A Multi-Tenant OAI-PMH 2.0 Endpoint with Type-Constrained Resource Description Schema for Federated SSH Research Infrastructures

Pietro Sichera[1][0000−0003−2157−7717], Cristina Marras[1][0000−0001−5927−897X], and Enrico Pasini[2][0000−0002−4525−187X]

[1] Istituto per il Lessico Intellettuale Europeo e Storia delle Idee (ILIESI), National Research Council of Italy (CNR), Rome, Italy
{pietro.sichera,cristina.marras}@cnr.it
[2] University of Turin, Italy
enrico.pasini@unito.it

Abstract. We present the design, implementation, and evaluation of a multi-tenant OAI-PMH 2.0 endpoint serving federated metadata for the Italian nodes of four European research infrastructures in the Social Sciences and Humanities (SSH): OPERAS, E-RIHS, CLARIN, and DARIAH. The system addresses three challenges in federated digital library construction: the need to expose metadata from multiple infrastructures through a single, validator-compliant endpoint while preserving infrastructure-specific semantics; the design of a resource description schema that supports eight resource types with type-constrained property assignments; and the deployment of such a system using exclusively free-tier and open-source components, suitable for resource-constrained research institutions. The implementation comprises five loosely coupled components (a web-based data entry application, a backend validation API, two Git-based repositories for data and configuration, and the OAI-PMH server itself) and exposes both Dublin Core and a custom H2IOSC native metadata format. We describe the four-stage data lifecycle that transforms user input into protocol-compliant XML, the type-property constraint matrix that enforces semantic validity, and the security architecture based on hashed API keys and least-privilege access tokens. The endpoint has been validated against the official OpenArchives validator with 38 tests passed, 0 warnings, and 0 errors, and is in production use with a status of "COMPLIANT". We report performance measurements, discuss the system's alignment with FAIR principles and EOSC interoperability requirements, and reflect on the implications of the design choices for other federated digital library projects.

Keywords: OAI-PMH · Digital libraries · Metadata harvesting · Resource description schema · FAIR · Federated research infrastructure · Social Sciences and Humanities · Multi-tenant architecture · Dublin Core · H2IOSC · OPERAS.

1 Introduction

Federated research infrastructures in the Social Sciences and Humanities (SSH) face a recurring problem in metadata management: enabling cross-infrastructure resource discovery without imposing a single, uniform metadata schema or operational model. Each participating infrastructure typically maintains its own resource catalogues, descriptive vocabularies, and curatorial practices, reflecting the disciplinary specificities of its community. Yet researchers and aggregators increasingly need unified entry points through which resources from multiple infrastructures can be discovered, harvested, and integrated into broader catalogues.

This paper presents the design, implementation, and evaluation of a system that addresses this challenge in the context of the H2IOSC[3] project (Humanities and Heritage Italian Open Science Cloud), an initiative funded under the Italian National Recovery and Resilience Plan (PNRR) that federates the Italian nodes of four ESFRI research infrastructures: OPERAS for open scholarly communication, E-RIHS for heritage science, CLARIN for language resources, and DARIAH for arts and humanities. The system implements a multi-tenant OAI-PMH 2.0 endpoint that serves metadata from all participating infrastructures through infrastructure-specific paths, while sharing a common server, data model, and operational infrastructure.

Problem Statement. Three concrete problems motivated the design of the system. First, the H2IOSC infrastructures have heterogeneous data practices but must contribute to a unified central catalogue (the H2IOSC Marketplace) that aggregates and presents their resources. A naive approach in which each infrastructure independently exposes its own OAI-PMH endpoint would multiply operational costs and fragment the harvesting configuration. Second, the resource description schema must support a range of resource types – not only datasets and publications, as in many digital library systems, but also software tools, services, projects, terminology resources, learning resources, and executable workflows – each with its own characteristic properties. Third, the system must be sustainable beyond the project funding period without requiring dedicated infrastructure investments.

Contribution. This paper presents several interrelated contributions to the field of digital library infrastructure for Social Sciences and Humanities (SSH) research. First, we describe the architecture of a multi-tenant OAI-PMH 2.0 server capable of exposing multiple infrastructure-specific endpoints from a single deployment; the underlying configuration-driven model minimizes the effort required to onboard new infrastructures, reducing it to a single dictionary entry.

Second, we introduce a resource description schema encompassing eight resource types, multilingual fields, controlled vocabulary integration, and structured identity management through role-based actor associations. The schema is further governed by a type-property constraint matrix that formally defines

[3] Official website https://www.h2iosc.cnr.it/

which properties are valid for each resource type, ensuring semantic consistency across records.

