .
          ?authorRole pro:withRole pro:author ;
                      pro:isHeldBy ?author .
          ?author foaf:familyName ?authorName .
        }
      }
      BIND(CONCAT(
        COALESCE(?authorList, ""),
        IF(BOUND(?authorList), ". ", ""),
        ?title
      ) AS ?display)
    }
  displayProperties:
    - property: "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
      displayName: "Type"
      shouldBeDisplayed: true
      supportsSearch: false
    - property: "http://purl.org/dc/terms/title"
      displayName: "Title"
      shouldBeDisplayed: true
      inputType: "textarea"
      supportsSearch: true
      minCharsForSearch: 2
      searchTarget: "self"
    - property: "http://purl.org/spar/datacite/hasIdentifier"
      displayName: "Identifier"
      shouldBeDisplayed: true
      supportsSearch: true
      searchTarget: "parent"
  sortableBy:
    - property: "http://purl.org/dc/terms/title"
      sortOrder: ["asc", "desc"]
    - property:
    "http://prismstandard.org/namespaces/basic/2.0/publicationDate"
      sortOrder: ["desc", "asc"]
```

Listing 23: YAML configuration for a Journal Article entity in ParaText.

in catalogue and time vault views, with the first direction in the array serving as the default.

Where the order of multi-valued properties matters (author sequences, for instance), the `orderedBy` attribute activates drag-and-drop reordering in the interface, as visible in Figure 6.8. The system stores the ordering as RDF linked lists, keeping sequence information independent of triple insertion order in the store.

Some ontologies model relationships that carry their own metadata as intermediate entities rather than direct predicates. Author roles in the bibliographic domain are a typical case: a `pro:RoleInTime` entity connects a person to a document while recording the role type (author, editor) and, optionally, temporal bounds. For the user, however, adding an author feels like a direct action, not the construction of an intermediate entity. YAML `intermediateRelation` blocks handle this mismatch by specifying the class of the intermediate entity and the target entity type. Separate `displayRules` within the same block can differentiate shapes, so that authors (`AuthorShape`) and editors (`EditorShape`) present different fields despite sharing the same `pro:RoleInTime` class.

Duplicate detection, introduced in Section 6.5, is also configured through YAML. The `similarity_properties` attribute accepts property URIs combined with disjunctive and conjunctive logic. Listing 24 illustrates three strategies for a bibliographic system: journal articles match on title *or* identifier (disjunctive), journal volumes match only when both sequence identifier *and* container match simultaneously (conjunctive), and agents combine both patterns to cover name, surname-plus-given-name, and identifier matches.

6.6.3 Configuration interaction matrix

Since both SHACL and YAML are independently optional, their presence or absence produces four operational scenarios, summarised in Table 6.1. The governing rule is that SHACL always takes precedence: when shapes exist for an entity class, only the properties they declare are available for editing, and YAML display rules can style those properties but cannot add new ones or relax validation. If `JournalArticleShape` defines only `dcterms:title` and `dcterms:abstract`, a YAML rule for `prism:keyword` has no effect until that property is added to the shape. When no SHACL shapes are present, the interface falls back to free-form property entry where users type URIs and values directly. Display names follow the same priority: YAML `displayName` attributes override automatic URI formatting, which converts identifiers like `hasPublicationDate` into “has publication date” as a fallback.

```yaml
# Disjunctive logic: match if ANY property matches
- target:
    class: "http://purl.org/spar/fabio/JournalArticle"
  similarity_properties:
    - "http://purl.org/dc/terms/title"
    - "http://purl.org/spar/datacite/hasIdentifier"

# Conjunctive logic: match if ALL properties match
- target:
    class: "http://purl.org/spar/fabio/JournalVolume"
  similarity_properties:
    - and:
      - "http://purl.org/spar/fabio/hasSequenceIdentifier"
      - "http://purl.org/vocab/frbr/core#partOf"

# Combined logic: match through multiple strategies
- target:
    class: "http://xmlns.com/foaf/0.1/Agent"
  similarity_properties:
    - "http://xmlns.com/foaf/0.1/name"
    - and:
      - "http://xmlns.com/foaf/0.1/familyName"
      - "http://xmlns.com/foaf/0.1/givenName"
    - "http://purl.org/spar/datacite/hasIdentifier"
```

Listing 24: YAML similarity properties configuration demonstrating disjunctive, conjunctive, and combined matching strategies for different entity types.

This optionality matters for deployment. HERITRACE can connect to an existing triple store, discover entities through their RDF types, and present them with default formatting without any prior configuration. SHACL and YAML refine the experience incrementally: an institution can start with no configuration and add shapes and display rules as needs become clearer.

6.6.4 Virtual properties

The `intermediateRelation` mechanism described above handles cases where users still see the intermediate entity's fields. Virtual properties go one step further: they present what looks like a direct relationship field, but when the user populates it the system silently creates and configures one or more intermediate entities behind the scenes. The distinction matters because some ontological patterns are too distant from the user's mental model to expose at all. The Citation Typing Ontology (CITO) (Peroni & Shotton, 2012), for instance, models citations as `cito:Citation` entities that link a citing resource to a cited resource while carrying metadata such as citation type. A domain expert, however, thinks "I want to cite this paper," not "I want to create a Citation entity linking my paper to that paper." A virtual property labelled "Citations" lets the user search for and select a target resource;

Table 6.1: Configuration interaction matrix showing how different combinations of SHACL and YAML configurations determine system behavior.

Scenario	**SHACL Schema**	**Display Rules**	**Entity Creation Method**	**Property Constraints**	**Form Generation**	**Display Names**
Full Configuration	Present	Present	SHACL-based forms (predefined entity types)	SHACL constraints enforced	Generated from SHACL + Display Rules styling	From Display Rules with URI fallback
SHACL Only	Present	Absent	SHACL-based forms (predefined entity types)	SHACL constraints enforced	Generated from SHACL only	Auto-generated from URI
Display Rules Only	Absent	Present	Custom properties form (free-form)	No validation	Custom properties form	From Display Rules (if class matches) or URI
Neither Present	Absent	Absent	Custom properties form (free-form)	No validation	Custom properties form	Auto-generated from URI

the system creates the `cito:Citation` entity, sets its type, links it to the current resource via `cito:hasCitingEntity`, and exposes only the two fields the user needs to fill: the cited resource and, optionally, the citation characterisation.

Listing 25 shows the YAML configuration that produces this behaviour. The `implementedVia` block specifies the class and shape of the intermediate entity, and `fieldOverrides` controls which of its properties are hidden (pre-populated automatically) and which are shown to the user. The placeholder `${currentEntity}` resolves to the URI of the resource being edited, so the linking predicate is filled without user intervention. The corresponding SHACL shape (Listing 26) constrains citation entities to exactly one citing and one cited resource, and restricts the citation characterisation to a controlled CITO vocabulary via `sh:in`.

6.7 System configuration

Runtime behaviour, external service integration, and deployment parameters are controlled through environment variables, keeping application code separate from deployment-specific settings. The configuration covers four areas: database connec-

```yaml
displayProperties:
  - displayName: "Citations"
    isVirtual: true
    shouldBeDisplayed: true
    fetchValueFromQuery: |
      PREFIX cito: <http://purl.org/spar/cito/>
      PREFIX dcterms: <http://purl.org/dc/terms/>
      SELECT ?display ?target WHERE {
        [[value]] a cito:Citation ;
                  cito:hasCitingEntity [[subject]] ;
                  cito:hasCitedEntity ?target .
        ?target dcterms:title ?title .
        BIND(CONCAT(?title, " (", STR(?target), ")") AS ?display)
      }
    implementedVia:
      target:
        class: "http://purl.org/spar/cito/Citation"
        shape: "http://schema.org/CitationShape"
      fieldOverrides:
        "http://www.w3.org/1999/02/22-rdf-syntax-ns#type":
          shouldBeDisplayed: false
          value: "http://purl.org/spar/cito/Citation"
        "http://purl.org/spar/cito/hasCitingEntity":
          shouldBeDisplayed: false
          value: "${currentEntity}"
        "http://purl.org/spar/cito/hasCitedEntity":
          displayName: "Cited Resource"
        "http://purl.org/spar/cito/hasCitationCharacterization":
          displayName: "Citation Type"
```

Listing 25: YAML configuration for a virtual property representing bibliographic citations, showing automatic population of intermediate entity fields and custom display names for user-facing properties.

tivity, authentication, provenance baseline, and entity lifecycle.

Database configuration is triplestore-agnostic. Administrators specify SPARQL endpoints for the dataset and provenance stores separately, and declare which triplestore product is in use (currently tested with Virtuoso and Blazegraph) so that the query layer can apply product-specific optimisations such as full-text indexing. A separate flag for each store indicates whether named graphs are supported, allowing the system to adapt its query patterns to triple-only or quad-capable backends.

Authentication relies on ORCID OAuth 2.0 credentials, as described in Section 6.1. An allowlist of ORCID identifiers restricts editing access to designated individuals, so that the same ORCID infrastructure used for scholarly identity also serves as the access control mechanism.

When HERITRACE is deployed on a pre-existing RDF collection, the provenance subsystem needs two baseline values: a default primary source (typically a DOI or URL identifying the dataset's origin) and the creation timestamp of the

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix cito: <http://purl.org/spar/cito/> .
@prefix schema: <http://schema.org/> .

schema:CitationShape a sh:NodeShape ;
    sh:targetClass cito:Citation ;
    sh:property [
        sh:path rdf:type ;
        sh:hasValue cito:Citation ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path cito:hasCitingEntity ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path cito:hasCitedEntity ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path cito:hasCitationCharacterization ;
        sh:in (cito:cites cito:citesAsAuthority cito:citesAsEvidence
               cito:citesAsMetadataDocument cito:citesAsRelated
               cito:citesAsSourceDocument cito:citesForInformation) ;
        sh:maxCount 1 ;
    ] .
```

Listing 26: SHACL shape for Citation entities created through virtual properties. Cardinality constraints enforce exactly one citing and one cited resource; `sh:in` restricts characterisation to a controlled CITO vocabulary.

imported data. Without an explicit timestamp, the system defaults to the current time, which would misrepresent the provenance of historical data. These values populate the `prov:hadPrimarySource` and `prov:generatedAtTime` properties of the initial snapshot, establishing an accurate starting point for subsequent change tracking. URI generation is pluggable: institutions can replace the default UUID-based strategy with counter-based schemes or external identifier services by providing an alternative Python class, without modifying application code.

Deletion of an entity can leave behind disconnected resources. An orphan is a resource that loses all incoming references: removing the last article from a journal issue, for instance, may leave its identifier entities unreferenced. An intermediate entity (Section 6.6.2) loses its purpose when the relationship it mediates is deleted. For both cases the administrator chooses one of three policies: prompt the user for confirmation before removing the disconnected resource, remove it automatically, or

keep it in the dataset for potential future reuse.

Chapter 7

Evaluation

This chapter presents an empirical evaluation of HERITRACE through a mixed-methods study involving both domain experts and technical staff. The evaluation assesses whether the system addresses the research questions stated in Chapter 1: whether domain experts can participate in semantic data curation through user-friendly interfaces (RQ1), and whether technical staff can successfully perform the one-time configuration that adapts the system to specialized domains (RQ2).

Two distinct user populations participated, corresponding to the two research questions. Domain experts (referred to as end users) represent scholars and cultural heritage professionals who interact with semantic data daily. Technical staff (referred to as technicians) represent information technology professionals responsible for initial system setup. This separation reflects operational reality: technicians configure the system once during deployment, after which domain experts curate data independently.

The evaluation employs a mixed-methods design where quantitative and qualitative data are collected simultaneously but analyzed independently, with integration occurring during interpretation. The quantitative component measures task performance through completion rates, duration metrics, error frequencies, and standardized usability assessment via the System Usability Scale (SUS) (Brooke, 1996). The qualitative component employs grounded theory methodology (Corbin & Strauss, 2008) to identify adoption barriers and characterize interaction patterns during actual system use. The evaluation protocol implements self-guided testing with screen and voice recording: participants received documentation and testing materials in advance, completed tasks independently without researcher intervention, and provided both SUS assessments and qualitative feedback through written reflections and think-aloud protocols. This approach balances experimental control with ecological validity (Kieffer, Sangiorgi, & Vanderdonckt, 2015), generating data about

how participants engage with the system when working autonomously in their own environments rather than under direct observation in a laboratory setting.

All evaluation materials, including Docker-based testing packages for both user populations, anonymized transcripts, coding data at all levels (open, axial, selective), written responses, SUS scores, and aggregate analyses, have been published on Zenodo (Massari, 2025b) to support transparency and reproducibility.

7.1 Methodology

7.1.1 Participants and recruitment

Participant recruitment targeted two distinct populations corresponding to HERITRACE's dual-audience design. End user recruitment sought academic professionals with knowledge of bibliographic resource structure and metadata, basic web application experience, and familiarity with cultural heritage or scholarly communication workflows. Technician recruitment targeted technical staff with working knowledge of Semantic Web technologies. The final sample consisted of 9 end users and 10 technicians. Figure 7.1 presents the distribution of prior familiarity with HERITRACE and SHACL across both participant groups.

The end user sample included 8 participants (88.9%) with no prior HERITRACE experience and 1 participant (11.1%) with usage knowledge from previous exposure. This distribution reflects realistic deployment scenarios where most users encounter the system for the first time during institutional adoption. The technician sample included 9 participants (90%) with no prior HERITRACE experience and 1 participant (10%) with usage knowledge. Regarding SHACL expertise, 4 technicians (40%) had no prior experience with SHACL, while 6 technicians (60%) possessed working knowledge. This heterogeneity enables evaluation of whether the system supports users across experience levels or whether it requires specific prerequisite knowledge for adoption.

Participants received pre-testing materials including HERITRACE documentation (https://opencitations.github.io/heritrace/), equipment setup instructions for screen recording software and system requirements (1GB RAM, 5GB free disk space, modern web browser), and testing protocols describing session structure and task sequence.

Participants completed testing sessions independently in their own environments using personal computers. Testing employed Docker-based packages containing preconfigured HERITRACE instances, datasets, and task protocols, enabling consistent deployment across heterogeneous computing environments while minimizing tech-

Figure 7.1: Participant recruitment distribution showing HERITRACE familiarity for end users and technicians, and SHACL familiarity for technicians.

nical setup barriers. This naturalistic setting maximizes ecological validity (Kieffer et al., 2015) by avoiding laboratory effects from unfamiliar equipment or direct observation by researchers during testing sessions, reducing observation pressure on participants. However, it introduces variability in hardware performance and environmental distractions that could influence task completion times and error rates. The evaluation acknowledges these trade-offs between experimental control and real-world applicability.

Participants signed a privacy consent form informing that personal data including voice recordings would be used for research purposes. The privacy form received approval from the University of Bologna privacy office before testing commenced.

7.1.2 Testing protocol and tasks

The evaluation implemented self-guided testing sessions lasting a maximum of 60 minutes for both user populations. Sessions followed a structured sequence beginning with a brief warm-up exploration period, proceeding through task execution,

and concluding with questionnaire completion and written reflection. Participants received written instructions for each phase and were instructed to think aloud throughout the session, verbalizing their expectations, observations, questions, and reactions to support qualitative analysis of interaction patterns.

Both user populations employed the same dataset: a subset of OpenCitations Meta (Massari et al., 2024) containing a hundred journal articles. The difference between end user and technician testing environments lay in configuration completeness: end users worked with fully configured SHACL schemas and display rules enabling immediate data curation activities, while technicians received partially configured instances where some entities lacked configuration elements that they had to implement during testing tasks.

End users completed four metadata management tasks of increasing complexity. Each task requires the system to manage substantial RDF machinery behind the scenes; the evaluation measures whether users can accomplish their goals without being exposed to that machinery.

Task 1 consisted in editing an existing publication record. Participants received a direct URL to a journal article and were asked to add an author and reorder the author sequence. The task exercises entity disambiguation through autocomplete (detecting that the author already exists in the dataset) and drag-and-drop reordering. Behind the interface, authorship is modeled through SPAR Ontologies (Peroni, Shotton, & Vitali, 2012) proxy entities (`pro:RoleInTime`) linked by `oco:hasNext` chains, as described in Section 6.6.2; users interact only with drag handles and name fields.

Task 2 consisted in merging duplicate author entities. Participants identified duplicate author records and consolidated them through the merge interface (Section 6.5). The underlying operation rewires all triples referencing the duplicate, transfers its properties to the primary entity, and deletes the duplicate while preserving full reversibility through provenance tracking.

Task 3 consisted in restoring a previous version. Participants used the Time Machine to revert the journal article modified in Task 1 to its pre-edit state. Restoration requires reconstructing the original proxy entity network and `oco:hasNext` chain, an operation the user triggers with a single button.

Task 4 consisted in creating a new publication record from scratch. Participants received complete metadata for a journal article (title, date, ordered authors with ORCID identifiers, publisher, issue, volume, journal) and had to enter it through the creation form. This is the most structurally demanding task: it involves proxy entities for author ordering, hierarchical identifier creation (ORCID identifiers modeled

as separate entities linked through intermediaries), date datatype selection among `xsd:date`, `xsd:gYearMonth`, and `xsd:gYear`, and nested containment hierarchies (article within issue within volume within journal).

The testing session was allocated a maximum of 60 minutes. Time was distributed across activities based on presumed complexity: 2 minutes for warm-up exploration, 8 minutes for editing existing publications, 10 minutes for merge operations, 7 minutes for version restoration, 20 minutes for creating new publications, 3 minutes for SUS questionnaire completion, and 10 minutes for written reflection. When participants exceeded the allocated time for a specific task, this time limit violation was recorded even if the participant ultimately completed the task successfully.

Technician testing employed a partially configured HERITRACE instance where some entities possessed complete SHACL schemas and display rules while others lacked configuration elements.

Technicians completed two configuration tasks designed to evaluate SHACL and YAML configuration workflows. Task 1 consisted in adding SHACL validation for abstract. Participants were required to modify SHACL shapes to add the `dcterms:abstract` property to journal article entities with maximum cardinality of one. This task assessed whether SHACL provides an intuitive mechanism for form generation connected to semantic models, both for users with prior SHACL expertise and for users encountering SHACL for the first time but possessing Semantic Web knowledge. Task 2 consisted in improving abstract display. Participants were required to configure display rules for the `dcterms:abstract` property added in Task 1 with specific requirements: display name "Abstract", visibility enabled, and textarea input type. This task assessed YAML configuration comprehension and the ability to translate natural language requirements into declarative configuration syntax.