A third contribution concerns the four-stage data lifecycle implemented in the system, which traces metadata from HTML form input through normalized JSON storage to fully denormalized OAI-PMH XML output. The pipeline supports two serialization formats - Dublin Core, for broad interoperability, and H2IOSC native, for infrastructure-specific richness - thus balancing standardization with expressiveness. Fourth, the paper presents a deployment architecture built exclusively on free-tier and open-source components, including Cloudflare Pages, Render, GitHub, and Python Flask. This design choice demonstrates that production-grade digital library services for SSH research can be realized at negligible operational cost, lowering the barrier to adoption for institutions with limited resources.[4]

Finally, the paper reports on the validation and performance evaluation of the deployed system, encompassing conformance results obtained from the official OpenArchives validator as well as response time measurements collected under representative harvesting workloads. At the time of writing, the system is operational and exposes 4 records for OPERAS-IT and 15 records for E-RIHS.it, while integration with CLARIN is currently in progress.

2 Related Work

OAI-PMH and Digital Libraries. The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) version 2.0, specified by Lagoze et al. [1], remains the most widely adopted standard for metadata exchange in digital libraries. The protocol defines six verbs (Identify, ListMetadataFormats, ListSets, ListIdentifiers, ListRecords, GetRecord) and a request/response model based on HTTP and XML. Its longevity is due in part to the simplicity of the data provider role – exposing metadata in standardized formats – and to the existence of an established validator infrastructure that ensures interoperability between independently developed providers and harvesters [2].

OAI-PMH has been adopted by major aggregators including OpenAIRE [3], BASE, and DOAJ, and remains the primary harvesting protocol for institutional repositories built on platforms such as DSpace and EPrints. Recent work has explored extensions to OAI-PMH for richer formats (e.g., DataCite metadata for research data) and complementary protocols such as ResourceSync [4], but the core OAI-PMH model continues to serve as the lowest common denominator for metadata exchange.

Multi-Tenant Endpoint Architectures. Most OAI-PMH server implementations adopt a single-repository model: one server instance serves one repository. Multi-tenant deployments, in which a single server exposes multiple logical

[4] "Negligible" denotes the absence of licensing and hosting fees within the providers' free tiers, which impose rate and quota limits (e.g., request throttling, instance spin-down) adequate for the current scale but requiring a paid tier for high-traffic or very large deployments.

repositories, are less common but have been explored in the context of repository hosting services (e.g., DSpace-based hosted services that serve multiple institutions from a single deployment). Established platforms such as EPrints and DSpace can also approximate this: a single EPrints installation can host multiple archives, each with its own `Identify` response, sets, and formats. Our contribution is therefore not multi-tenancy alone, but its combination with a type-constrained schema and a database-free, free-tier architecture assembled entirely from managed components – a minimal-footprint design suited to small SSH federations rather than a full repository platform. The novelty of the H2IOSC implementation lies in combining multi-tenancy with strict logical separation: each tenant repository has its own Identify response, base URL, and record set, and external harvesters interact with each as an independent OAI-PMH provider.

Metadata for SSH Research Infrastructures. The metadata practices of SSH research infrastructures have been shaped by several initiatives. The CLARIN infrastructure developed the CMDI (Component Metadata Infrastructure) framework [5], which provides a flexible, schema-based approach to describing language resources. DARIAH has historically used a combination of CIDOC-CRM and Dublin Core for cultural heritage metadata. The SSH Open Marketplace, developed within the SSHOC project, introduced a unified resource description schema covering tools, datasets, training materials, and workflows, with curation provided by domain experts.

The H2IOSC schema described in this paper builds on these precedents but introduces several distinctive features: explicit support for executable workflows as a first-class resource type with its own properties (manifest, workflow_script, life_cycle_status); type-property constraints that enforce schema validity at the data entry layer; and structured identity management with role-based actor associations, supporting the integration with persistent identifier systems (ORCID, ROR).

Positioning of This Work. Cross-catalogue interoperability in the European landscape has been addressed at multiple levels: the SSHOC Open Marketplace adopts a curated approach; OpenAIRE [3] provides large-scale OAI-PMH harvesting. The H2IOSC system is complementary: its endpoints can be harvested by any of them, contributing Italian SSH descriptions to the European ecosystem.

The literature on OAI-PMH implementations is rich, but it is dominated by large-scale repository systems (DSpace, EPrints, Fedora) designed for substantial institutional deployments. Less attention has been paid to lightweight, multi-tenant implementations suitable for small federations of resource-constrained research nodes – a configuration that is increasingly relevant as smaller national initiatives and disciplinary federations seek to expose their resources at the European scale. Our contribution targets this gap.