The testing session was allocated a maximum of 60 minutes. Time was distributed across activities based on presumed complexity: 2 minutes for warm-up exploration, 22 minutes for implementing SHACL validation, 23 minutes for improving display configuration, 3 minutes for SUS questionnaire completion, and 10 minutes for written reflection. As with end users, time limit violations were recorded regardless of eventual task completion.

Task presentation avoided prescriptive step-by-step instructions, instead providing goal descriptions and requirements while allowing participants to determine their own strategies.

7.1.3 Data collection instruments

Data collection employed three instruments: screen and voice recordings capturing interaction processes, SUS questionnaires assessing perceived usability, and written reflection questions eliciting qualitative feedback about user experiences.

VoxTral (Liu et al., 2025), an automatic speech recognition model, processed voice recordings locally to generate initial transcripts, ensuring participant privacy by avoiding external cloud services. The author of this thesis, acting as the sole analyst, manually corrected these transcripts while reviewing videos to ensure accuracy and capture contextual information that automated transcription might miss.

The SUS provides standardized usability assessment through a 10-item questionnaire using 5-point Likert scales ranging from strongly disagree to strongly agree. SUS score calculation follows the standard formula where odd items receive scores of (rating - 1), even items receive scores of (5 - rating), and the sum multiplied by 2.5 produces a 0-100 scale score. The resulting scores enable comparison against established benchmarks (Sauro, 2016): below 68 indicates below-average usability, 68 to 80.3 indicates above-average usability, and above 80.3 indicates excellent usability. While subscale analysis distinguishing Usability (items 1, 2, 3, 5, 6, 7, 8, 9) from Learnability (items 4 and 10) was initially proposed (J. R. Lewis & Sauro, 2009), subsequent research consistently failed to replicate the proposed factor structure and recommended treating the SUS as a unidimensional measure of perceived usability rather than computing separate subscales (J. Lewis & Sauro, 2017). This evaluation therefore reports only overall SUS scores.

Written reflection questions complemented the SUS quantitative assessment with open-ended qualitative inquiry. Four questions probed different experiential dimensions: effectiveness ("How effectively did HERITRACE support you in these tasks?"), feature value ("What were the most useful features for your work?"), weaknesses ("What were the main weaknesses or frustrating aspects you encountered?"), and enhancement opportunities ("What additional features would have made this more useful for your work?"). This structure balances positive and negative inquiry, avoiding leading questions that might bias responses toward either favorable or critical assessments. Participants provided written responses without length constraints.

7.1.4 Analysis framework

The analysis framework implements separate procedures for quantitative and qualitative data, followed by integration during interpretation. The author of this thesis

conducted all analysis as the sole analyst. Quantitative analysis computes descriptive statistics for success rates, task durations, error frequencies, and SUS scores, while qualitative analysis implements grounded theory methodology through systematic coding procedures.

Task completion tracking constitutes the first analysis phase. For each task, the author documented actual duration measured from task initiation to completion or timeout, completion status classified as complete (working deliverable produced), partial (incomplete attempt with some progress), success timeout (time limit expired while actively working), failed misunderstanding (task or feature misunderstood preventing completion), failed bug (system defect prevented completion), or failed blocked (participant understood the task but encountered insurmountable obstacles preventing progress and abandoned the attempt). Error counting followed a protocol where the author identified each observable event preventing expected progress, forcing retry or workaround, or indicating user confusion tied to specific interface or configuration causes. Each error received severity classification (low: minor issue with little impact; medium: requires workaround and causes noticeable delay; high: blocks task completion or requires restart) and symptom, trigger, and outcome documentation enabling root cause analysis. This manual coding process generated structured performance data from unstructured video recordings, enabling subsequent quantitative analysis.

Quantitative analysis begins with task completion metrics calculation using the data extracted through task completion tracking. Task completion distributions report the proportion of each completion status (complete, partial, success timeout, failed misunderstanding, failed bug, failed blocked) separately for each task and user type. Error analysis computes both frequency counts and severity-weighted scores where low-severity errors receive weight 1, medium-severity errors receive weight 2, and high-severity errors receive weight 4. Duration analysis compares expected and actual task times, identifying tasks where users consistently exceed or complete ahead of expected durations.

SUS analysis computes descriptive statistics (mean, standard deviation, median, range) for overall scores, separately for each user type.

Qualitative analysis implements grounded theory methodology through three sequential coding phases: open coding, axial coding, and selective coding. Open coding begins with line-by-line analysis of transcripts and written responses, identifying concepts and assigning codes using participants' own words where possible (in vivo coding). Each open code receives sentiment classification as positive, negative, or neutral.

Axial coding groups related open codes into higher-level categories representing broader phenomena. The author applied constant comparison method where each axial category is systematically compared with existing axial categories, asking "What other categories are similar or different?", "Should this be a new category or merged with an existing one?", and "How do these categories relate?" Each axial category receives a descriptive name, documentation of related open codes, description explaining how codes relate, task contexts where the category appears, and frequency indicating the total number of open codes subsumed. Axial categories group only open codes sharing the same sentiment classification, ensuring thematic coherence. Verification software checks that all open codes have been assigned to axial categories, maintaining 100% coverage.

Selective coding identifies the core category with greatest explanatory power and frequency, develops a theory statement explaining the central phenomenon emerging from the data, and groups axial categories into supporting categories that explain different aspects of the core theory. Verification software checks that all axial categories have been integrated into the final theoretical framework, maintaining 100% coverage and ensuring that the theory accounts for the complete dataset rather than highlighting only supportive findings. This phase produces a substantive theory grounded in empirical data.

7.2 Quantitative results

7.2.1 Task completion analysis

Task completion analysis reveals high success rates across both user populations with notable variation in task difficulty, completion time, and error patterns. Figure 7.2 presents task completion distributions for all tasks, showing the proportion of complete, partial, timeout, and failure outcomes.

End users achieved success rates of 78% for editing an existing publication, 100% for merging duplicate authors, 78% for restoring a previous version, and 67% for creating a new publication. The merge authors task demonstrated the highest success rate with all 9 participants completing the task. The editing and version restoration tasks both achieved 78% success rates, though through different failure modes. Two participants exceeded the time limit while actively working on the edit task (success timeout status), while the version restoration task experienced two distinct failure modes. One participant demonstrated a conceptual misunderstanding of the restore functionality, attempting to manually undo changes by editing and removing the previously added author rather than using the Time Machine feature. Another

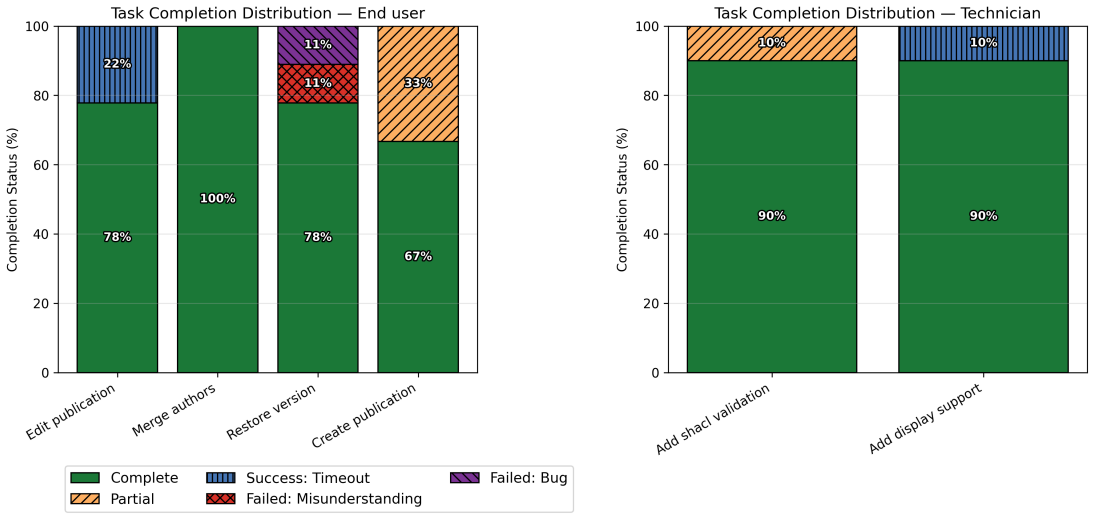


Figure 7.2: Task completion status distribution by user type and task, showing success rates and failure modes across end user metadata management tasks and technician configuration tasks. Categories are distinguished by fill pattern: complete (solid), partial (left-to-right diagonal), timeout (vertical lines), misunderstanding (crosshatch), bug (right-to-left diagonal).

participant encountered a system bug triggered by an interaction pattern that the Time Machine snapshot generation mechanism did not anticipate. The task required adding a new author as the first author in the sequence, which would push the existing first author to the second position. However, this participant modified the existing first author's data directly, overwriting it with the new author's information instead of adding a new author entity. This direct modification pattern prevented the Time Machine from generating a snapshot, causing the restore button to become unavailable when the participant later attempted version restoration.

The create publication task achieved 67% success with 6 complete outcomes and 3 partial outcomes. The partial completions stemmed from three distinct issues that prevented full metadata entry. One participant encountered entity creation visibility issues where created entities appeared in a collapsed state within the interface, making it unclear whether entities had been successfully created and preventing verification of the metadata entry process. Another participant failed to recognize that journal identifiers follow the same pattern as author identifiers, attempting to locate a dedicated ISSN field rather than adding the ISSN through the journal's identifier mechanism in the same way ORCID identifiers are added to authors. A third participant experienced context loss while navigating the deeply nested entity creation interface, accidentally entering the journal ISSN into the title field instead

of the identifier field due to loss of awareness about which entity and field type was being edited within the multi-level dropdown structure. These cases represent a combination of interface visibility issues, pattern recognition challenges in applying identifier entry mechanisms across different entity types, and navigation difficulties in deeply nested forms. No participants experienced complete failure on this task, indicating that the creation workflow remains fundamentally usable despite these specific friction points.

Technicians achieved success rates of 90% for both adding SHACL validation and adding display support. The SHACL validation task yielded 9 complete outcomes and 1 partial outcome. The partial completion occurred when a participant misread the task requirements and configured the abstract property as required rather than optional, setting minimum cardinality to 1 instead of 0. The display support task produced 9 complete outcomes and 1 success timeout. The timeout case occurred when a participant placed the abstract display configuration in the wrong YAML section, configuring it under the journal entity type instead of the journal article entity type. This structural error caused the system to ignore the configuration entirely, requiring extended troubleshooting to identify the misplacement and correct the entity type assignment.

Across both user populations and all tasks, no participants abandoned tasks due to insurmountable obstacles (failed blocked status). While participants experienced various difficulties including timeouts, bugs, and misunderstandings, all maintained engagement either until task completion, partial completion, timeout expiration, or until technical failures prevented further progress.

7.2.2 Task duration analysis

Task completion times in usability studies typically exhibit positive skewness, where most participants complete tasks quickly while a minority require substantially longer durations due to difficulties, confusion, or technical issues. This asymmetric distribution pattern makes arithmetic means misleading, as a single extremely slow completion can inflate the average far beyond typical user experience. The box plot visualization addresses this challenge by presenting multiple statistical measures that collectively reveal both central tendency and variability while remaining resistant to outliers (Sauro & Lewis, 2010). Figure 7.3 and Figure 7.4 present box plot distributions showing actual completion times for end user and technician tasks respectively, including median values, interquartile ranges, and 95% confidence intervals for the means.

The box plot visualizations follow the standard format introduced by Tukey

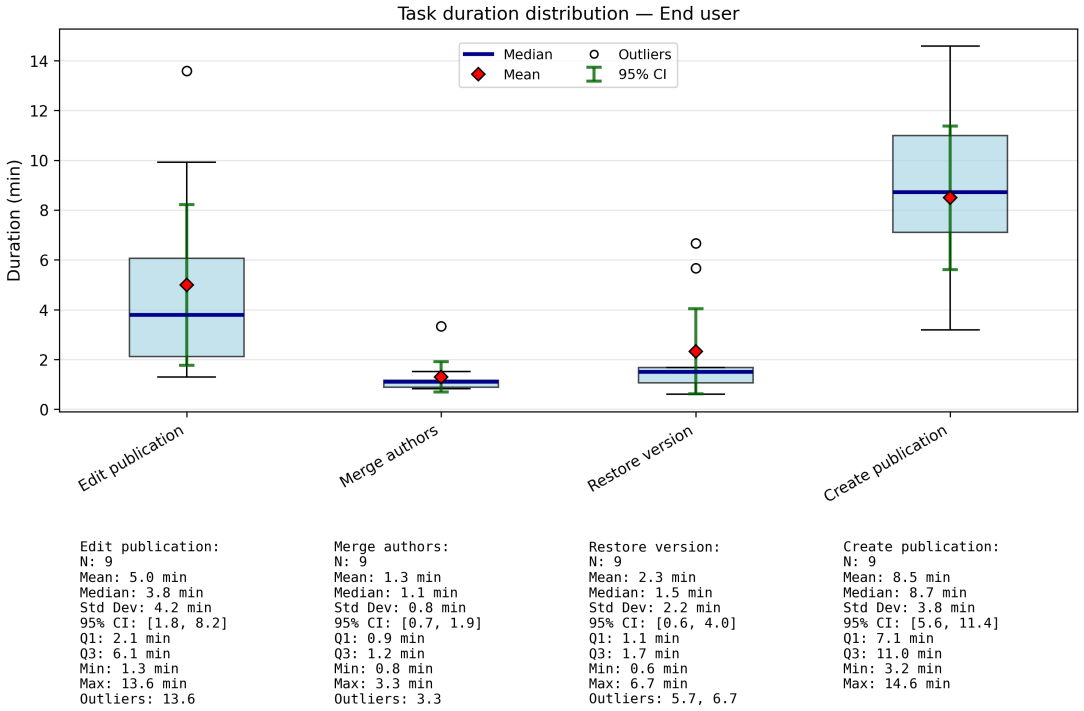


Figure 7.3: End user task duration distributions showing completion time variability across metadata management tasks, with box plots indicating median, mean, quartiles, and 95% confidence intervals.

(1977): boxes span the interquartile range (IQR) with median lines and mean diamonds, whiskers extend to 1.5×IQR, and observations beyond this threshold appear as individual outlier markers. Error bars display 95% confidence intervals for the mean, calculated using the Student's t-distribution to account for small sample sizes (Sauro, 2016).

End user task durations reveal distinct patterns across task types. The merge authors task demonstrated the most rapid completion with a median of 1.1 minutes and IQR from 0.9 to 1.2 minutes. The narrow confidence interval reflects consistent performance. One outlier appears at 3.3 minutes, representing a participant who spent additional time exploring adjacent resources to verify whether the two entries represented the same person or homonyms, as the available contextual information was insufficient to make this determination confidently.

The restore version task showed similarly rapid completion with a median of 1.5 minutes and IQR from 1.1 to 1.7 minutes. Two outliers appear beyond the upper whisker. The first outlier at 5.7 minutes represents a participant who completed the task without difficulty but dedicated substantial time to exploring and criticizing

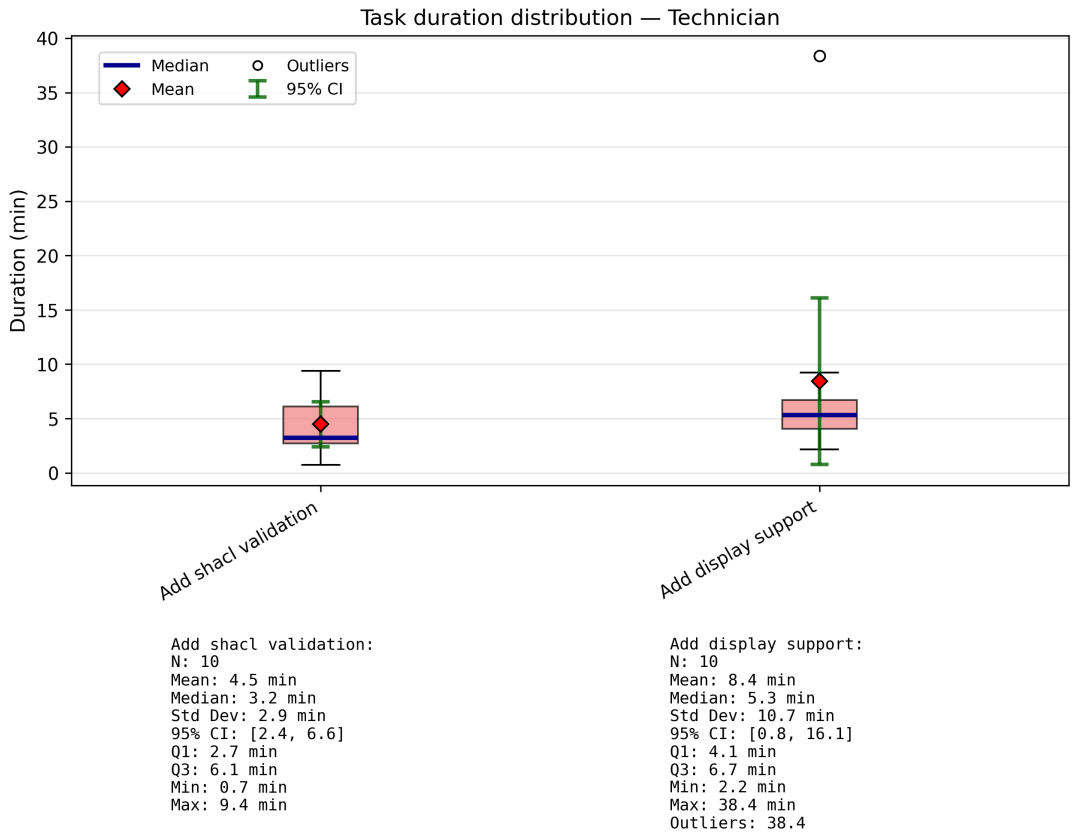


Figure 7.4: Technician task duration distributions showing completion time variability across configuration tasks, with box plots indicating median, mean, quartiles, and 95% confidence intervals.

the Time Machine interface, verbally commenting on perceived usability problems: the interface appeared confusing and cluttered, clicking "View version" opened a new page instead of a popup or new tab as expected, and the timeline started excessively zoomed requiring manual adjustment with unpredictable scrolling behavior. The second outlier at 6.7 minutes experienced a snapshot generation bug where editing the first author directly instead of adding a new author entity prevented snapshot creation, causing the restore button to become unavailable and blocking task completion, as already mentioned in subsection 7.2.1.

The edit publication task produced a median of 3.8 minutes with IQR from 2.1 to 6.1 minutes. The wider confidence interval reflects moderate variability in navigation efficiency. Two participants exceeded the time limit while actively working. One outlier appears at 13.6 minutes, representing a participant who encountered severe navigation difficulties due to absent search functionality, repeatedly expressing frustration about having to manually browse through all records to locate the

target publication, then struggled extensively to find the save button after making changes, repeatedly questioning how to save before eventually discovering it. The second timeout case at 9.9 minutes occurred when a participant initially added the author in the wrong position in the author sequence instead of as first author as required by the task, requiring Time Machine restoration to undo the error and retry, then struggled with the author ordering mechanism across multiple attempts and faced save button discoverability issues before eventually completing the task.

The create publication task yielded the longest median duration at 8.7 minutes with IQR from 7.1 to 11 minutes, reflecting the complexity of multi-step entity creation. The wide confidence interval indicates substantial variability. The longest completion was 14.6 minutes. This participant experienced confusion with the entity selection mechanism, where the interface allows selecting existing entities from the database instead of creating new ones. The participant appeared to expect automatic saving after entity selection and was confused by the display of provisional entities prior to explicit save button activation, leading to cancellation of automatic selections and manual data re-entry despite ultimately selecting existing entities, substantially increasing task duration. Additionally, this participant could not locate appropriate fields for volume and issue number input, suggesting potential domain knowledge gaps regarding the hierarchical containment structure where journal articles are contained within issues, issues within volumes, and volumes within journals. Given that the system targets domain experts, this difficulty may indicate a participant profile slightly outside the intended user base. The participant also reported finding the hierarchically nested form structure overwhelming and excessively complex.

The add SHACL validation task demonstrated a median of 3.2 minutes with IQR from 2.7 to 6.1 minutes. Most participants completed in under four minutes. No extreme outliers appear beyond the whiskers, though one participant required 9.4 minutes. This participant initially expected to configure the system through the web interface, exploring the interface by clicking on edit resource and new record buttons in an attempt to locate configuration options. After this exploration proved unsuccessful, the participant realized that configuration must be performed through direct file editing and successfully navigated to the SHACL file to complete the task.

The add display support task showed a median of 5.3 minutes with IQR from 4.1 to 6.7 minutes. One extreme outlier at 38.4 minutes achieved only success timeout status. As already mentioned in subsection 7.2.1, this participant placed the abstract display configuration in the wrong YAML section (journal instead of journal article), causing an extended troubleshooting period to determine why the abstract field was

not appearing in the interface.

Across all tasks, the observed variability in task durations reflects specific obstacles rather than generic individual differences: absent search functionality, save button discoverability issues, snapshot generation bugs, time machine interface friction, entity creation visibility gaps, identifier pattern recognition failures, nested form context loss, and YAML section misplacement. The detailed case analysis above traced each outlier duration to one of these system issues rather than participant characteristics such as technical background or prior experience.

7.2.3 Error analysis

Figure 7.5 shows error counts and severity-weighted scores across participants and tasks. Severity weighting uses a three-level scale: low (weight 1, minor friction), medium (weight 2, workarounds required), and high (weight 4, task blocking or restart required). This section focuses on the highest-concentration cells in the heatmaps, adding detail beyond what the completion and duration analyses already covered.

The highest end user score (17, create publication) was caused by a system bug: leaving the optional *name* field empty while populating *given name* and *family name* created an entity displaying "None," because the display query prioritized the null value over the populated name components. The bug cascaded into repeated save failures, inability to delete the malformed entity, and the created publication becoming invisible in the catalogue.

Two other high-concentration cases during create publication (scores of 8 and 7) share a common pattern: participants expected capabilities the system does not provide. One expected authors to exist in the local database or to be fetched from external registries via ORCID lookup; the other expected a dedicated ISSN field rather than the generic identifier mechanism (subsection 7.2.1). Additional friction arose from the "add manually" button required when no database match exists (an extra step that felt redundant), from HTML5 date inputs inheriting format from browser locale rather than displaying a consistent format, from interface slowdowns during deeply nested form interactions, and from autocomplete returning irrelevant suggestions.

During edit publication, two participants reached scores of 7–9. The causes overlap with those documented in subsection 7.2.2: absent search functionality forcing manual catalogue browsing, save button not immediately visible, and unclear visual feedback after selecting an existing entity from autocomplete (participants could not tell whether the selection had taken effect, leading some to cancel and re-enter

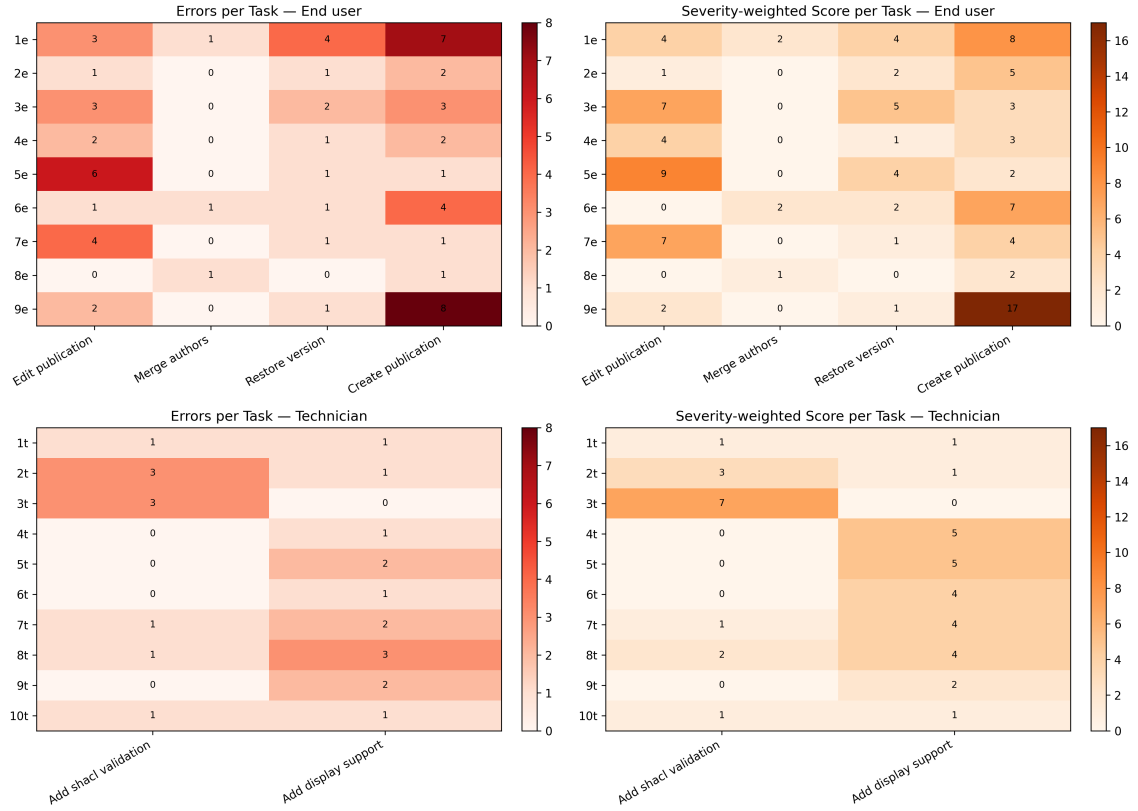


Figure 7.5: Error analysis heatmaps showing error counts (left) and severity-weighted scores (right) across participants and tasks, highlighting error concentration patterns.

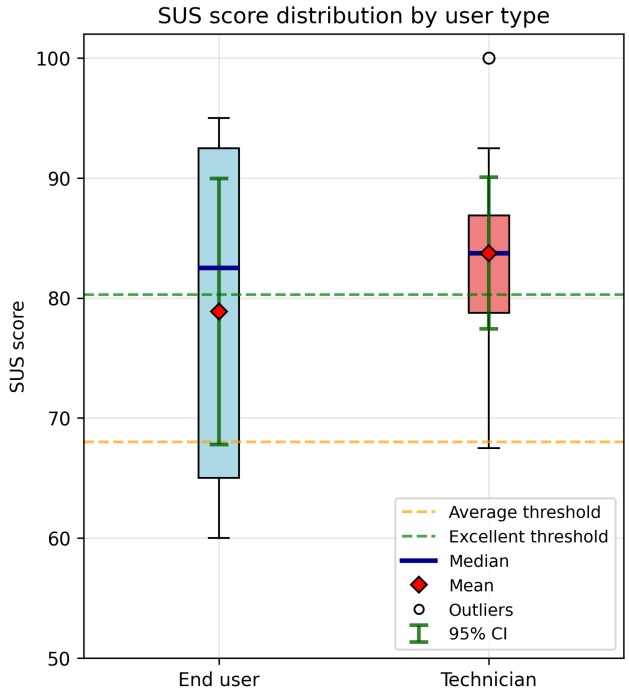
data manually). One participant also remained uncertain whether author reordering required an explicit save or was persisted automatically.

The restore version errors (two participants) and the merge task (zero or minimal errors) were already characterised in subsections 7.2.1 and 7.2.2.

Technician errors were fewer and less severe. The highest score (7, SHACL validation) stemmed from a task comprehension error: the participant set minimum cardinality to 1 instead of 0, making the abstract required rather than optional (subsection 7.2.1). Two participants scored 5 during display configuration after YAML syntax errors caused complete application crashes. The file-based configuration approach provides no validation feedback: a misplaced colon or indentation error triggers an unrecoverable failure requiring container restart. Both participants also experienced uncertainty about non-self-evident configuration key names, though they resolved this by inspecting existing patterns.

7.2.4 System Usability Scale results

SUS score distributions are visualized using box plots, following the same interpretation framework described in subsection 7.2.2. Figure 7.6 presents the distribution of SUS scores for both user populations alongside descriptive statistics.




```
SUS statistics summary

End user:
  N: 9
  Mean: 78.9
  Median: 82.5
  Std Dev: 14.4
  95% CI: [67.8, 90.0]
  Q1: 65.0
  Q3: 92.5
  Min: 60.0
  Max: 95.0

Technician:
  N: 10
  Mean: 83.8
  Median: 83.8
  Std Dev: 8.8
  95% CI: [77.4, 90.1]
  Q1: 78.8
  Q3: 86.9
  Min: 67.5
  Max: 100.0
  Outliers: 100.0
```

Figure 7.6: SUS score distributions by user type, showing median, mean, quartiles, and 95% confidence intervals with benchmark thresholds for average (68) and excellent (80.3) usability.

End users produced a mean SUS score of 78.9 (median = 82.5, range = 60–95). The 95% confidence interval for the mean spans from 67.8 to 90.0. The distribution shows substantial variability (standard deviation 14.4) with scores ranging across 35 points. The median of 82.5 exceeds the mean of 78.9, revealing that the distribution is negatively skewed with scores extending more toward the lower end. This pattern reflects divergent experiences where some participants encountered friction points while others found the system highly usable. The mean score of 78.9 places end user usability in the above-average range (68–80.3), approaching but not reaching the excellent threshold of 80.3.

Technicians produced a mean SUS score of 83.8 (median = 83.8, range = 67.5–100). The 95% confidence interval for the mean spans from 77.4 to 90.1, demonstrating higher precision than the end user population. The distribution shows lower variability compared to end users (standard deviation 8.8 versus 14.4), though the

range spans 32.5 points. One outlier is visible in the distribution at 100, representing a technician who completed both configuration tasks successfully and achieved the maximum possible SUS score. The lower bound of 67.5 corresponds to a technician who encountered a YAML syntax error causing complete application crash during display configuration, as documented in subsection 7.2.3. The lower standard deviation suggests that most technicians experienced more consistent usability levels compared to end users, despite varying SHACL expertise levels reported in subsection 7.1.1. The mean score of 83.8 exceeds the excellent usability threshold of 80.3, indicating that the configuration layer achieves high perceived usability for technical audiences.

Across both populations, 12 of 19 participants (63%) scored in the excellent range, 3 (16%) in the above-average range, and 4 (21%) below average. Each below-average score traces to specific obstacles documented in subsections 7.2.1–7.2.3: three end users (scores 60, 62.5, 65) encountered combinations of absent search functionality, save button discoverability failure, snapshot generation bugs, and nested form context loss, while one technician (67.5) experienced a YAML syntax error that crashed the application with no validation feedback. The correspondence between low SUS scores and documented obstacles confirms that perceived usability reflects concrete system deficiencies rather than individual preference variation.

7.3 Qualitative analysis: grounded theory

The quantitative results confirm that HERITRACE works (high completion rates, favourable SUS scores) but do not explain how users navigate the system or why specific tasks prove more challenging than others. Grounded theory analysis addresses these questions by deriving explanatory theories from user behaviour observations, think-aloud protocols, and written reflections.

The analysis followed the three-phase coding procedure described in subsection 7.1.4. Open coding produced distinct concepts from line-by-line analysis of transcripts and written responses. Axial coding grouped these concepts into 16 categories for end users and 17 for technicians, each with consistent internal sentiment (positive or negative); neutral codes were excluded to focus on experiences that directly affect adoption. Selective coding synthesized the categories into core theories grounded in the complete dataset.

7.3.1 End user experience

The axial coding produced 146 positive and 114 negative codes, distributed across 16 categories (Figure 7.7). The co-existence of strong positive and negative experiences points to a dual-phase adoption pattern: users first navigate complexity barriers, then achieve functional mastery once they develop working mental models of the system.

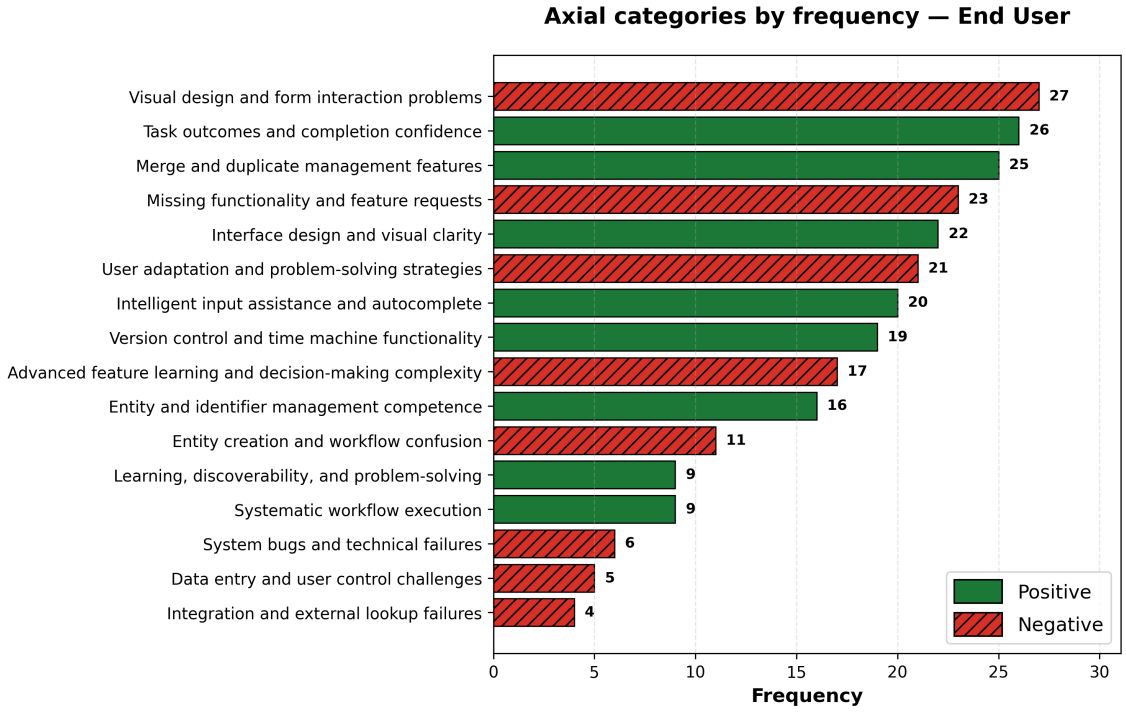


Figure 7.7: End user axial coding categories by frequency, showing positive (solid) and negative (hatched) sentiment distributions across interaction dimensions.

Three capabilities generated the strongest positive responses. Merge and duplicate management (25 codes) was the only feature with unanimously positive reception, consistent with its 100% task success rate. Users appreciated the transparency of the merge interface (which entity is retained, which is deleted) and the ability to consolidate records without manual data transfer. One participant went beyond the assigned task to explore additional author records, verifying whether similar entries were true duplicates or homonyms, which suggests the merge workflow would benefit from richer contextual information. The Time Machine (19 codes) functioned as a safety net: participants discovered it through exploration, correctly inferred its purpose, and used it for error recovery. One participant explicitly noted that knowing versions could be restored reduced anxiety about making mistakes, a confidence-building effect that enabled the successful retry pattern documented

in subsection 7.2.1. Autocomplete and entity disambiguation (20 codes) received appreciation for detecting existing authors and offering selection rather than duplication, and for ORCID-based lookup, though some participants expected the system to query external registries rather than only the local database.

Beyond individual features, participants reported growing competence with the semantic model through interaction (16 codes): they learned to select resource types, create hierarchical relationships (article to issue to volume to journal), and transfer knowledge between tasks. Interface design and visual clarity (22 codes) drew positive first impressions: clear layout, logical field grouping matching bibliographic structure, and intuitive catalogue presentation. Task completion confidence (26 codes) and systematic workflow execution (9 codes) reflect the mastery phase, where users expressed satisfaction with accomplished results and adopted methodical field-by-field approaches once they understood system mechanics.

The most frequent negative category is visual design and form interaction problems (27 codes). Collapsed sections hide already-populated fields, causing context loss; provisional entities appear before explicit save, creating uncertainty about whether selections took effect; and deeply nested entity creation forms degrade both orientation and performance. One participant characterised the nested structure as overwhelming. Date format inconsistencies compound the confusion: the HTML5 date input inherits its format from browser locale, producing unexpected layouts that users attributed to system errors.