3 System Architecture

The system follows a five-component architecture, with each component deployed independently and communicating through standard web protocols and the GitHub API (Fig. 1).

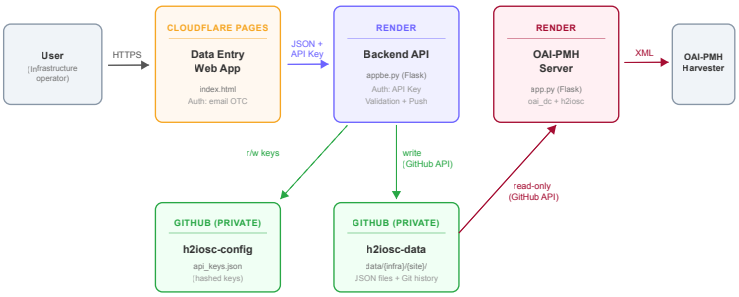

Fig. 1. System architecture and data flow. The five main components – Data Entry Web App, Backend API, Configuration Repository, Data Repository, and OAI-PMH Server – communicate through standard web protocols and the GitHub Contents API. Solid arrows indicate the primary data flow from user input to OAI-PMH harvesting; the dashed arrow indicates API key verification. Each connection uses a dedicated fine-grained access token with the minimum required privileges.

3.1 Component Overview

Data Entry Web Application: a single-page HTML application served through Cloudflare Pages, providing a tabbed interface for entering resource metadata organized according to the H2IOSC data model (items, properties, media, identities, actors, contacts, sources, relations). Authentication is provided by Cloudflare Access through email-based one-time-code verification.

Backend API: a Python Flask application deployed on Render, receiving submissions from the data entry application, validating them against the schema, and writing the validated JSON to the data repository through the GitHub Contents API. Authentication uses API keys transmitted in the X-API-Key header, stored in hashed form in a separate configuration repository.

Configuration Repository (h2iosc-config): a private GitHub repository containing the API keys file (in hashed form) and configuration metadata. Access to this repository is shared with infrastructure operators to enable distributed key generation through a command-line tool.

Data Repository (h2iosc-data): a private GitHub repository serving as the persistent storage layer. Its directory structure (`data/{infrastructure}/{site}/{timestamp}.json`) reflects the organizational hierarchy of the project. Each modification is recorded as a Git commit, providing a complete audit trail.

OAI-PMH Server: a separate Python Flask application deployed on a Render instance independent from the backend API. It reads data from the data repository through the GitHub Contents API using a read-only token, processes

the JSON files, and serves responses conforming to the OAI-PMH 2.0 protocol in two metadata formats.

3.2 Multi-Tenant Routing

The OAI-PMH server operates as a multi-tenant endpoint. Each infrastructure is accessible at a dedicated URL path, and the root path (/) serves a portal page listing all active infrastructures with their record counts. Adding a new infrastructure requires only an entry in a configuration dictionary in the server code:

```
INFRASTRUCTURES = {
    'operas': { 'display': 'OPERAS', 'full': 'OPERAS
        Infrastructure', 'github_folder': 'data/operas' },
    'erihs':  { 'display': 'E-RIHS', 'full': 'E-RIHS
        Infrastructure', 'github_folder': 'data/erihs' },
    # Adding a new infrastructure: one dictionary entry
        suffices
}
```

Each tenant endpoint behaves as an independent OAI-PMH repository with its own Identify response, record set, and base URL. From the perspective of an external harvester, the multi-tenant nature of the deployment is transparent: each endpoint is indistinguishable from a stand-alone OAI-PMH server.

The server implements OAI-PMH flow control through a `resumptionToken` (stateless, base64url-encoded cursor, default page size 100), so responses are bounded regardless of collection size, and exposes a set hierarchy on two dimensions derived from the records – resource type (`type:tool`, ...) and site (`site:roma`, ...) – enabling selective harvesting via the `set` argument.

3.3 Design Choices: Decoupling and Free-Tier Deployment

A key architectural choice is the decoupling of the read and write paths. The Backend API has write access to the data repository (through a fine-grained token scoped to write permissions on h2iosc-data), but the OAI-PMH server has only read access (through a separate token scoped to read-only). This separation has two benefits: it enforces the principle of least privilege at the credential level, ensuring that a compromise of the OAI-PMH server cannot result in data modification; and it allows the two services to scale and fail independently, since read-heavy harvesting traffic does not affect the write path used by infrastructure operators.