Missing search functionality (23 codes) was the most consistently reported gap: six of nine end users requested it. Users approached HERITRACE expecting search-oriented discovery, but the system's editing-focused design offers only catalogue browsing. The resulting manual scanning through paginated records generated persistent frustration and workaround strategies (21 codes): participants scrolled through all records to locate targets, created manual entry fallbacks when autocomplete failed, and deleted and recreated entities when editing proved difficult. One participant experienced severe disorientation and negative self-assessment when unable to locate resources. Beyond search, users requested collapsed section summaries to maintain awareness of hidden content, and bottom pagination controls.

The Time Machine, despite its positive reception once understood, created initial confusion (17 codes) through terminology mismatches ("Time Vault" vs. user expectations), cluttered timeline interfaces, and version viewing that opened new pages instead of modals (subsection 7.2.2). Entity creation workflow confusion (11 codes) centred on uncertainty about save timing and cumbersome multi-step processes. Smaller categories captured concerns about loss of agency when autocomplete pop-

ulated fields without preview (5 codes), failed expectations about external service integration (4 codes), and genuine bugs (6 codes) distinct from usability issues, documented in subsections 7.2.1 and 7.2.3.

The 260 codes converge on a substantive theory of dual-phase adoption. End users encounter initial complexity barriers from entity lifecycle confusion, visual design inconsistencies, missing expected functionality, and a mismatch between search-oriented discovery expectations and the system's editing-focused design. Those who persist through the learning curve discover merge operations, version control, and intelligent input assistance, reaching functional success reflected in high completion rates and favourable SUS scores. Adoption depends on navigating usability barriers and developing autonomous problem-solving strategies rather than on overcoming system capability limitations. Genuine bugs create a separate class of technical failures distinct from learnable complexity.

7.3.2 Technician experience

The axial coding produced 172 positive and 91 negative codes, distributed across 17 categories (Figure 7.8). The ratio is markedly more positive than for end users, reflecting a population that brings domain knowledge and problem-solving habits well suited to the configuration task.

The dominant strategy was pattern replication (27 codes): all participants located an existing SHACL property or YAML rule similar to what they needed, copied it, and adapted it. This copy-paste-modify workflow proved effective regardless of prior SHACL experience, because the configuration files contain enough inline examples to serve as templates. Semantic reasoning (22 codes) complemented the strategy: technicians translated natural language requirements into SHACL constraints, inferred datatypes from property semantics, and verified namespace availability before use. Immediate visual feedback (12 codes) closed the loop. Automatic backend hot-reloading of SHACL and YAML files (requiring only a manual browser refresh) eliminated container restarts, enabling rapid hypothesis testing that reinforced the pattern replication cycle. All technicians verified their changes through the UI (30 codes), and several characterised configuration as easier than expected despite encountering errors along the way. One participant stated that mistakes were their own fault rather than system complexity.

Technicians also explored the end user interface during warm-up. Feature appreciation and design quality (38 codes) and intuitive navigation (36 codes) were the two most frequent positive categories: participants praised provenance tracking, the Time Machine, orphan prevention logic, and the minimalist layout with clear

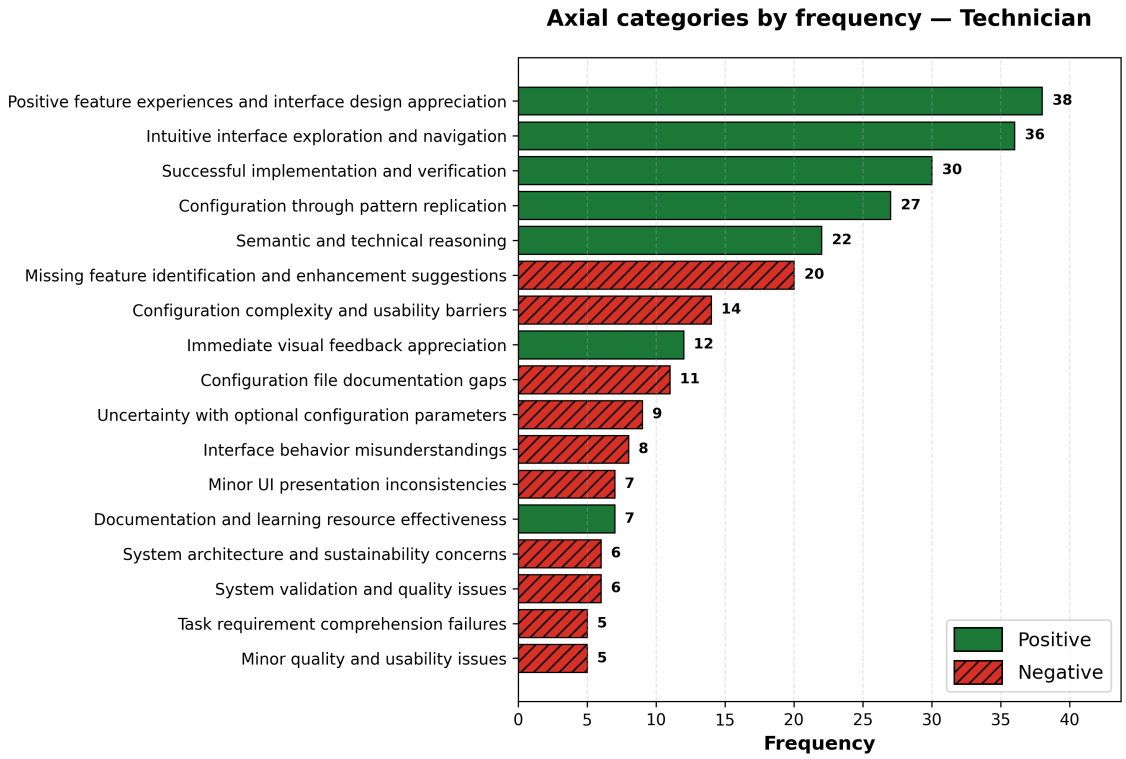


Figure 7.8: Technician axial coding categories by frequency, showing positive (solid) and negative (hatched) sentiment distributions across configuration and interface interaction dimensions.

information hierarchy. This appreciation aligns with the 83.8 mean SUS score exceeding the excellent threshold (subsection 7.2.4). Participants with no prior SHACL knowledge found the documentation beginner-friendly (7 codes), though one characterised the system as self-explanatory, needing documentation only to locate files rather than to understand syntax.

The most frequent negative category (20 codes) collected feature requests: search functionality, SHACL validation tooling, contextual UI help, a visual editor for configuration, and a back navigation button. These are gaps rather than failures: participants identified them while successfully completing tasks.

More substantive friction arose from the file-based configuration paradigm itself (14 codes). The web-based editing interface set an expectation of web-based configuration, so several participants initially looked for configuration options inside the UI before realising that files must be edited directly. Participants identified an iterative experimental mindset (modify, reload, inspect) as a core competence requirement beyond any specific technology. Documentation gaps (11 codes) compounded this: the separation between SHACL validation and YAML display rules was not imme-

diately obvious, and some participants expected display rules to auto-generate from SHACL changes. Optional parameters (9 codes) caused further uncertainty: display property ordering effects, whether to omit or explicitly set optional fields, and non-self-evident key names for search configuration.

Temporary mental model gaps during interface exploration (8 codes) included misinterpreting pagination controls, reading the cascade deletion dialog as an all-or-nothing choice rather than a selective one, and interpreting unconfigured entities (missing labels) as UI bugs rather than configuration responsibilities. Minor polish issues (7 codes) and task comprehension errors (5 codes, such as setting the abstract as required instead of optional) did not prevent completion. Genuine bugs (6 codes) included validation failures, cryptic error messages, and application crashes after YAML syntax errors, consistent with the error analysis in subsection 7.2.3.

The most thought-provoking finding came from a small but pointed category (6 codes). Several technicians identified a paradox: the user-facing interface is intuitive, but the configuration layer that produces it is not. In practice this means technical experts perform initial setup while domain experts handle daily curation. However, real-world use is iterative: domain experts will discover missing features and need configuration changes they cannot make themselves. One participant questioned whether continuous technical support is a sustainable expectation at the project level; another suggested CMS-style configuration interfaces or UI-driven generation as possible mitigations.

The 263 codes converge on a theory of expert-driven configuration through pattern-based learning. Technicians overcome file-based configuration constraints and documentation gaps by leveraging discoverable patterns in existing files, semantic reasoning from their Semantic Web background, and rapid feedback loops that confirm correct implementation. Even participants with no prior SHACL knowledge completed tasks successfully. However, this success depends on domain knowledge and a problem-solving disposition that non-technical users may lack, creating a persistent collaboration requirement between experts and non-experts that individual projects may struggle to sustain over time.

7.4 Addressing the research questions

Table 7.1 synthesizes the relationship between requirements identified through empirical analysis in Chapter 3, their implementation in HERITRACE as described in Chapter 6, and the evaluation outcomes documented in the preceding sections.

The evaluation findings provide evidence for both research questions.

Table 7.1: Mapping between requirements, design implementations, and evaluation outcomes.

Requirement	**Design implementation**	**Evaluation task**	**Outcome**
Provenance and change tracking	Automatic recording of agents, timestamps, and sources; snapshot-based versioning; Time Machine and Time Vault interfaces	Restore version (Task 3, end users)	78% success; valued as safety net enabling error recovery
Domain expert usability	Form-based interfaces abstracting RDF complexity; autocomplete for entity disambiguation; merge workflow for duplicate consolidation	All end user tasks (Tasks 1–4)	67–100% success; SUS 78.9 (above-average); merge task achieved 100% success rate
Flexible customization	SHACL for data model definition and validation; YAML for interface presentation and behavior control	Technician tasks (Tasks 1–2)	90% success; SUS 83.8 (excellent); pattern-based learning effective without prior SHACL knowledge
Integration with existing RDF collections	Direct connection to RDF quadstores via SPARQL; automatic entity discovery through RDF types	Evaluation setup	OpenCitations Meta subset used as test data without migration or format conversion

For RQ1, end users without RDF expertise completed metadata curation tasks at 67–100% success rates depending on task complexity. The system abstracts Semantic Web complexity while automatically maintaining provenance documentation and change tracking, as demonstrated by successful version restoration through the Time Machine feature and 100% success on merge operations. The mean SUS score of 78.9 indicates above-average usability for the daily curation workflow.

For RQ2, technicians configured HERITRACE through SHACL validation rules and YAML display settings, achieving 90% task completion. The evaluation used an OpenCitations Meta subset as test data, confirming integration with existing RDF repositories. The mean SUS score of 83.8 indicates excellent usability for initial system setup.

However, the evaluation also revealed systematic limitations that require acknowledgment. Entity creation workflows generated the highest frequency of neg-

ative feedback from end users, with nested form structures causing context loss and confusion about save timing, indicating that while domain experts can participate in curation, specific interaction patterns require refinement. The absence of search functionality created persistent friction, with six of nine end users explicitly requesting this capability, demonstrating that the editing-focused architectural choice conflicts with user expectations for integrated discovery. The file-based configuration paradigm proved effective for technically proficient administrators but raised sustainability concerns articulated by technicians themselves, who questioned whether continuous technical support for configuration modifications represents a viable long-term model, suggesting that flexible customization capability remains contingent on technical expertise availability. Additionally, system bugs prevented some task completions, indicating that technical implementation requires further stability improvements.

Chapter 8

Extending HERITRACE to non-RDF data sources: preliminary experimentation on RML mapping inversion

The evaluation presented in Chapter 7 demonstrates that HERITRACE successfully enables domain experts to participate in semantic data curation through user-friendly interfaces that hide RDF complexity. However, hiding the complexity of RDF addresses only part of the challenge identified in RQ1. The most effective approach to enabling domain expert participation is not merely simplifying RDF interaction, but hiding the presence of RDF entirely by maintaining transparent connections with the data formats institutions already use.

The reality of institutional data management reveals that pre-existing collections rarely exist in RDF format. Many galleries, libraries, archives, and museums maintain their collections in relational database systems, with CSV exports serving as the primary data interchange format (Stefanova & Risch, 2013; Tamper, 2023). Domain experts in these institutions work daily with spreadsheets and database interfaces, developing proficiency with tabular data representations rather than RDF triples and SPARQL queries. Requiring these institutions to migrate their data to RDF before benefiting from semantic curation capabilities creates an adoption barrier that contradicts the thesis objective of enabling domain expert participation.

This chapter presents preliminary experimentation on extending HERITRACE's applicability by enabling bidirectional transformations between relational data and RDF. The approach leverages the RDF Mapping Language (RML) (Dimou et al., 2014), which provides a W3C-standardized mechanism for converting structured

data sources including relational databases and CSV files into RDF graphs. While RML supports the forward transformation from structured data to RDF, the reverse process, i.e., reconstructing the original relational structures from RDF graphs, presents significant technical challenges. This chapter proposes an algorithm for inverting RML mappings, enabling the reconstruction of relational database structures from RDF graphs.

8.1 Background

The RDF Mapping Language (RML) (Dimou et al., 2014) has become a central tool in transforming various data sources, including relational databases, into RDF graphs. RML extends the RDB to RDF Mapping Language (R2RML) (W3C, 2012b) originally designed to generate RDF from relational databases, by incorporating support for additional formats like CSV and JSON.

Despite the effectiveness of RML in mapping various data sources to RDF, the inverse process of reconstructing original data sources from RDF graphs remains a complex challenge. Addressing this gap, RML2CSV (Allocca & Gougousis, 2015) was introduced to reverse RDF datasets back into their original CSV tabular structures using the same RML mappings that generated the RDF graphs. However, this algorithm exhibits several limitations that constrain its applicability. It relies on the *Dependency Tree Assumption*, requiring RML mappings to form a single n-ary tree with clear parent-child relationships, which restricts its use to strictly hierarchical data structures. Additionally, it assumes implicit cardinality constraints of 0:0 or 1:1 between CSV columns, making it inadequate for handling one-to-many or many-to-many relationships commonly found in real-world datasets. The algorithm also struggles with blank nodes and complex mappings, such as nested structures or composite keys, due to difficulties in associating data lacking explicit identifiers.

Another approach to the inversion of RDF data back into relational databases precedes the introduction of RML. R2D (RDF-to-Database) (Ramanujam, Gupta, Khan, Seida, & Thuraisingham, 2009) aimed to generate RML-like mappings from RDF data by automatically inferring schema information, also leveraging OWL ontologies, when available. Unfortunately, the source code for R2D is not publicly available, which limits the direct reuse. However, the conceptual approach remains relevant and can be extended to use other contextual elements. While in R2D OWL ontologies were used, in this case information present in the original relational database could be utilized if available.

An alternative approach to RML for accessing heterogeneous data in a format-

agnostic manner is SPARQL Anything (Asprino, Daga, Gangemi, & Mulholland, 2023), a system that allows querying heterogeneous resources as RDF through query-time virtualization. The system operates by overloading the SPARQL SERVICE clause, intercepting queries during execution and materializing temporary RDF views using the Façade-X meta-model. SPARQL Anything supports a wide range of formats including JSON, XML, HTML, YAML, Markdown, Bibtex, binary files, archives, plain text, spreadsheets, word processing documents, presentation slides, and CSV. The system extends Apache Jena with custom triplifiers that transform input sources into RDF representations on demand, enabling users to query heterogeneous data sources using standard SPARQL 1.1. One may wonder why this work extends RML rather than SPARQL Anything. Two factors make RML the more appropriate foundation for this work. First, the focus of this chapter is bidirectional transformation between relational databases and RDF. While the Façade-X meta-model has been formally proven capable of representing relational databases, the SPARQL Anything implementation currently supports only static files, with relational database connectivity explicitly identified as future work. Extending SPARQL Anything for this work would therefore require implementing both relational database support and inversion capabilities. Second, inverting a transformation requires access to persistent specifications that encode the transformation logic. SPARQL queries are inherently selective and projective: they describe which data to extract through WHERE clause filters and which variables to project in SELECT clauses, not how to systematically transform complete datasets. RML provides persistent declarative mapping documents that explicitly encode transformation logic through logical sources, subject map templates, predicate-object maps, and join conditions, making it suitable for bidirectional transformation work.

8.2 Methodology

The software implementation of the algorithm described in this chapter is available as open source under the ISC license (Massari, 2025a). The repository includes also the evaluation code, enabling full reproduction of both the conformance tests and the benchmark for scalability assessment described in Section 8.3.

The core of the inversion process relies on a detailed analysis of the RML mappings. This analysis provides the information necessary to reverse-engineer the original relational database from the RDF graph. To illustrate this process, the following sections use the RML mapping example in Listing 27, analyzed step by step.

The analysis begins with the identification of the logical source specified in the

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.com/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@base <http://example.com/base/> .

<#TriplesMap1>
    a rr:TriplesMap;

    rr:logicalTable [ rr:tableName "\"Student\"" ];

    rr:subjectMap [ rr:template "http://example.com/{\"ID\"}/{\"Name\"}";
                    rr:class foaf:Person ];

    rr:predicateObjectMap
    [
      rr:predicate    ex:id ;
      rr:objectMap    [ rr:column "\"ID\""; ]
    ];

    rr:predicateObjectMap
    [
      rr:predicate    foaf:name ;
      rr:objectMap    [ rr:column "\"Name\"" ]
    ] .
```

Listing 27: RML mapping example for student table.

RML mapping. Logical sources in RML typically reference either direct table names or views. This distinction is critical, as it dictates how the RDF data was originally generated, and therefore influences how the graph can be reverted to its relational form. Listing 28 shows the logical source declaration from Listing 27, which is a direct table reference to the "Student" table.

```
    rr:logicalTable [ rr:tableName "\"Student\"" ];
```

Listing 28: Logical source declaration in RML mapping.

When the logical source points to a table, the table name is immediately extracted. This provides a straightforward path to reconstructing the original relational schema, as the RML mappings specify a one-to-one relationship between the table and its corresponding RDF triples. In this case, the RDF graph can be directly mapped back to its table without needing to infer additional structural information.

If the logical source is a view, while it's technically expressed as a SQL query, it represents a virtual table. The process of analyzing a view depends on whether the original database is available. When the original database is available, the view definition can be extracted directly, providing complete information about

how the view was constructed, including the underlying tables, join conditions, and any transformations applied, allowing for more accurate reconstruction of the original data structure and relationships. In the absence of the original database, the SQL query defining the view must be analyzed by parsing it to identify the tables involved, their relationships, and any transformations applied to the data (*e.g.*, aggregations, filtering, or column renaming). However, the current implementation does not support SQL queries as logical sources.

After analyzing the logical source, the algorithm processes the triple maps. This involves handling the subject, predicate, and object maps to construct appropriate SPARQL query patterns. The processing is tailored to different mapping types: constants, references, and templates. Figure 8.1 illustrates the flowchart for subject map processing.

For each logical source identified during the mapping analysis, the algorithm examines the subject map to determine how subjects were constructed in the RDF graph. When the subject map specifies a constant IRI, the algorithm directly binds this value in the query, associating it with the `?subject` variable without further manipulation. For column references, the algorithm introduces a variable representing the column and includes a triple pattern in the WHERE clause that matches the subject variable with the appropriate RDF type or class as specified in the mapping, allowing retrieval of all subjects corresponding to rows in the original table with the variable capturing unique identifiers or keys. When subjects are constructed using templates, the algorithm employs string manipulation functions to reverse-engineer the original column values from the subject IRI. Listing 29 shows a template-based subject map that combines the "ID" and "Name" columns into the subject IRI and assigns the class `foaf:Person`.

```
    rr:subjectMap [ rr:template "http://example.com/{\"ID\"}/{\"Name\"}";
                    rr:class foaf:Person ];
```

Listing 29: Subject map declaration in RML mapping.

The algorithm binds the subject IRI to the `?subject` variable and then uses functions like `STRAFTER` and `STRBEFORE` to extract the `ID` and `Name` components, as shown in Listing 30:

These extracted values are then associated with variables corresponding to the original columns, enabling the retrieval of the ID and Name values from the RDF data.

The processing of predicate maps follows the same approach described for subject maps. Similarly, object map processing applies the same constant, reference,

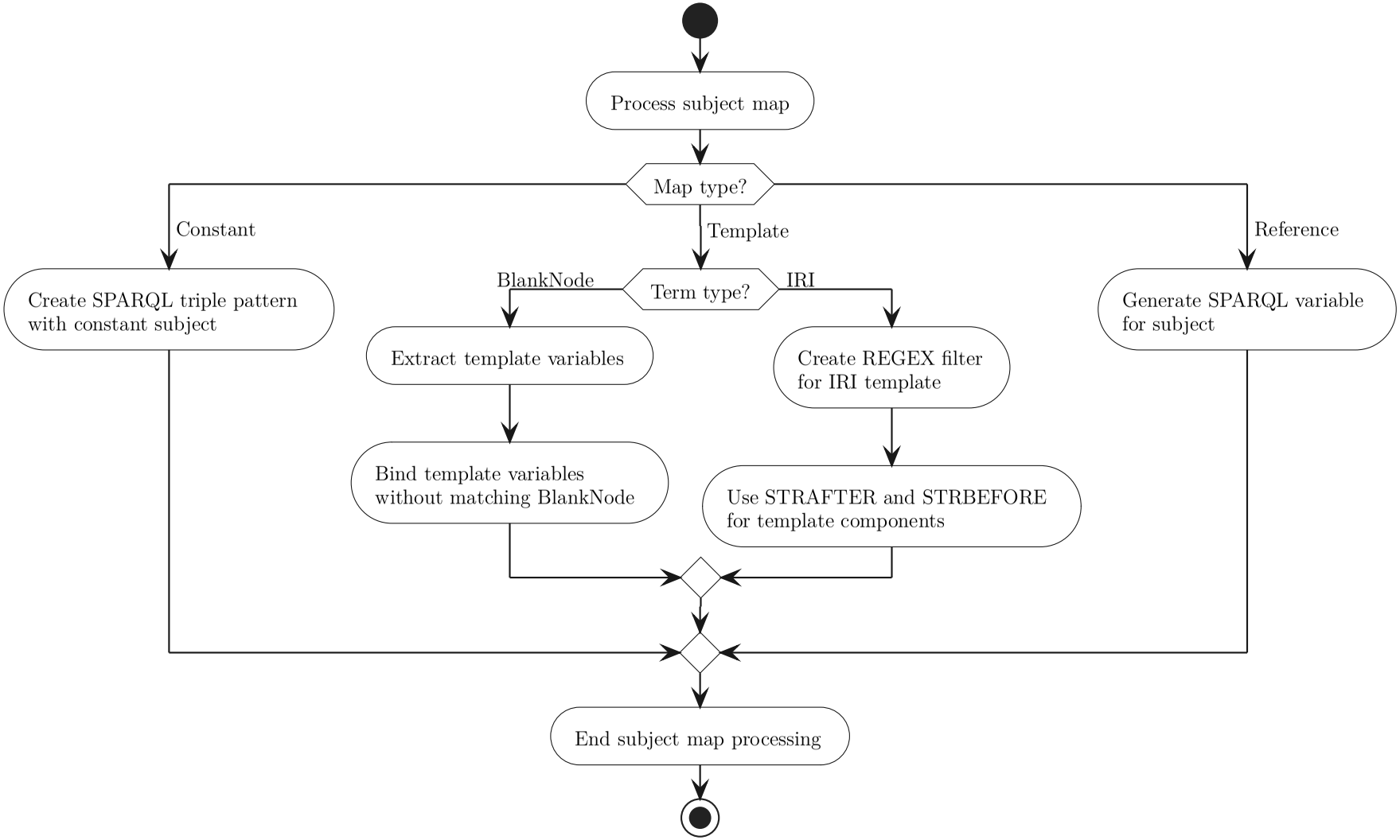


Figure 8.1: Flowchart of subject map processing.

```
BIND(STRAFTER(STR(?subject), "http://example.com/") AS ?idname)
BIND(STRBEFORE(?idname, "/") AS ?ID)
BIND(STRAFTER(?idname, "/") AS ?Name)
```

Listing 30: SPARQL string manipulation for extracting template components.

and template patterns, with one key distinction: objects require explicit datatype preservation. For object maps using references or templates, an additional binding captures the datatype using the `DATATYPE` function in SPARQL, as shown in Listing 31:

```
OPTIONAL {
  ?subject ?predicate ?objectreference .
  BIND(DATATYPE(?objectreference) AS ?objectreferencedatatype)
}
```

Listing 31: SPARQL datatype extraction for object maps.

This ensures query results include both values and their associated datatypes when available, enabling accurate reconstruction of numeric, date, or boolean values. Without explicit datatypes, values would be misinterpreted as strings. When datatypes are not specified in the RML mapping, the algorithm retrieves this information from the original relational database schema if available.

Once all components are prepared, the algorithm assembles and executes the complete SPARQL query against the RDF graph, retrieving the data needed for relational database reconstruction.

8.2.1 Handling blank nodes

In the process of inverting RDF graphs to reconstruct relational database structures, handling blank nodes presents a unique challenge. Blank nodes can appear as subjects or objects in RDF triples, but not as predicates. Regardless of their position, the approach to processing blank nodes remains consistent.

Figure 8.2 illustrates the specific steps involved in handling blank nodes during subject or object map processing. The following example illustrates this process. Consider the R2RML mapping shown in Listing 32.

This mapping generates the RDF output shown in Listing 33.

To process blank nodes during the inversion of this RDF graph, the algorithm first identifies all blank nodes in the RDF graph regardless of their position (subject or object), then examines the identifier structure of each blank node. In Listing 33, the blank node `_:students10` contains information about the original “ID”

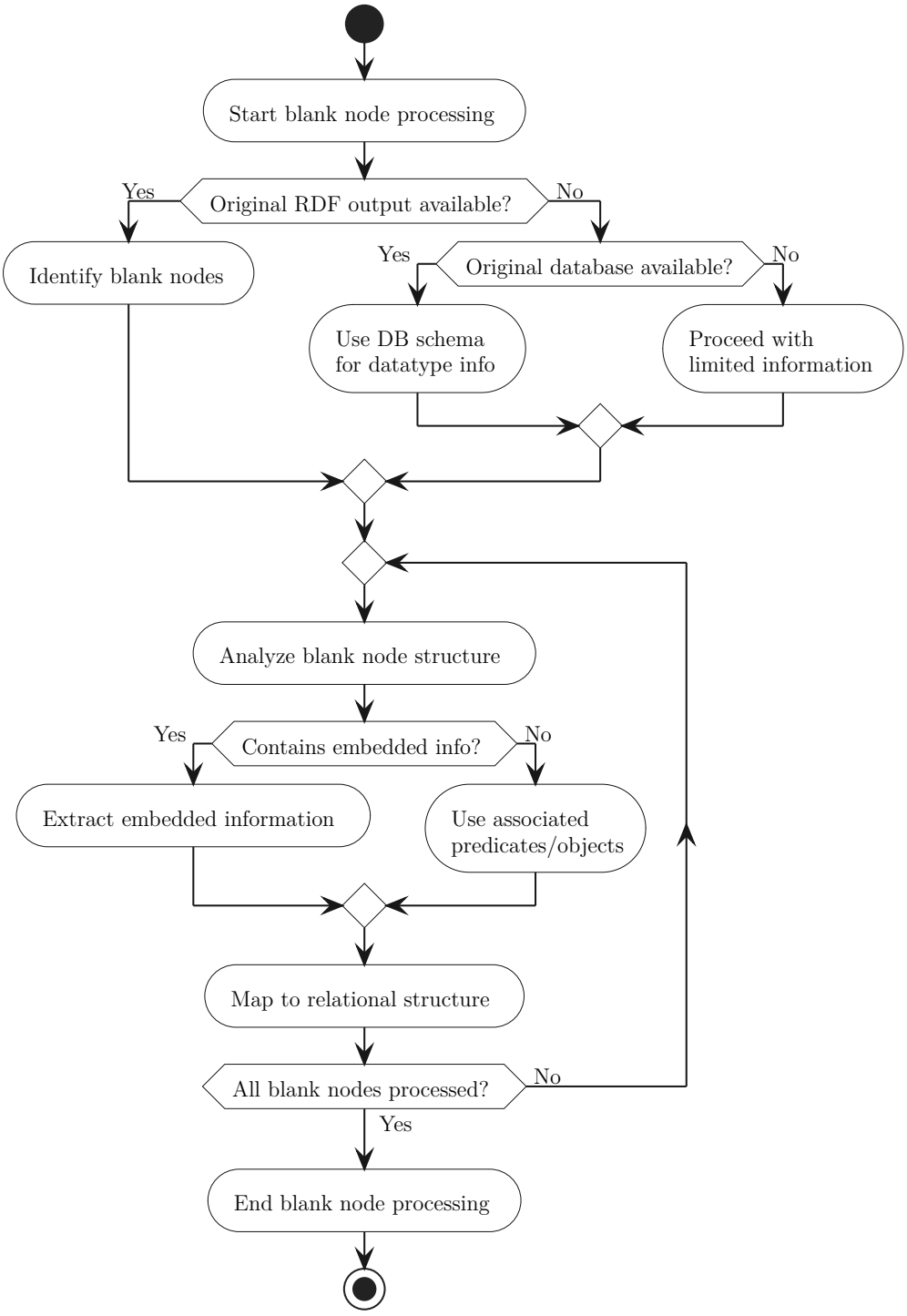


Figure 8.2: Flowchart of blank node processing.

column. When the blank node identifier contains embedded information (as with "students10"), the algorithm extracts this information, retrieving "10" as the ID value. Finally, using the collected information, the algorithm maps the blank node and its associated data to the relational structure, creating a row in the "Student" table with ID 10.

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.com/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@base <http://example.com/base/> .

<#TriplesMap1>
     a rr:TriplesMap;

    rr:logicalTable [ rr:tableName "\"Student\"" ];

    rr:subjectMap [ rr:template "students{\"ID\"}"; rr:termType
    rr:BlankNode ];

    rr:predicateObjectMap
    [
      rr:predicate    foaf:name ;
      rr:objectMap    [ rr:column "\"Name\"" ]
    ] .
```

Listing 32: R2RML mapping with blank node subject.

```
_:students10 <http://xmlns.com/foaf/0.1/name> "Venus" .
```

Listing 33: RDF output with blank node subject.

Blank nodes, by definition, are anonymous entities in RDF graphs. While they can be generated using templates based on RDB data, their anonymity presents unique challenges in the inversion process. The algorithm's approach to handling blank nodes is significantly affected by the availability of the original RDF output or the relational database. When working directly with the original RDF file, the algorithm can often utilize the information encoded in the blank node identifiers to retrieve the original values from the RDB, representing the ideal scenario for accurate inversion. However, many RDF libraries (like RDFLib) and triple stores (such as Blazegraph or Virtuoso) tend to anonymize blank nodes during processing, causing the original information used to generate the blank nodes to be lost and making accurate inversion challenging or impossible. If the original relational database is available, it can provide information about data types and structures, aiding in accurate reconstruction even when blank nodes have been anonymized.

In this specific example, in the absence of the original RDF output or the original database, it is impossible to recover the datatype information for the ID "10". While it was originally an integer in the database, it would be inverted as a string due to the lack of datatype information in the RDF representation.

8.3 Evaluation

The algorithm was evaluated through conformance testing using the W3C R2RML test suite (Villazón-Terrazas & Hausenblas, 2012), which consists of 62 test cases covering a wide spectrum of mapping scenarios. Each test case includes input data in relational format, an R2RML mapping document, and the expected RDF output. The test suite was applied in reverse: rather than generating RDF from relational data, the algorithm reconstructs relational data from the RDF output using the same R2RML mappings, comparing the reconstructed data with the original input to measure correctness.

The evaluation was conducted using PostgreSQL as the database system. Of the 62 test cases, 23 passed successfully (37.1%), 6 failed (9.68%), 17 involved unsupported features (27.42%), 14 revealed mapping quality issues (22.58%), and 2 contained invalid mappings that were correctly rejected (3.23%).

The passed tests validate the algorithm's capability across diverse R2RML patterns. The algorithm correctly handles blank node generation with embedded column information, template-based subject and literal construction using SPARQL string manipulation to reverse templates, named graph distribution with triples across multiple graphs, multiple class typing through `rr:class`, graph names constructed from column values, and preservation of SQL datatypes during reconstruction. The algorithm also successfully reconstructs many-to-many relationships, where intermediate junction tables containing foreign key pairs are correctly regenerated from RDF representations that model such relationships through referencing object maps.

The unsupported features represent fundamental architectural limitations. The primary limitation involves SQL queries as logical tables through `rr:sqlQuery`, which account for the majority of unsupported tests. Inverting such queries would require parsing arbitrary SQL with joins, aggregations, and transformations, determining base tables, and reconstructing pre-transformation data, a problem that is computationally complex and often unsolvable uniquely.

Mapping quality issues represent a distinct category where R2RML mappings lack information for complete reconstruction. Duplicate rows are lost when tables without primary keys use subject templates that do not include unique identifiers, causing rows with identical values to collapse into single RDF subjects that cannot be distinguished during inversion. Unmapped columns that are omitted from R2RML mappings cannot be reconstructed as no corresponding RDF triples exist. Mappings consisting entirely of constants contain no column-derived information and cannot be inverted.

The algorithm correctly detects and rejects invalid R2RML mappings, including mappings that attempt to use literals as graph names instead of IRIs and mappings that lack required components such as subject maps. This validation behavior prevents processing of non-conformant mappings and ensures that inversion attempts only occur on well-formed mapping documents.

Beyond conformance testing, scalability was evaluated using the KROWN benchmark (Van Assche, Chaves-Fraga, & Dimou, 2025), a framework designed to assess RML materialization systems by systematically varying dataset size and mapping complexity. It provides a data generator that produces synthetic scenarios with configurable characteristics including row count, number of triples maps, predicate-object maps, and cell size. The benchmark scenarios were adapted to test the inversion algorithm by first generating RDF from relational data using Morph-KGC (Arenas-Guerrero, Chaves-Fraga, Toledo, Pérez, & Corcho, 2024), then applying the inversion algorithm to reconstruct the original relational structure.

The implementation supports two query execution modes: in-memory processing using RDFLib (Krech et al., 2025) and triplestore-based processing. The benchmark employed Virtuoso as the SPARQL endpoint because SPARQL queries on larger in-memory graphs failed to complete within acceptable time frames. Each iteration begins by completely emptying the Virtuoso triplestore, then reloading the materialized RDF data using Virtuoso's bulk load function. The bulk load time is included in the reported materialization time.

The evaluation was conducted on a workstation with a 12th Gen Intel Core i9-12900K processor (24 cores), 125 GB of RAM, running Arch Linux (kernel version 6.17.5-arch1-1). Each benchmark scenario was executed 100 times to obtain reliable statistical measures including median, mean, standard deviation, interquartile ranges, and 95% confidence intervals.

Three scenarios were configured to assess scalability across increasing data volumes and mapping complexity. The small scenario contains 1,000 rows with 3 triples maps and 3 predicate-object maps per triples map, generating 6,000 RDF triples. The medium scenario scales to 10,000 rows with 5 triples maps and 5 predicate-object maps, producing 150,000 triples. The large scenario processes 50,000 rows with 8 triples maps and 8 predicate-object maps, resulting in 2 million triples. Each scenario varies cell size and the number of properties to simulate realistic heterogeneity in relational data structures.

Execution time distributions are visualized using box plots, following the same interpretation framework described in Section 7.2.2. Figures 8.3, 8.4, and 8.5 present the distributions for total execution time, inversion time, and inversion overhead

percentage across the three scenarios.