The complete system is deployed using exclusively free-tier services: Cloudflare Pages for the data entry web application (with built-in authentication via Cloudflare Access); Render for both the Backend API and the OAI-PMH server; GitHub for both the data repository and the configuration repository, including its REST API for programmatic access. The total monthly operational cost of the production deployment is effectively zero within the providers' free-tier

limits. This design choice is not incidental: it directly addresses the sustainability constraints faced by SSH research infrastructures, which typically operate without dedicated cloud infrastructure budgets.

4 Resource Description Schema

The H2IOSC resource description schema is the conceptual core of the system, defining what kinds of resources can be described, how they are structured, and what semantic constraints apply.

4.1 Design Principles

The schema was designed according to four guiding principles:

Multi-type resource support. The schema defines eight resource types, each representing a distinct category of digital resource relevant to SSH research: tool (software tools and applications), dataset (research data collections), publication (scholarly outputs), project (research initiatives), service (continuously operated services), terminology_resource (controlled vocabularies, thesauri, ontologies), learning_resource (training materials, courses, tutorials), and workflow (executable research workflows orchestrating multiple tools).

Multilingual by default. All user-facing textual fields (names, descriptions, property values, identity labels, contact labels) support parallel values in Italian and English, in alignment with H2IOSC's national-European positioning. The schema is designed to allow extension to additional languages without structural changes.

Separation of items and metadata. The schema separates items (resource descriptions) from associated metadata of different types: properties, media attachments, identities (persons and organizations), actors (role-based associations between identities and items), contacts, sources (external identifiers), and relations (typed links between items). This normalized structure facilitates editing and avoids data duplication.

Controlled vocabulary integration. Properties representing categorical values reference external controlled vocabularies, including TaDiRAH for activities, ISO 639-3 for languages, and the SSHOC vocabularies for keywords and resource categories. This grounds the H2IOSC catalogue in established semantic frameworks and supports cross-catalogue interoperability.

4.2 Entity Structure

The schema comprises eight entity types organized around the central Item entity:

Item is the primary entity. Required fields: `id`, `name` (multilingual), `type` (constrained to the eight allowed values), `uri` (unique identifier), `publicationState` (draft or published). Optional fields include multilingual descriptions, version, validation date (for workflows), `visible` and

`permissionRequired` booleans (controlling Marketplace visibility and access restrictions), and `documentationUri`. **Property** associates a typed metadata value with an item. Each property has a `propertyCode` from a controlled set and a multilingual value. Properties are organized into five sections: Categorisation (activity, keyword, discipline, language, mode_of_use, object_format, extent, intended_audience, standard, resource_category), Context (website, repository_url, usermanual_url, helpdesk_url, service_level_url), Access (license, terms_of_use, access_policy, geographical availability, privacy policy), Bibliographic (publication_type, publisher, year, journal, conference, volume, issue, pages, publication_place), and Technical (life_cycle_status, technical_readiness_level, societal_readiness_level, organization_readiness_level, manifest, workflow_script). **Media** represents multimedia attachments with fields for name (multilingual), MIME type, external URL, and a flag indicating whether the media is the cover image. **Identity** represents a person, organization, or infrastructure. Required fields: `id`, `type` (person/organization/infrastructure), `label` (multilingual). Optional `externalId` field accepts ORCID, ROR, or similar persistent identifiers. **Actor** creates a role-based association between an Identity and an Item. The `role` field is constrained to: creator, reviewer, curator, contributor, author, provider, contact. **Contact** associates communication information (email, URL, phone) with an Identity. **Source** associates an external identifier with an Identity, supporting multiple identifier systems (ORCID, Wikidata, DBLP, LinkedIn, GitHub, ResearchGate, Academia.edu, ResearcherID). **Relation** creates a typed directional link between two Items, with relation types including `relates_to`, `mentions`, `documents`, `extends`, and their inverses.

4.3 Type-Property Constraint Matrix

A distinctive feature of the schema is the type-property constraint matrix, which defines which property codes are recommended (and which are restricted) for each item type. Figure 2 shows an excerpt of the matrix for the Categorisation section.

The matrix serves two purposes. First, it acts as a data entry guide: the web application highlights recommended properties based on the selected item type, helping infrastructure operators populate descriptions consistently. Second, it acts as a validation constraint: properties that are not recommended for a given type are flagged during ingestion. The matrix reflects disciplinary practice: bibliographic properties are restricted to publications and datasets; technical readiness levels are restricted to tools; workflow-specific properties (manifest, workflow_script) are restricted to workflows. Formally, let T be the set of eight resource types and P the set of 35 property codes. The constraint model is a relation $R \subseteq T \times P$, where $(t, p) \in R$ iff property p is admissible for type t. A description of type t is *schema-valid* iff every asserted property p satisfies $(t, p) \in R$. The relation is materialized as the matrix of Fig. 2 and enforced both at data entry (only $\{p : (t, p) \in R\}$ is offered for type t) and at ingestion (submissions asserting $(t, p) \notin R$ are rejected), making type-conditioned

Property code	Tool	Dataset	Publica-tion	Project	Service	Termino-logy res.	Learning resource	Work-flow
Categorisation								
activity	●	●	●	●	●	●	●	●
keyword	●	●	●	●	●	●	●	●
discipline	●	●	●	●	●	●	●	●
language	●	●	●	●	●	●	●	●
mode_of_use	●							
object_format		●	●			●	●	
extent		●	●			●	●	
intended_audience					●	●	●	
standard						●		●
resource_category	●	●	●	●	●	●		●

Fig. 2. Excerpt from the type-property constraint matrix (Categorisation section). Red dots indicate properties recommended for each item type. Some properties (e.g., activity, keyword, discipline, language, resource_category) apply to all types; others (e.g., mode_of_use) are restricted to specific types. The complete matrix includes five sections (Categorisation, Context, Access, Bibliographic, Technical) covering 35 property codes.

admissibility a checkable property of every record rather than a curatorial convention. The matrix is implemented as a data structure in the web application's JavaScript code and is also enforced server-side by the validation logic in the Backend API; together, these two enforcement points provide defense in depth against malformed submissions.

Identifiers and Persistent References.

Each Item has a globally unique id constructed by concatenating infrastructure, site, and a local identifier (e.g., `operas-roma-item-001`). The OAI-PMH identifier follows the pattern `oai:h2iosc:{infrastructure}:{site}:{local_id}` (e.g. `oai:h2iosc:erihs:firenze01:erihs-firenze01-item-012-Prototyper`), ensuring globally unique, structured identifiers compatible with OAI-PMH conventions and amenable to persistent identifier resolution.

External identifiers (ORCID, ROR, Wikidata) are systematically captured in the Identity and Source entities and exposed in the metadata output, supporting persistent identifier-based aggregation by external harvesters.

5 Data Lifecycle and Format Transformations

A central contribution of this work is the explicit specification of the data lifecycle: the transformations that metadata fields undergo from user input to protocol output. We trace this lifecycle through four stages.

5.1 Stages 1–2: Form Input and JSON Persistence

The data entry application presents a tabbed interface, with each tab corresponding to one of the eight entity types. For multilingual fields, two parallel input controls are provided (one for Italian, one for English). For type-constrained

properties, the available options are dynamically filtered based on the selected item type, using the constraint matrix described in Section 4.3.

The application operates entirely in the browser, with work-in-progress data persisted in localStorage. When the operator submits, the complete dataset is serialized as a single JSON document and transmitted to the Backend API.

The submitted JSON document follows a standardized envelope structure:

```
{ "data": {
    "items":[...], "properties":[...], "media":[...], "
        identities":[...],
    "actors":[...], "contacts":[...],  "sources":[...], "
        relations":[...]
  },
  "metadata": {
    "infrastructure":"operas", "site":"roma",
    "timestamp":"2026-03-04T13:42:04.161Z", "version":"4.1-
        SECURE"
  }
}
```

The JSON layer preserves the normalized structure of the schema: each entity type occupies a separate array, and cross-entity relationships are maintained through identifier references. This normalized form simplifies editing operations (an Identity can be updated independently of the Items that reference it) but complicates the OAI-PMH output, where each record must be self-contained.

5.2 Stage 3 – Native OAI-PMH Format (h2iosc)

The OAI-PMH server, on receiving a request, reads the JSON files from the data repository and transforms them into XML records conforming to the `h2iosc` namespace (`http://h2iosc.org/h2iosc/item/`). This transformation involves two operations:

Denormalization. For each Item, the server resolves all foreign key references: Actor references to Identities, Identity references to Contacts and Sources, Item references to Properties and Media. The result is a single, deeply nested XML tree that is independently interpretable by any compliant harvester.