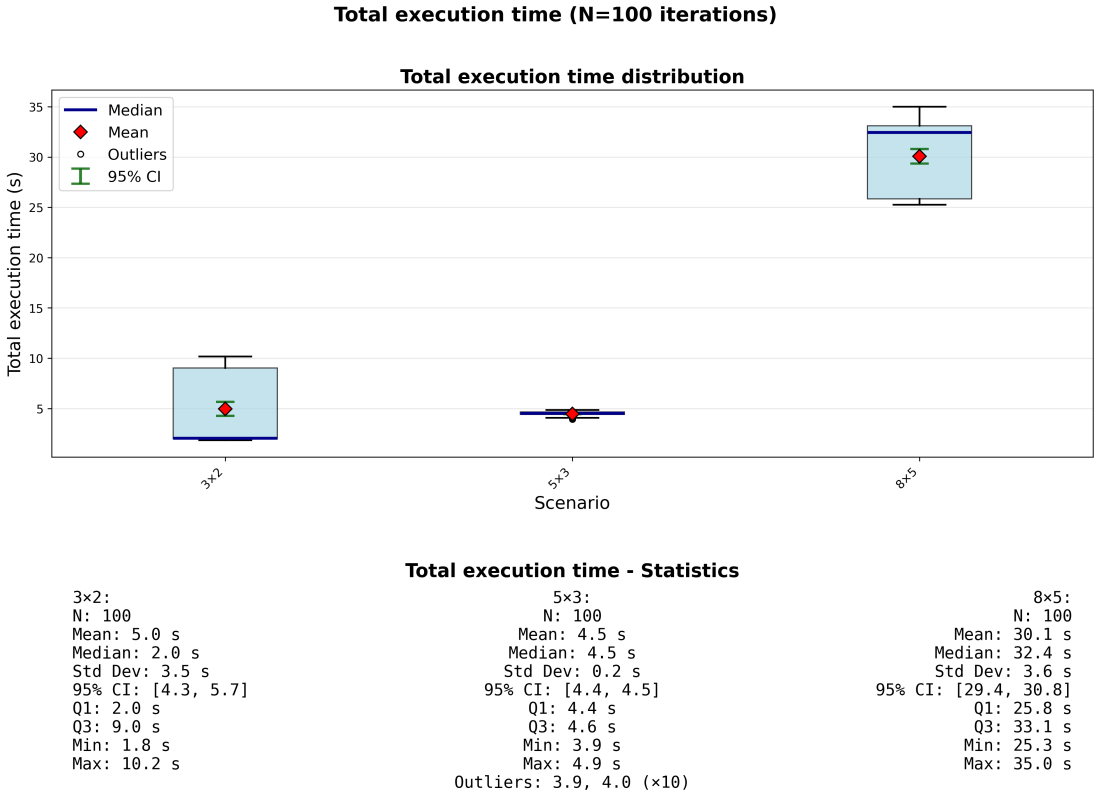


Figure 8.3: Total execution time distribution across KROWN benchmark scenarios, showing median, mean, quartiles, and 95% confidence intervals over 100 iterations.

Total execution time distributions demonstrate clear scaling patterns, with the small scenario achieving a median of 2.0 seconds, the medium scenario producing a median of 4.5 seconds, and the large scenario resulting in a median of 32.4 seconds.

Inversion time distributions exhibit greater complexity. The small scenario demonstrates highly variable performance with a median inversion time of 1.2 seconds but a substantially higher mean of 4.1 seconds (95% CI: [3.4, 4.8]), indicating that while typical executions complete rapidly, a subset of iterations requires significantly longer duration. The medium scenario shows consistent performance with median and mean both approximately 3.1 seconds (95% CI: [3.0, 3.1]). The large scenario produces a median inversion time of 25.3 seconds with mean 22.9 seconds (95% CI: [22.2, 23.6]), where the median exceeding the mean reflects distribution patterns influenced by SPARQL query execution variability.

Inversion overhead percentages measured relative to materialization time reveal non-linear scaling patterns. The small scenario exhibits median overhead of 145.7% (mean 510.8%, 95% CI: [422.8, 598.8]), where the substantial gap between median

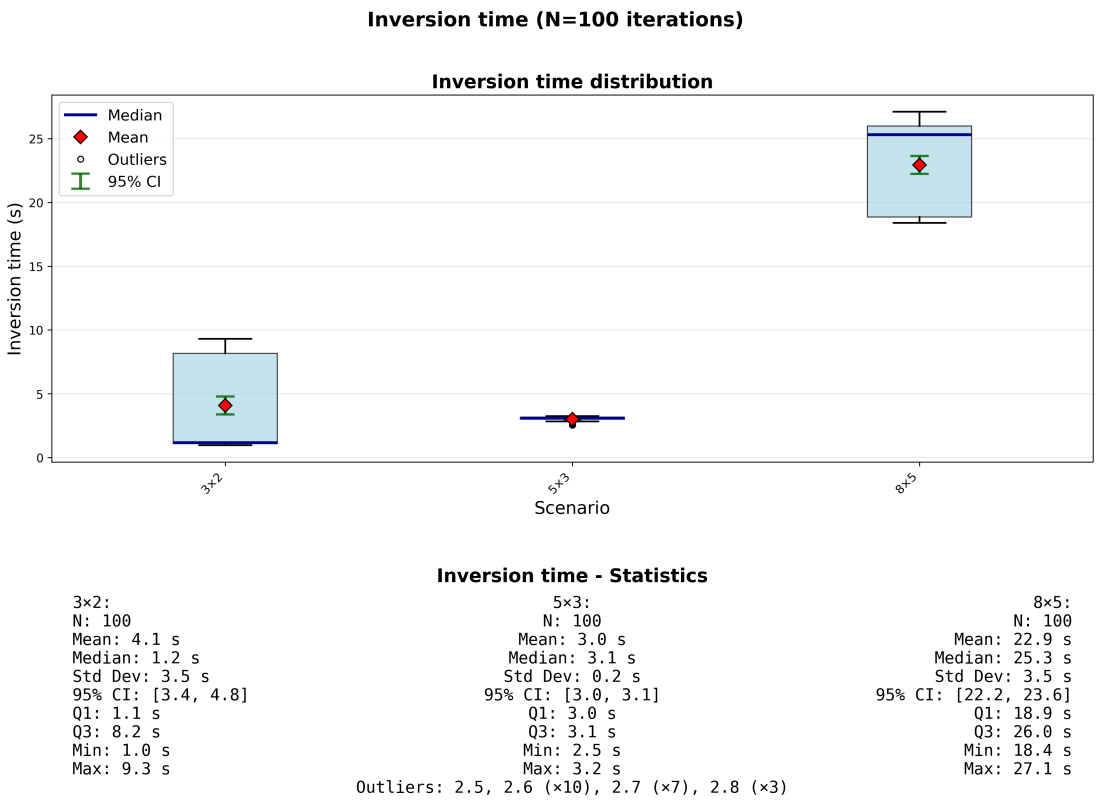


Figure 8.4: RML inversion time distribution across KROWN benchmark scenarios, showing median, mean, quartiles, and 95% confidence intervals over 100 iterations.

and mean reflects the highly variable execution pattern observed in absolute inversion times. The medium scenario produces median overhead of 273.4% (mean 267.0%, 95% CI: [263.8, 270.3]) with narrow confidence intervals indicating consistent relative performance. The large scenario demonstrates median overhead of 486.1% (mean 462.9%, 95% CI: [448.4, 477.4]). While median overhead values suggest relatively stable relative costs (145.7%, 273.4%, 486.1%), the underlying absolute inversion times scale from 1.2 seconds to 25.3 seconds (21× increase in median values), compared to materialization time scaling from 0.8 seconds to 4.9 seconds (6.1× increase in median values).

This disparity in absolute time scaling confirms that SPARQL query execution against the RDF graph constitutes the primary performance bottleneck. As the number of triples maps and predicate-object maps increases, the algorithm must execute more SPARQL queries to reconstruct the relational structure, with each query potentially traversing larger graphs. The performance variability observed in the small scenario, despite the database being emptied and reloaded for each iteration, suggests that execution time may depend on factors such as operating

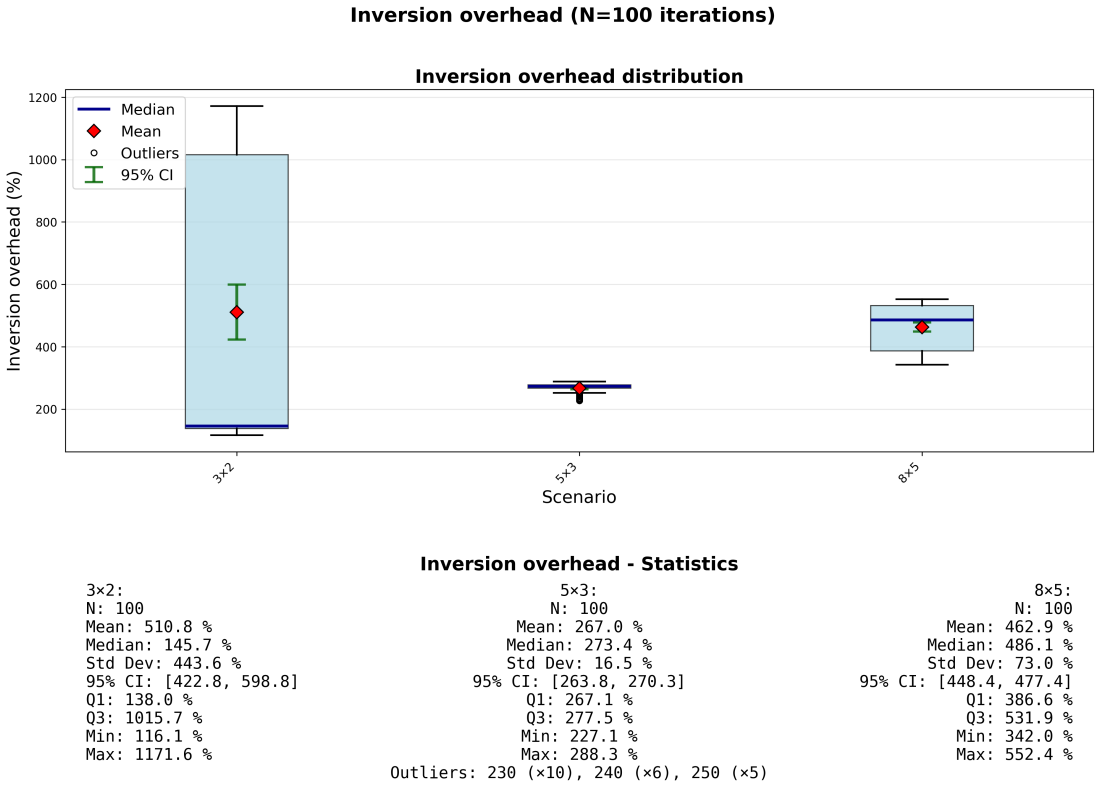


Figure 8.5: Inversion overhead percentage distribution across KROWN benchmark scenarios, showing median, mean, quartiles, and 95% confidence intervals over 100 iterations.

system scheduling and I/O variability.

The evaluation results demonstrate that this represents preliminary experimentation with evident limitations. The 37% success rate on conformance tests, combined with fundamental architectural constraints including the inability to process SQL queries as logical tables, indicates that substantial work remains before the algorithm can reliably handle the full spectrum of R2RML mapping scenarios. The non-linear growth in inversion overhead, with median values reaching 486.1% for graphs with 2 million triples, raises performance concerns for larger datasets. These limitations indicate that integration of this approach into HERITRACE would require addressing both correctness and efficiency challenges that extend beyond the scope of this preliminary investigation.

Chapter 9

Discussion and conclusion

The evaluation in Chapter 7 validated that HERITRACE addresses the research questions stated in Chapter 1. Regarding RQ1, end users without RDF expertise successfully completed metadata curation tasks at 67–100% success rates, demonstrating that domain experts can participate in semantic data curation through interfaces that abstract technical complexity while the system automatically maintains provenance documentation and change tracking. Regarding RQ2, technicians achieved 90% success on configuration tasks, confirming that initial system setup for domain-specific requirements remains feasible for technical staff. System usability scores indicated excellent usability for technicians and above-average usability for end users. However, the evaluation also revealed specific friction points and raised questions about long-term sustainability that warrant reflection.

One of the most frequently reported barriers among end users involved entity creation workflows where nested form structures caused context loss. When creating complex entities such as journal articles with authors, identifiers, and containment hierarchies (e.g., issue, volume, journal), users encountered dropdown sections that concealed previously entered information. Two distinct approaches could address this issue. A dialog-based approach would present sub-entity creation in modal dialogs that obscure the parent form, focusing user attention exclusively on the current entity being created. Alternatively, a fully expanded approach similar to Zenodo (European Organization For Nuclear Research & OpenAIRE, 2013) would display all fields simultaneously with required fields marked by asterisks.

However, the barrier most frequently reported by end users involved the absence of search functionality, with six of nine participants explicitly requesting this capability and experiencing disorientation when attempting exploratory discovery through browsing interfaces. This represents a deliberate design choice rather than an oversight. The problem of searching RDF datasets presents independent com-

plexity requiring specialized approaches. OSCAR (Heibi et al., 2019) addresses this domain through a configurable text-based search system that enables free-text queries, advanced multi-field searches with logical connectors, faceted filtering, and CSV export functionality. OSCAR operates as a client-side JavaScript application configured through JSON files that map user inputs to SPARQL queries using regular expressions, providing search capabilities without requiring end users to understand SPARQL syntax.

Visualizing RDF data presents similar independent complexity. HERITRACE provides basic entity visualization functionality, but customization of graphical presentation remains limited within the system. Specialized software for RDF visualization addresses requirements that extend beyond HERITRACE scope. LUCINDA (Heibi, 2025) provides detailed resource visualization through customizable HTML templates that present entity metadata in human-readable formats. The system enables relationship traversal and detailed property inspection through template-based rendering configured to match institutional display requirements.

The vision for HERITRACE positions the system within a workflow where specialized tools address distinct problems with independent complexity. OSCAR handles search and discovery. LUCINDA handles visualization and resource browsing. HERITRACE concentrates specifically on solving the editing problem: enabling domain experts to create, modify, merge, and restore semantic data while maintaining provenance documentation and change tracking. This separation would allow each tool to focus implementation effort on a single complex domain rather than attempting to cover all semantic data management activities within a monolithic system.

While end users encountered barriers in search and visualization, technicians faced different challenges. The file based configuration paradigm through SHACL and YAML proved effective for those with Semantic Web expertise. The evaluation demonstrated that pattern based learning enabled successful configuration even among participants lacking prior SHACL experience, who characterized the system as self-explanatory based on examination of existing examples. However, configuration errors triggered complete application failures requiring Docker container restarts, with error messages available only through log file inspection. A graphical configuration interface providing inline documentation, real-time validation, and comprehensible error feedback would reduce barriers for administrators. Whether such an interface can accommodate the full expressiveness of SHACL shape constraints without introducing its own complexity represents a research question.

The comparative analysis in Chapter 2 showed that no existing system simul-

taneously addresses user-friendliness, admin-friendliness, provenance management, change tracking, and direct RDF import capability. HERITRACE addresses all five requirements identified through the empirical analysis in Chapter 3.

However, this technical completeness introduces a new trade-off. While HERITRACE proves user-friendly for end users and admin-friendly for technicians familiar with W3C standards, it remains admin-friendly only for technical audiences. The file-based configuration approach creates a dependency on expertise that emerged explicitly during the evaluation when one technician questioned whether continuous technical support for configuration modifications represents a sustainable model for cultural heritage projects. The SUS score gap between technicians (83.8) and end users (78.9) quantitatively supports this concern: technical expertise still provides measurable advantages. Pattern-based learning functions effectively when extensive examples exist and when administrators possess computational thinking capabilities, but this approach may prove unavailable to humanities scholars and cultural heritage professionals who often serve as project administrators. Addressing the usability issues identified in the evaluation would narrow this gap and reduce dependency on technical support.

The deployment of HERITRACE in active projects provides evidence beyond prototype demonstration. The ParaText Bibliographical Database, described in Section 3.2, operates as a production system where philologists curate specialized bibliographic data about ancient Greek exegetical traditions. OpenCitations plans to adopt HERITRACE for enabling domain experts to contribute corrections and enhancements to citations and related metadata.

Several limitations constrain the scope and generalizability of this research. The system operates exclusively on RDF data, creating an adoption barrier for institutions maintaining collections in relational databases or other formats. Chapter 8 presented preliminary experimentation on inverting RML mappings to enable bidirectional transformation between relational data and RDF, but the 37% success rate on W3C R2RML conformance tests and non-linear performance overhead with median values reaching 486.1% for graphs with 2 million triples demonstrate that substantial work remains before this approach could enable production deployment.

The technician evaluation methodology presents a methodological limitation. Participants worked with partially configured HERITRACE instances where some entities possessed complete SHACL schemas and display rules while others lacked configuration elements. Creating configuration from scratch would require additional competencies that the partial configuration approach did not test. The 90% task completion rates and excellent usability scores therefore characterize config-

uration modification and extension tasks rather than initial system deployment. Whether administrators without existing configuration examples could successfully bootstrap HERITRACE configuration remains an open question.

These limitations point toward several directions for future research. The sustainability concerns raised by evaluation participants regarding configuration could be addressed through visual editors that guide administrators through SHACL shape creation with immediate feedback on constraint validity, paired with YAML configurators that preview interface changes before deployment. Such tools would need to preserve the full expressiveness of the underlying standards while hiding their syntactic complexity.

The architectural vision outlined above, where OSCAR handles search and LUCINDA handles visualization while HERITRACE focuses on editing, currently exists only as a conceptual separation. The three systems operate independently without integration. Future work would connect them, enabling direct transitions from search results or browsing to the editing interface. Search capabilities could also extend to provenance records, allowing queries like "show all entities modified last month" or "find changes made by a particular curator."

A more fundamental architectural extension concerns dataset scope. HERITRACE manages one triplestore at a time, but infrastructures like OpenCitations maintain interdependent datasets where bibliographic records in Meta link to citation assertions in Index. Modifying an entity in one dataset can affect referencing entities in another. Supporting such scenarios would require declaring dependencies between datasets and coordinating modifications through atomic transactions that span multiple triplestores.

Finally, the RML inversion approach described in Chapter 8 encountered three obstacles that future work would need to overcome: extending support beyond direct table references to handle SQL queries with joins as logical sources, enforcing unique identifiers in subject templates to prevent row collapse during round-trip conversion, and optimizing SPARQL reconstruction for graphs with millions of triples where current performance degrades.

References

Adamou, A., Brown, S., Barlow, H., Allocca, C., & d'Aquin, M. (2019, March). Crowdsourcing Linked Data on listening experiences through reuse and enhancement of library data. *International Journal on Digital Libraries*, *20*(1), 61–79. doi: 10.1007/s00799-018-0235-0

ADLab - Laboratorio Analogico Digitale, & /DH.arc - Digital Humanities Advanced Research Centre. (2022). *FICLIT Digital Library.* Retrieved 2025-10-29, from https://dl.ficlit.unibo.it/s/lib/page/home

Allocca, C., & Gougousis, A. (2015, July). A Preliminary Investigation of Reversing RML: From an RDF dataset to its Column-Based data source. *Biodiversity Data Journal*, *3*, e5464. doi: 10.3897/BDJ.3.e5464

Anderson, J. (2019, May). RDF Graph Stores as Convergent Datatypes. In *Companion Proceedings of The 2019 World Wide Web Conference* (pp. 940–942). San Francisco USA: ACM. doi: 10.1145/3308560.3316517

Arenas-Guerrero, J., Chaves-Fraga, D., Toledo, J., Pérez, M. S., & Corcho, O. (2024, January). Morph-KGC: Scalable knowledge graph materialization with mapping partitions. *Semantic Web*, *15*(1), 1–20. doi: 10.3233/SW-223135

Arndt, N., Naumann, P., Radtke, N., Martin, M., & Marx, E. (2019). Decentralized Collaborative Knowledge Management Using Git. *Journal of Web Semantics*, *54*, 29–47. doi: 10.1016/j.websem.2018.08.002

Asprino, L., Daga, E., Gangemi, A., & Mulholland, P. (2023). Knowledge graph construction with a façade: A unified method to access heterogeneous data sources on the web. *ACM Transactions on Internet Technology*, *23*(1), 1–31. doi: 10.1145/3555312

Bast, H., & Buchhold, B. (2017, November). QLever: A Query Engine for Efficient SPARQL+Text Search. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (pp. 647–656). Singapore Singapore: ACM. doi: 10.1145/3132847.3132921

Beck, K. (2003). *Test-driven development: By example.* Boston: Addison-Wesley.