Restructuring. The flat multilingual fields from the JSON layer (e.g., `nameIT`, `nameEN`) are restructured into the h2iosc multilingual pattern, in which each multilingual element contains multiple `<h2iosc:value>` children with `xml:lang-style` language attributes:

```
<h2iosc:item xmlns:h2iosc="http://h2iosc.org/h2iosc/item/">
  <h2iosc:name>
    <h2iosc:value h2iosc:language="en">NormaTEI</h2iosc:
        value>
    <h2iosc:value h2iosc:language="it">NormaTEI</h2iosc:
        value>
  </h2iosc:name>
  <h2iosc:type>tool</h2iosc:type>
  <h2iosc:actors><h2iosc:actor>
```

```
    <h2iosc:role>author</h2iosc:role>
    <h2iosc:identity>[...type, label, sources: ORCID/ROR
        ...]</h2iosc:identity>
  </h2iosc:actor></h2iosc:actors>
</h2iosc:item>
```

This structural enrichment makes the XML representation semantically richer than the storage format, capturing the role-based actor model and the multilingual nature of the schema in a way that can be parsed without knowledge of storage-layer conventions.

5.3 Stage 4 – Dublin Core Mapping (oai_dc)

For interoperability with generic harvesters, the server provides a Dublin Core mapping. The mapping necessarily reduces information, since the 15 Dublin Core elements cannot capture the full expressiveness of the h2iosc schema. The mapping rules are summarized in Table 1.

Table 1. Mapping between H2IOSC fields and Dublin Core elements.

H2IOSC field	**Dublin Core element**
name (multilingual)	`dc:title` (English preferred, Italian fallback)
shortDescription	`dc:description`
`uri`, `id`	`dc:identifier`
`type`	`dc:type`
`createdAt`	`dc:date`
Property `keyword`	`dc:subject`
Property `language`	`dc:language`
Property `license`	`dc:rights`
Property `publisher`	`dc:publisher`
Actor with role creator or author	`dc:creator` (using identity label)
`documentationUri`	`dc:relation`
`permissionRequired` (if true)	`dc:rights` (string “Access restricted”)

This mapping is conservative: it prioritizes broad interoperability over completeness. Consumers requiring the full description should request the h2iosc metadata prefix. The four-stage lifecycle is a general pattern that applies to all fields in the schema. The key insight is that different stages have different design constraints: the HTML layer optimizes for usability, the JSON layer for editability, the native XML layer for expressiveness, and the Dublin Core layer for interoperability. The transformations reconcile these constraints, allowing each layer to be optimized for its specific purpose.

6 Security Architecture

The system implements multiple security layers appropriate for a research data infrastructure handling pre-publication metadata.

Authentication. The data entry web application is protected by Cloudflare Access, which provides email-based one-time-code authentication. Only operators with email addresses on the project's allowlist can access the application. The Backend API uses infrastructure-specific API keys, transmitted in the `X-API-Key` HTTP header. Each key is bound to a specific infrastructure-site combination (e.g., `operas-roma`), enforcing strict data isolation: a key issued for one site cannot be used to submit data for another site. API keys are stored only in hashed form (SHA-256). The plaintext key is shown to the operator at generation time and is not recoverable from the server. The key generation tool (a Python script distributed to infrastructure operators) computes the hash locally before committing it to the configuration repository, ensuring that plaintext keys never traverse the project's communication channels.

Input Validation. Infrastructure and site identifiers are validated against a strict regular expression pattern (`^[a-z0-9][a-z0-9_-]*$`) before being used in path construction, preventing path traversal attacks. The submitted JSON is validated against the schema before being accepted: type values are checked against the allowed enumeration; required fields are verified for presence and type; property codes are validated against the constraint matrix.

Access Control and Network Isolation. The system follows the principle of least privilege for repository access tokens. The OAI-PMH server uses a token scoped to read-only access on the data repository, while the Backend API uses a separate token with write access to the same repository and read-write access to the configuration repository. Neither token has access to source code repositories. Token scopes are configured using GitHub's fine-grained access tokens, which constrain permissions at the per-repository, per-permission-type level. Cross-Origin Resource Sharing (CORS) is restricted on the Backend API to the data entry application's domain, preventing unauthorized web applications from making requests. Rate limiting is enforced in memory (rather than on the filesystem, given the ephemeral file system of the Render free tier).

Audit Trail. The use of Git as the storage layer provides a complete, immutable audit trail of all data modifications, with commit-level traceability of every change. Every submission produces a Git commit signed by the Backend API, identifying the date, time, infrastructure, and site of the modification. The Git history can be inspected at any time without specialized tooling, and historical states of the data can be reconstructed deterministically.

7 Validation and Performance

OAI-PMH 2.0 Conformance. The system has been validated against the official OpenArchives validator.[5] The validation process consists of two steps: an initial check that verifies the Identify response and confirms administrator email ownership, and a complete check that exercises all six protocol verbs, validates response schemas, tests error handling, and verifies edge cases including

[5] https://www.openarchives.org/Register/ValidateSite

invalid arguments, non-existent identifiers, mixed datestamp granularity, and resumption token semantics.