Beckett, D. (2010, April). *RDF Syntaxes 2.0.* Retrieved 2021-07-22, from https://

www.w3.org/2009/12/rdf-ws/papers/ws11

Beek, W., Hoekstra, R., Maas, F., Meroño-Peñuela, A., & Leemans, I. (2014). Linking the STCN and Performing Big Data Queries in the Humanities. In *Digital Humanities Benelux Conference 2014*.

Ben-Kiki, O., Evans, C., & Ingerson, B. (2009). *Yaml ain't markup language (yaml™) version 1.1.* Retrieved 2025-10-29, from https://yaml.org/spec/1.1/

Berners-Lee, T. (2005, August). *Notation 3 Logic.* Retrieved 2021-07-23, from https://www.w3.org/DesignIssues/N3Logic

Berners-Lee, T., & Connolly, D. (2004). *Delta: An ontology for the distribution of differences between RDF graphs.* Retrieved from https://www.w3.org/DesignIssues/lncs04/Diff.pdf

Berthereau, D. (2024). *CSV Import.* Retrieved from swh:1:snp:1ae6e996991c5a188bb7f0bb2df6c8f7a14bf4a0

Bizer, C., Heath, T., & Berners-Lee, T. (2009, July). Linked Data - The Story So Far:. *International Journal on Semantic Web and Information Systems*, *5*(3), 1–22. doi: 10.4018/jswis.2009081901

Board, D. U. (2020). *DCMI Metadata Terms.* Retrieved 2021-07-16, from http://dublincore.org/specifications/dublin-core/dcmi-terms/2020-01-20/

Brandi, C. (2000). *Teoria del restauro* (Rist ed.) (Nos. N.S., 53). Torino: G. Einaudi.

Brase, J. (2009, November). DataCite - A Global Registration Agency for Research Data. In *2009 Fourth International Conference on Cooperation and Promotion of Information Resources in Science and Technology* (pp. 257–261). Beijing, China: IEEE. doi: 10.1109/COINFO.2009.66

Brooke, J. (1996, June). SUS: A 'Quick and Dirty' Usability Scale. In P. W. Jordan, B. Thomas, I. L. McClelland, & B. Weerdmeester (Eds.), *Usability Evaluation In Industry* (pp. 207–212). CRC Press. doi: 10.1201/9781498710411-35

Candela, G., Escobar, P., Carrasco, R. C., & Marco-Such, M. (2018, June). Migration of a library catalogue into RDA linked open data. *Semantic Web*, *9*(4), 481–491. doi: 10.3233/SW-170274

Caplan, P. (2017). *Understanding PREMIS.* Library of Congress Network Development and MARC Standards Office. Retrieved from https://www.loc.gov/standards/premis/understanding-premis-rev2017.pdf

Cappelli, F., Colavizza, G., & Peroni, S. (2025, December). Deep Learning Approaches to Author Name Disambiguation: A Survey. *International Journal*

on Digital Libraries, *26*(4), 21. doi: 10.1007/s00799-025-00428-6

Carriero, V. A., Gangemi, A., Mancinelli, M. L., Marinucci, L., Nuzzolese, A. G., Presutti, V., & Veninata, C. (2019). ArCo: The Italian Cultural Heritage Knowledge Graph. In C. Ghidini et al. (Eds.), *The Semantic Web – ISWC 2019* (Vol. 11779, pp. 36–52). Cham: Springer International Publishing. doi: 10.1007/978-3-030-30796-7_3

Carroll, J. J., Bizer, C., Hayes, P., & Stickler, P. (2005). Named graphs, provenance and trust. In *Proceedings of the 14th international conference on World Wide Web - WWW '05* (p. 613). Chiba, Japan: ACM Press. doi: 10.1145/1060745.1060835

Carroll, J. J., & Stickler, P. (2004). RDF triples in XML. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters - WWW Alt. '04* (p. 412). New York, NY, USA: ACM Press. doi: 10.1145/1013367.1013501

Cerdeira-Pena, A., De Bernardo, G., Fariña, A., Fernández, J. D., & Martínez-Prieto, M. A. (2024, January). Compressed and queryable self-indexes for RDF archives. *Knowledge and Information Systems*, *66*(1), 381–417. doi: 10.1007/s10115-023-01967-7

Cerdeira-Pena, A., Farina, A., Fernandez, J. D., & Martinez-Prieto, M. A. (2016, March). Self-Indexing RDF Archives. In *2016 Data Compression Conference (DCC)* (pp. 526–535). Snowbird, UT, USA: IEEE. doi: 10.1109/DCC.2016.40

Ciccarese, P., Wu, E., Wong, G., Ocana, M., Kinoshita, J., Ruttenberg, A., & Clark, T. (2008, October). The SWAN biomedical discourse ontology. *Journal of Biomedical Informatics*, *41*(5), 739–751. doi: 10.1016/j.jbi.2008.04.010

Cioffi, A., Coppini, S., Massari, A., Moretti, A., Peroni, S., Santini, C., & Shahidzadeh Asadi, N. (2022, June). Identifying and correcting invalid citations due to DOI errors in Crossref data. *Scientometrics*, *127*(6), 3593–3612. doi: 10.1007/s11192-022-04367-w

Corbin, J., & Strauss, A. (2008). *Basics of Qualitative Research (3rd ed.): Techniques and Procedures for Developing Grounded Theory.* 2455 Teller Road, Thousand Oaks California 91320 United States: SAGE Publications, Inc. doi: 10.4135/9781452230153

Corine, D., Neil, W., Luca, C., & Pierre-Yves, V. (2017, November). The British National Bibliography: Who Uses Our Linked Data? In *Proceedings of the International Conference on Dublin Core and Metadata Applications.* Dublin Core Metadata Initiative. doi: 10.23106/DCMI.952137546

da Silva, P. P., McGuinness, D. L., & Fikes, R. (2006, June). A proof markup

language for Semantic Web services. *Information Systems*, *31*(4-5), 381–395. doi: 10.1016/j.is.2005.02.003

da Silveira, L., da Silva, F. C. C., Zilio, S. C., & Cordeiro, L. S. (2020). Convergência de práticas Linked Open Data na Bibliothèque Nationale de France (BNF DATA). *Revista ACB: Biblioteconomia em Santa Catarina*, *25*(1), 21–40.

Daquino, M., & Peroni, S. (2019). *OCO, the OpenCitations Ontology.* Retrieved 2021-09-04, from https://w3id.org/oc/ontology/2019-09-19

Daquino, M., Peroni, S., Shotton, D., Colavizza, G., Ghavimi, B., Lauscher, A., ... Zumstein, P. (2020). The OpenCitations Data Model. In J. Z. Pan et al. (Eds.), *The Semantic Web – ISWC 2020* (Vol. 12507, pp. 447–463). Cham: Springer International Publishing. doi: 10.1007/978-3-030-62466-8_28

Daquino, M., Wigham, M., Daga, E., Giagnolini, L., & Tomasi, F. (2023, September). CLEF. A Linked Open Data Native System for Crowdsourcing. *Journal on Computing and Cultural Heritage*, *16*(3), 1–17. doi: 10.1145/3594721

Dibowski, H. (2024, December). Full Traceability and Provenance for Knowledge Graphs. In C. Trojahn, D. Porello, & P. P. F. Barcelos (Eds.), *Frontiers in Artificial Intelligence and Applications.* IOS Press. doi: 10.3233/FAIA241309

Diefenbach, D., Wilde, M. D., & Alipio, S. (2021). Wikibase as an Infrastructure for Knowledge Graphs: The EU Knowledge Graph. In A. Hotho et al. (Eds.), *The Semantic Web – ISWC 2021* (Vol. 12922, pp. 631–647). Cham: Springer International Publishing. doi: 10.1007/978-3-030-88361-4_37

Dimou, A., Sande, M. V., Colpaert, P., Verborgh, R., Mannens, E., & Walle, R. (2014). RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *Proceedings of the 7th Workshop on Linked Data on the Web.*

Dividino, R., Sizov, S., Staab, S., & Schueler, B. (2009, September). Querying for provenance, trust, uncertainty and other meta knowledge in RDF. *Journal of Web Semantics*, *7*(3), 204–219. doi: 10.1016/j.websem.2009.07.004

Dobriy, D., & Polleres, A. (2023, December). O2WB: A tool enabling ontology reuse in Wikibase. In *Proceedings of the 12th Knowledge Capture Conference 2023* (pp. 101–104). Pensacola FL USA: ACM. doi: 10.1145/3587259.3627568

Doerr, M. (2003). The CIDOC Conceptual Reference Module: : An Ontological Approach to Semantic Interoperability of Metadata. *AI Mag.*, *24*(3), 75–92. doi: 10.1609/aimag.v24i3.1720

European Organization For Nuclear Research, & OpenAIRE. (2013). *Zenodo.* CERN. doi: 10.25495/7GXK-RD71

Falco, R., Gangemi, A., Peroni, S., Shotton, D., & Vitali, F. (2014). Modelling OWL

Ontologies with Graffoo. In V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, & A. Tordai (Eds.), *The Semantic Web: ESWC 2014 Satellite Events* (Vol. 8798, pp. 320–325). Cham: Springer International Publishing. doi: 10.1007/978-3-319-11955-7_42

Fernández, J., Umbrich, J., Polleres, A., & Knuth, M. (2016). Evaluating Query and Storage Strategies for RDF Archives. In *Proceedings of the 12th International Conference on Semantic Systems.*

Fionda, V., Chekol, M. W., & Pirrò, G. (2016). Gize: A Time Warp in the Web of Data. In *SEMWEB.*

Flouris, G., Fundulaki, I., Pediaditis, P., Theoharis, Y., & Christophides, V. (2009). Coloring RDF Triples to Capture Provenance. In D. Hutchison et al. (Eds.), *The Semantic Web - ISWC 2009* (Vol. 5823, pp. 196–212). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-04930-9_13

Giacomini, S., Daquino, M., Tomasi, F., & Fintoni, L. A. (2025, January). CLEF 2.0. Solutions for Native Linked Data Cataloguing of Italian Digital Cultural Heritage. *JLIS.it*, *16*(1), 108–126. doi: 10.36253/jlis.it-611

Gil, Y., Cheney, J., Groth, P., Hartig, O., Miles, S., Moreau, L., & Silva, P. (2010). *Provenance XG Final Report* [W3C.]. Retrieved from http://www.w3.org/2005/Incubator/prov/XGR-prov-20101214/

Gosden, C., & Marshall, Y. (1999, October). The cultural biography of objects. *World Archaeology*, *31*(2), 169–178. doi: 10.1080/00438243.1999.9980439

Grandi, F. (2010). T-SPARQL: A TSQL2-like Temporal Query Language for RDF. In *ADBIS (local proceedings)* (pp. 21–30).

Graube, M., Hensel, S., & Urbas, L. (2016). Open semantic revision control with R43ples: Extending SPARQL to access revisions of named graphs. In *Proceedings of the 12th International Conference on Semantic Systems* (pp. 49–56).

Groth, P., Gibson, A., & Velterop, J. (2010, September). The anatomy of a nanopublication. *Information Services & Use*, *30*(1-2), 51–56. doi: 10.3233/ISU-2010-0613

Gschwend, A., & Lassila, O. (2022). *PROPOSED RDF-star Working Group Charter.* W3C. Retrieved 2022-08-14, from https://w3c.github.io/rdf-star-wg-charter/

Haak, L. L., Fenner, M., Paglione, L., Pentz, E., & Ratner, H. (2012, October). ORCID: A system to uniquely identify researchers. *Learned Publishing*, *25*(4), 259–264. doi: 10.1087/20120404

Hannemann, J., & Kett, J. (2010). Linked data for libraries. In *Proc of the world library and information congress of the Int'l Federation of Library Associations*

and Institutions (IFLA).

Hartig, O., & Thompson, B. (2019, March). Foundations of an Alternative Approach to Reification in RDF. *arXiv:1406.3399 [cs].* Retrieved 2021-10-07, from http://arxiv.org/abs/1406.3399

Heibi, I. (2025, October). *Opencitations/lucinda.* Zenodo. doi: 10.5281/ZENODO.17476884

Heibi, I., Peroni, S., & Shotton, D. (2019, November). Enabling text search on SPARQL endpoints through OSCAR. *Data Science*, *2*(1-2), 205–227. doi: 10.3233/DS-190016

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

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004, March). Design Science in Information Systems Research1. *MIS Quarterly*, *28*(1), 75–106. doi: 10.2307/25148625

Hilbold, I. (2018). Jules Marouzeau et « L'Année philologique »: Aux origines de la réforme de la bibliographie d'études classiques. *Revue des Études Latines*, *96*, 239–258.

Hoffart, J., Suchanek, F. M., Berberich, K., & Weikum, G. (2013, January). YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence*, *194*, 28–61. doi: 10.1016/j.artint.2012.06.001

Hyvönen, E. (2020, June). Linked Open Data Infrastructure for Digital Humanities in Finland. *Digital Humanities in the Nordic and Baltic Countries Publications*, *3*(1), 254–259. doi: 10.5617/dhnbpub.11195

Hyvönen, E. (2023, April). Digital humanities on the Semantic Web: Sampo model and portal series. *Semantic Web*, *14*(4), 729–744. doi: 10.3233/SW-223034

ICite, Hutchins, B. I., & Santangelo, G. (2022). *iCite Database Snapshots (NIH Open Citation Collection).* The NIH Figshare Archive. doi: 10.35092/YHJC.C.4586573

Im, D.-H., Lee, S.-W., & Kim, H.-J. (2012, February). A Version Management Framework for RDF Triple Stores. *International Journal of Software Engineering and Knowledge Engineering*, *22*(01), 85–106. doi: 10.1142/S0218194012500040

Immonen, V., Bonnie, R., Dixon, H., Tervahauta, U., & Thomas, S. (Eds.). (2020). *Working with cultural objects and manuscripts: Provenance, legality, and responsible stewardship* (No. 77). Helsinki: Suomen museoliitto ry.

International Council of Museums. (2017). *ICOM code of ethics for museums.* Paris:

International Council of Museums.

Isaac, A., & Haslhofer, B. (2013). Europeana Linked Open Data – data.europeana.eu. *Semantic Web*, *4*(3), 291–297. doi: 10.3233/SW-120092

Johnson, M. K., Hashtroudi, S., & Lindsay, D. S. (1993). Source monitoring. *Psychological Bulletin*, *114*(1), 3–28. doi: 10.1037/0033-2909.114.1.3

Jones, S. M., Klein, M., de Sompel, H. V., Nelson, M. L., & Weigle, M. C. (2021). Interoperability for Accessing Versions of Web Resources with the Memento Protocol. In D. Gomes, E. Demidova, J. Winters, & T. Risse (Eds.), *The Past Web: Exploring Web Archives* (pp. 101–126). Cham: Springer International Publishing. doi: 10.1007/978-3-030-63291-5_9

Kellogg, G., Hartig, O., Champin, P.-A., & Seaborne, A. (2025, December). *RDF 1.2 concepts and abstract data model* (W3C Working Draft). World Wide Web Consortium. Retrieved from https://www.w3.org/TR/rdf12-concepts/

Keskisärkkä, R., Blomqvist, E., Lind, L., & Hartig, O. (2019). RSP-QL* : Enabling Statement-Level Annotations in RDF Streams. In M. Acosta, P. Cudré-Mauroux, M. Maleshkova, T. Pellegrini, H. Sack, & Y. Sure-Vetter (Eds.), *Semantic Systems. The Power of AI and Knowledge Graphs* (Vol. 11702, pp. 140–155). Cham: Springer International Publishing. doi: 10.1007/978-3-030-33220-4_11

Kieffer, S., Sangiorgi, U. B., & Vanderdonckt, J. (2015, January). ECOVAL: A Framework for Increasing the Ecological Validity in Usability Testing. In *2015 48th Hawaii International Conference on System Sciences* (pp. 452–461). HI, USA: IEEE. doi: 10.1109/HICSS.2015.61

Knublauch, H., & Kontokostas, D. (2017, July). *Shapes Constraint Language (SHACL)* (W3C Recommendation). W3C. Retrieved from https://www.w3.org/TR/shacl/

Krech, D., Grimnes, G. A., Higgins, G., Car, N., Hees, J., Aucamp, I., … Stuart, V. (2025, September). *RDFLib*. doi: 10.5281/zenodo.6845245

Kremer, C. (2021, July). Bibliothèque nationale du Luxembourg : multiple et ouverte à tous. *Arabesques*, *102*, 26–27. doi: 10.35562/arabesques.2657

Krötzsch, M., Vrandečić, D., & Völkel, M. (2006). Semantic MediaWiki. In D. Hutchison et al. (Eds.), *The Semantic Web - ISWC 2006* (Vol. 4273, pp. 935–942). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/11926078_68

Kurkowska-Budzan, M., Soroko, E., & Stasiak, M. (2022, January). Structured Interview in Historical Research: a Description of Research Procedures. *Historyka Studia Metodologiczne*, 299–323. doi: 10.24425/hsm.2021.138890

Lampa, S., Willighagen, E., Kohonen, P., King, A., Vrandečić, D., Grafström, R., & Spjuth, O. (2017, December). RDFIO: Extending Semantic MediaWiki for interoperable biomedical data management. *Journal of Biomedical Semantics*, *8*(1), 35. doi: 10.1186/s13326-017-0136-y

La Niece, S. (2009). Forgeries and public collections. *ArchéoSciences*, *33*, 329–333. doi: 10.4000/archeosciences.2419

Laurence, C. M. (2013). Linked data and the library of congress. *Library Philosophy and Practice*, 2–24.