Table 2. OAI-PMH 2.0 conformance validation results (OPERAS endpoint).

Validation aspect	**Result**
baseURL verification	PASS
Identify response	PASS (5 sub-tests)
`ListSets` (type/site set hierarchy)	PASS
`ListIdentifiers`	PASS
`ListMetadataFormats` (oai_dc)	PASS
`GetRecord` with valid id	PASS
`ListRecords` + `resumptionToken`	PASS
`ListRecords` + `set` filter	PASS
Datestamp granularity (days)	PASS
Error handling (13 invalid requests)	PASS (all correctly handled)
`badVerb` on unknown verb	PASS
`badArgument` on invalid parameters	PASS (10 cases)
`idDoesNotExist` for non-existent id	PASS
`badResumptionToken`	PASS
`noRecordsMatch`	PASS
HTTP POST support (`Identify`)	PASS
HTTP POST support (`GetRecord`)	PASS
Total tests passed	**38**
Total warnings	**0**
Total error count	**0**
Validation status	**COMPLIANT**

Coverage. The schema is fully implemented. All eight resource types are supported in the data entry application and serialized correctly in both metadata formats. All 35 property codes defined in the constraint matrix are recognized and validated. The eight role types for actors and the seven external identifier types for sources are fully supported. As of the validation date, the system serves 4 records for OPERAS-IT (concentrated in the Roma site) and 15 records for E-RIHS.it (distributed across the Firenze, Roma, Lecce, and Catania sites).

Performance and Scalability. We measured response times for representative queries against the deployed system, using curl with millisecond-level timing. Each query was executed N=20 times, and we report the mean response time and the standard deviation. The measurements were taken from a client located in Italy, against the production endpoint hosted on Render's free tier (which deploys to data centers in Oregon, USA – a deliberately worst-case latency configuration).

Table 3. Response time measurements (N=20 repetitions per query, mean ± std).

Query	**Response time**
`Identify` (E-RIHS)	177.8 ms ± 39.1 ms
`ListRecords` (E-RIHS, oai_dc, 15 records)	4736.8 ms ± 325.8 ms
`ListRecords` (E-RIHS, h2iosc, 15 records)	4892.3 ms ± 311.7 ms
`GetRecord` (single record, h2iosc)	4859.4 ms ± 274.8 ms

The end-to-end latencies in Table 3 are dominated by the per-request GitHub fetch-and-join (an I/O-bound step), not by record count or serialization. To characterize scaling independently of this I/O cost, we benchmarked response construction for synthetic collections from 100 to 50,000 records, comparing the paginated first page (size 100) against a single unpaginated response (Table 4). With pagination, construction time and payload stay effectively constant (≈3 ms, 56 KB) across three orders of magnitude, whereas the unpaginated response grows linearly to 1.6 s and 28 MB. Pagination thus bounds per-request cost; the fetch-and-join I/O, which explains the absolute latencies above, is addressed by caching and pre-computed denormalized views, left as future work.

Table 4. Response construction: paginated first page (size 100) vs. single unpaginated response, by collection size (mean of 10 runs).

Records	Paged (ms)	Paged	Full (ms)	Full
100	3.2	56 KB	2.5	56 KB
1,000	2.7	56 KB	24.8	561 KB
10,000	2.6	56 KB	293.3	5.6 MB
50,000	2.9	56 KB	1636.2	28.3 MB

8 Discussion

Alignment with FAIR Principles. The system supports the FAIR principles at multiple levels. **Findability** is supported through standardized metadata exposure via OAI-PMH, persistent identifiers (`oai:h2iosc:` identifiers with structured paths), and integration with external identifier systems (ORCID, ROR). **Accessibility** is supported through the use of standard protocols (HTTP, OAI-PMH) over the open Internet, with no authentication required for harvesting. **Interoperability** is supported through the dual-format output (Dublin Core for broad interoperability, h2iosc native for expressiveness) and through the use of controlled vocabularies referenced in property values. **Reusability** is supported by detailed metadata records, license declarations through the dedicated property, and the audit trail provided by the Git-based storage layer. **Implications for SSH Digital Library Practice.** Three aspects of the system have implications for SSH digital library practice more broadly. First, the multi-tenant model demonstrates that small federations of resource-constrained nodes can be served from a single, lightweight deployment, lowering the barrier to participation in metadata exchange. Second, the type-property constraint matrix offers a model for combining schema flexibility (multiple resource types) with semantic discipline (per-type constraints) – a balance that is often difficult to achieve in heterogeneous SSH catalogues. Third, the deployment on free-tier services demonstrates that production-grade digital library services can be provided at negligible operational cost, which is significant for sustainability beyond project funding periods. **Relationship to European Catalogue Ecosystems.** The system is designed for integration with the broader European catalogue ecosystem. The OAI-PMH endpoints can be harvested by any compliant aggregator, including the EOSC Marketplace, the SSHOC Open Marketplace, and OpenAIRE.