Lebo, T., Sahoo, S., & McGuinness, D. (2013, April). *PROV-O: The PROV Ontology* (Vol. 04 30). PROV-O. Retrieved 2021-07-16, from http://www.w3.org/TR/2013/REC-prov-o-20130430/

Lecy, J. D., & Beatty, K. E. (2012). Representative Literature Reviews Using Constrained Snowball Sampling and Citation Network Analysis. *SSRN Electronic Journal*. doi: 10.2139/ssrn.1992601

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., ... Bizer, C. (2015). DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, *6*(2), 167–195. doi: 10.3233/SW-140134

Lejeune, M. (2021, December). La flèche de l'ancienne abbatiale de Saint-Denis : Un bilan archéologique. *Bulletin du Centre d'études médiévales d'Auxerre*(25.2). doi: 10.4000/cem.18557

Lewis, J., & Sauro, J. (2017, August). Revisiting the factor structure of the system usability scale. *Journal of Usability Studies*, *12*(4), 183–192.

Lewis, J. R., & Sauro, J. (2009). The Factor Structure of the System Usability Scale. In M. Kurosu (Ed.), *Human Centered Design* (Vol. 5619, pp. 94–103). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-02806-9_12

Lindström, N., & Champin, P.-A. (2025, April). *RDF 1.2 primer* (W3C Group Draft Note). World Wide Web Consortium. Retrieved from https://www.w3.org/TR/rdf12-primer/

Liu, A. H., Ehrenberg, A., Lo, A., Denoix, C., Barreau, C., Lample, G., ... Tang, Y. (2025). *Voxtral*. arXiv. doi: 10.48550/ARXIV.2507.13264

Maltezou, H. C., La-Scola, B., Astra, H., Constantopoulou, I., Vlahou, V., Kafetzis, D. A., ... Raoult, D. (2004). Mycoplasma pneumoniae and Legionella pneumophila in community-acquired lower respiratory tract infections among hospitalized children: Diagnosis by real time PCR. *Scandinavian Journal of Infectious Diseases*, *36*(9), 639–642. doi: 10.1080/00365540410020884

Manghi, P., Manola, N., Horstmann, W., & Peters, D. (2010). An Infrastructure

for Managing EC Funded Research Output: The OpenAIRE Project. *Grey Journal (TGJ)*, *6*(1).

Manola, F., & Miller, E. (2004, February). *RDF Primer.* Retrieved 2021-07-22, from http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

Martis, C. (2013). L'enigma del PLouvre inv. 7733 verso : l'epigramma dell'ostrica. *Studi di egittologia e papirologia : 10, 2013*, 117–150. doi: 10.1400/213891

Massari, A. (2025a, October). *Arcangelo7/knowledge-graphs-inversion: V1.1.0.* Zenodo. doi: 10.5281/ZENODO.17448652

Massari, A. (2025b, October). *HERITRACE user testing results: Usability evaluation and qualitative analysis.* Zenodo. doi: 10.5281/ZENODO.17313842

Massari, A. (2025c, October). *Opencitations/heritrace: V2.8.0.* Zenodo. doi: 10.5281/ZENODO.17449716

Massari, A., Mariani, F., Heibi, I., Peroni, S., & Shotton, D. (2024, March). OpenCitations Meta. *Quantitative Science Studies*, *5*(1), 50–75. doi: 10.1162/qss_a_00292

Massari, A., & Peroni, S. (2022). *Performing live time-traversal queries via SPARQL on RDF datasets.* arXiv. doi: 10.48550/ARXIV.2210.02534

Massari, A., & Peroni, S. (2025a, July). HERITRACE: A User-Friendly Semantic Data Editor with Change Tracking and Provenance Management for Cultural Heritage Institutions. *Umanistica Digitale*, *9*(20), 317–340. doi: 10.6092/ISSN.2532-8816/21218

Massari, A., & Peroni, S. (2025b, July). *Opencitations/time-agnostic-library: 5.0.4.* Zenodo. doi: 10.5281/ZENODO.5172996

Massari, A., Peroni, S., Tomasi, F., & Heibi, I. (2025, July). Representing provenance and track changes of cultural heritage metadata in RDF: A survey of existing approaches. *Digital Scholarship in the Humanities*, fqaf076. doi: 10.1093/llc/fqaf076

May, R. (2009, October). Patrimoine(s) et Conservation-Restauration(s): Quelques réflexions pour une théorie globale. *CeROArt*(4). doi: 10.4000/ceroart.1235

Meinhardt, P., Knuth, M., & Sack, H. (2015). TailR: A Platform for Preserving History on the Web of Data. In *Proceedings of the 11th International Conference on Semantic Systems* (pp. 57–64). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/2814864.2814875

Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., ... den Bussche, J. V. (2011, June). The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems*, *27*(6), 743–756. doi: 10.1016/j.future.2010.07.005

Moretti, A., Soricetti, M., Heibi, I., Massari, A., Peroni, S., & Rizzetto, E. (2024, February). The Integration of the Japan Link Center's Bibliographic Data into OpenCitations. *Journal of Open Humanities Data*, *10*, 21. doi: 10.5334/johd.178

National Library of Finland. (2026). *Finna.fi: Search service for Finnish archives, libraries and museums.* Retrieved 2026-02-07, from https://www.finna.fi

Neumann, T., & Weikum, G. (2010). X-RDF-3X: Fast Querying, High Update Rates, and Consistency for RDF Databases. *Proceedings of the VLDB Endowment*, *3*, 256–263.

Nguyen, V., Bodenreider, O., & Sheth, A. (2014). Don't like RDF reification?: Making statements about statements using singleton property. In *Proceedings of the 23rd international conference on World wide web - WWW '14* (pp. 759–770). Seoul, Korea: ACM Press. doi: 10.1145/2566486.2567973

Nielsen, J. (1993). *Usability engineering.* Boston: Academic Press.

Nilsson, L., Faldella, G., Jacquet, J.-M., Storsaeter, J., Silfverdal, S.-A., & Ekholm, L. (2005). A fourth dose of DTPa-IPV vaccine given to 4-6 year old children in Italy and Sweden following primary vaccination at 3, 5 and 11-12 months of age. *Scandinavian Journal of Infectious Diseases*, *37*(3), 221–229. doi: 10.1080/00365540410020884

Nowack, B. (2024). *Arc2.* Retrieved from swh:1:snp:8e3f9ce5ccfadece9490aa3b43bd6479a02de2fe

Oldman, D., & Tanase, D. (2018). Reshaping the Knowledge Graph by Connecting Researchers, Data and Practices in ResearchSpace. In D. Vrandečić et al. (Eds.), *The Semantic Web – ISWC 2018* (Vol. 11137, pp. 325–340). Cham: Springer International Publishing. doi: 10.1007/978-3-030-00668-6_20

OpenCitations. (2025, June). *OpenCitations Meta CSV dataset of all bibliographic metadata.* Zenodo. doi: 10.5281/ZENODO.15625650

Pediaditis, P. (2008). *Querying and Updating RDF/S Named Graphs* (Doctoral dissertation, University of Crete, Heraklion, Greece). Retrieved from https://publications.ics.forth.gr/_publications/Pediaditis_master.pdf

Pediaditis, P., Flouris, G., Fundulaki, I., & Christophides, V. (2009). On Explicit Provenance Management in RDF/S Graphs. In *First Workshop on the Theory and Practice of Provenance.* San Francisco, CA, USA: USENIX. Retrieved from https://www.usenix.org/legacy/event/tapp09/tech/full_papers/pediaditis/pediaditis.pdf

Pelgrin, O., Galárraga, L., & Hose, K. (2021, October). Towards fully-fledged archiving for RDF datasets. *Semantic Web Journal*, *12*(6), 903–925. doi:

10.3233/SW-210434

Pelgrin, O., Taelman, R., Galárraga, L., & Hose, K. (2025). Expressive querying and scalable management of large RDF archives. *Semantic Web*. Retrieved from https://www.semantic-web-journal.net/content/expressive-querying-and-scalable-management-large-rdf-archives

Peroni, S., & Shotton, D. (2012, December). FaBiO and CiTO: Ontologies for describing bibliographic resources and citations. *Journal of Web Semantics*, *17*, 33–43. doi: 10.1016/j.websem.2012.08.001

Peroni, S., & Shotton, D. (2020, February). OpenCitations, an infrastructure organization for open scholarship. *Quantitative Science Studies*, *1*(1), 428–444. doi: 10.1162/qss_a_00023

Peroni, S., Shotton, D., & Vitali, F. (2012, September). Scholarly publishing and linked data: Describing roles, statuses, temporal and contextual extents. In *Proceedings of the 8th International Conference on Semantic Systems* (pp. 9–16). Graz Austria: ACM. doi: 10.1145/2362499.2362502

Peroni, S., Shotton, D., & Vitali, F. (2016). A Document-inspired Way for Tracking Changes of RDF Data. In L. Hollink, S. Darányi, A. Peñuela, & E. Kontopoulos (Eds.), *Detection, Representation and Management of Concept Drift in Linked Open Data* (pp. 26–33). Bologna: CEUR Workshop Proceedings. Retrieved from http://ceur-ws.org/Vol-1799/Drift-a-LOD2016_paper_4.pdf

Petz, G. (2023, August). Linked Open Data. Zukunftsweisende Strategien. *Bibliothek Forschung und Praxis*, *47*(2), 213–222. doi: 10.1515/bfp-2023-0006

Priem, J., Piwowar, H. A., & Orr, R. (2022). OpenAlex: A fully-open index of scholarly works, authors, venues, institutions, and concepts. *CoRR*, *abs/2205.01833*. doi: 10.48550/arXiv.2205.01833

Rachinger, J. (2008, April). The Austrian National Library: A New Orientation in the Shadows of a Long History. *Alexandria: The Journal of National and International Library and Information Issues*, *20*(1), 151–160. doi: 10.1177/0955749008020001 05

Ramanujam, S., Gupta, A., Khan, L., Seida, S., & Thuraisingham, B. (2009, September). R2D: A Bridge between the Semantic Web and Relational Visualization Tools. In *2009 IEEE International Conference on Semantic Computing* (pp. 303–311). Berkeley, CA, USA: IEEE. doi: 10.1109/ICSC.2009.29

Richards, M., & Ford, N. (2020). *Fundamentals of software architecture: An engineering approach* (First edition ed.). Beijing Boston Farnham Sebastopol Tokyo: O'Reilly. Retrieved from https://mrce.in/ebooks/

Software-Fundamentals%20of%20Software%20Architecture.pdf

Sahoo, S. S., Bodenreider, O., Hitzler, P., Sheth, A., & Thirunarayan, K. (2010). Provenance Context Entity (PaCE): Scalable Provenance Tracking for Scientific RDF Data. In D. Hutchison et al. (Eds.), *Scientific and Statistical Database Management* (Vol. 6187, pp. 461–470). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-13818-8_32

Sahoo, S. S., & Sheth, A. P. (2009, October). *Provenir Ontology: Towards a Framework for eScience Provenance Management.* Retrieved from https://corescholar.libraries.wright.edu/knoesis/80

Salarelli, A. (2016, November). Gestire piccole collezioni digitali con Omeka: l'esperienza di MoRE (A Museum of REfused and unrealised art projects). *Bibliothecae.it*, *Vol 5*, 177-200 Paginazione. doi: 10.6092/ISSN.2283-9364/6393

Sande, M., Colpaert, P., Verborgh, R., Coppens, S., Mannens, E., & Walle, R. (2013). R&Wbase: Git for triples. In *Proceedings of the 6th Workshop on Linked Data on the Web. 996. CEUR Workshop Proceedings.* CEUR Workshop Proceedings.

Sanfilippo, S. (2009). *Redis.* Retrieved from https://redis.io

Sauro, J. (2016). *Quantifying the user experience: Practical statistics for user research.* Morgan Kaufmann.

Sauro, J., & Lewis, J. R. (2010, April). Average task times in usability tests: What to report? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 2347–2350). Atlanta Georgia USA: ACM. doi: 10.1145/1753326.1753679

Sikos, L. F., & Philp, D. (2020, September). Provenance-Aware Knowledge Representation: A Survey of Data Models and Contextualized Knowledge Graphs. *Data Science and Engineering*, *5*(3), 293–316. doi: 10.1007/s41019-020-00118-0

Stefanova, S., & Risch, T. (2013, June). Scalable reconstruction of RDF-archived relational databases. In *Proceedings of the Fifth Workshop on Semantic Web Information Management* (pp. 1–4). New York New York: ACM. doi: 10.1145/2484712.2484717

Suchanek, F. M., Alam, M., Bonald, T., Chen, L., Paris, P.-H., & Soria, J. (2024, July). YAGO 4.5: A Large and Clean Knowledge Base with a Rich Taxonomy. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 131–140). Washington DC USA: ACM. doi: 10.1145/3626772.3657876

Suchanek, F. M., Lajus, J., Boschin, A., & Weikum, G. (2019). Knowledge Repre-

sentation and Rule Mining in Entity-Centric Knowledge Bases. In M. Krötzsch & D. Stepanova (Eds.), *Reasoning Web. Explainable Artificial Intelligence* (Vol. 11810, pp. 110–152). Cham: Springer International Publishing. doi: 10.1007/978-3-030-31423-1_4

Suominen, O., Pessala, S., Tuominen, J., Lappalainen, M., Nykyri, S., Ylikotila, H., ... Hyvönen, E. (2015). Deploying National Ontology Services: From ONKI to Finto. In *Proceedings of the International Conference on Dublin Core and Metadata Applications* (Vol. 1383). Retrieved from https://ceur-ws.org/Vol-1383/paper6.pdf

Taelman, R., Sande, M., & Verborgh, R. (2018). OSTRICH: Versioned Random-Access Triple Store. In *Companion Proceedings of the Web Conference 2018* (pp. 127–130). Retrieved from https://core.ac.uk/download/pdf/157574975.pdf

Tamper, M. (2023). *From Text to Knowledge: Methods, Tools, and Applications for Digital Humanities Based on Linked Data* (Unpublished doctoral dissertation). Aalto University.

Tappolet, J., & Bernstein, A. (2009). Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In L. Aroyo et al. (Eds.), *The Semantic Web: Research and Applications* (Vol. 5554, pp. 308–322). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-02121-3_25

Tukey, J. W. (1977). *Exploratory data analysis*. Reading, MA: Addison-Wesley.

Udrea, O., Recupero, D. R., & Subrahmanian, V. S. (2010, January). Annotated RDF. *ACM Transactions on Computational Logic*, *11*(2), 1–41. doi: 10.1145/1656242.1656245

Van Assche, D., Chaves-Fraga, D., & Dimou, A. (2025). KROWN: A Benchmark for RDF Graph Materialisation. In G. Demartini et al. (Eds.), *The Semantic Web – ISWC 2024* (Vol. 15233, pp. 20–39). Cham: Springer Nature Switzerland. doi: 10.1007/978-3-031-77847-6_2

Villazón-Terrazas, B., & Hausenblas, M. (2012, August). *R2RML and Direct Mapping Test Cases* (Tech. Rep.). W3C. Retrieved from http://www.w3.org/TR/2012/NOTE-rdb2rdf-test-cases-20120814/

Vitali, F., & Pasqual, V. (2025, May). The Representation of Historical Uncertainties as the Outcome of Competing and Incompatible Certainties. In *Models of Data Extraction and Architecture in Relational Databases of Early Modern Private Political Archives* (p. Chapter_18903). Venice: Fondazione Università Ca' Foscari. doi: 10.30687/978-88-6969-919-1/003

Völkel, M., Winkler, W., Sure, Y., Kruk, S., & Synak, M. (2005). SemVersion: A

Versioning System for RDF and Ontologies. In *Proc. of ESWC*.

Vrandečić, D., & Krötzsch, M. (2014, September). Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, *57*(10), 78–85. doi: 10.1145/2629489

W3C. (2006, December). *Defining N-ary Relations on the Semantic Web.* Retrieved 2021-07-22, from http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/

W3C. (2012a). *OWL 2 Web Ontology Language Document Overview (Second Edition).* Retrieved 2025-10-20, from https://www.w3.org/TR/owl2-overview/

W3C. (2012b). *R2RML: RDB to RDF Mapping Language.* Retrieved from https://www.w3.org/TR/r2rml/

W3C. (2013a). *Dublin Core to PROV Mapping.* Retrieved from https://www.w3.org/TR/prov-dc/

W3C. (2013b, April). *PROV-DM: The PROV Data Model.* Retrieved 2021-07-16, from http://www.w3.org/TR/2013/REC-prov-dm-20130430/

W3C. (2013c). *SPARQL 1.1 Query Language.* Retrieved 2025-10-20, from https://www.w3.org/TR/sparql11-query/

W3C. (2014). *RDF Schema 1.1.* Retrieved from https://www.w3.org/TR/rdf-schema/

W3C. (2024a). *RDF 1.1 N-Quads.* Retrieved from https://www.w3.org/TR/n-quads/

W3C. (2024b). *RDF 1.2 TriG.* Retrieved from https://www.w3.org/TR/rdf12-trig/

WikiTeq. (2023). *Semantic Watchlist.* Retrieved from swh:1:snp:5e269f8cdf2b31b21ab34495f89cdd741418bf3f

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., … Mons, B. (2016, March). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, *3*, 1–9. doi: 10.1038/sdata.2016.18

Wohlin, C. (2014, May). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (pp. 1–10). London England United Kingdom: ACM. doi: 10.1145/2601248.2601268

Wulff, E. (2024, July). Digital Humanities: The Case Study of the National Library in Spain. In M. Khosrow-Pour, D.B.A. (Ed.), *Advances in Information Quality*

and Management (pp. 1–19). IGI Global. doi: 10.4018/978-1-6684-7366-5.ch052

Zaniolo, C., Gao, S., Atzori, M., Chen, M., & Gu, J. (2018, April). User-friendly temporal queries on historical knowledge bases. *Information and Computation*, *259*, 444–459. doi: 10.1016/j.ic.2017.08.012

Zimmermann, A., Lopes, N., Polleres, A., & Straccia, U. (2012, March). A general framework for representing, reasoning and querying with annotated Semantic Web data. *Journal of Web Semantics*, *11*, 72–95. doi: 10.1016/j.websem.2011.08.006