Conversely, the same OAI-PMH infrastructure can be configured to harvest from external sources, creating bidirectional flows. Such bidirectional harvesting between H2IOSC and SSHOC is a natural extension and is technically simple – it requires only the activation of an additional harvesting configuration.

Limitations and Future Work. Several limitations deserve acknowledgment, each corresponding to a planned development direction. The system now supports OAI-PMH Sets (selective harvesting by resource type and site) and `resumptionToken` pagination; the remaining optimization is caching the fetched-and-joined records, currently recomputed per request. Multilingual support is bilingual (planned: extension as required); deployment depends on third-party services (planned: evaluation of self-hosted alternatives for long-term sustainability). Additional future directions include a formal XSD schema for the h2iosc namespace, JWT-based end-to-end authentication, and bidirectional harvesting with SSHOC and EOSC.

9 Conclusions

We have presented a multi-tenant OAI-PMH 2.0 endpoint with a type-constrained resource description schema, designed for federated SSH research infrastructures and deployed for the H2IOSC project. The contributions of this work are: an architecture that combines multi-tenancy with strict logical separation, allowing a single deployment to serve multiple independent infrastructures; a resource description schema that supports eight resource types with multilingual fields, controlled vocabulary integration, and a type-property constraint matrix; a four-stage data lifecycle that transforms metadata across heterogeneous representations while preserving semantic integrity; a security architecture based on hashed API keys, fine-grained access tokens, and least-privilege principles; and a deployment model based exclusively on free-tier and open-source services, demonstrating that production-grade digital library services can be provided at negligible operational cost. Beyond the specific implementation, the generalizable contribution is the type-property constraint model: a formal, enforceable mechanism for semantic consistency across heterogeneous resource types, transferable to any multi-type metadata system. The system has been validated against the official OpenArchives validator with 38 tests passed and 0 errors, and is currently in production use for the OPERAS and E-RIHS infrastructures. Performance measurements confirm acceptable response times for the current scale, with clear paths for optimization at larger scales. Beyond the immediate H2IOSC application, the design choices documented here are offered as a reference for other federated SSH research infrastructure projects facing similar challenges. The combination of multi-tenancy, type-constrained schemas, and negligible-cost deployment is particularly relevant for emerging European initiatives that need to expose metadata at the European scale without the budget for dedicated infrastructure investments.

Acknowledgments. This work was carried out within the framework of the H2IOSC project (IR0000029), Mission 4 “Education and Research” – Component 2 “From re-

search to enterprise" – Investment Line 3.1 "Fund for the realization of an integrated system of research and innovation infrastructure", Action 3.1.1 "Creation of new IRs or strengthening of existing ones that contribute to Horizon Europe's Scientific Excellence objectives and networking".

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Lagoze, C., Van de Sompel, H., Nelson, M., Warner, S.: The Open Archives Initiative Protocol for Metadata Harvesting – Version 2.0. Open Archives Initiative (2002). https://www.openarchives.org/OAI/openarchivesprotocol.html
2. Open Archives Initiative: OAI-PMH Data Provider Validation. https://www.openarchives.org/Register/ValidateSite
3. Manghi, P., Bardi, A., Atzori, C., Baglioni, M., Manola, N., Schirrwagen, J., Principe, P.: The OpenAIRE Research Graph Data Model. Zenodo (2019). https://doi.org/10.5281/zenodo.2643199
4. Klein, M., Sanderson, R., Van de Sompel, H., Warner, S., Haslhofer, B., Nelson, M.L., Lagoze, C.: A Technical Framework for Resource Synchronization. D-Lib Magazine 19(1/2) (2013)
5. Broeder, D., Kemps-Snijders, M., Van Uytvanck, D., Windhouwer, M., Withers, P., Wittenburg, P., Zinn, C.: A Data Category Registry- and Component-based Metadata Framework. In: Proceedings of LREC 2010, pp. 43–47 (2010)
6. Sichera, P., Marras, C., Pasini, E.: A Multi-Infrastructure OAI-PMH Endpoint for Research Infrastructure Orchestration in the H2IOSC Project. Zenodo (2026). https://doi.org/10.5281/zenodo.19353859
7. Sichera, P., Marras, C., Pasini, E.: From Data Model to Interoperable Metadata: Implementing the H2IOSC Resource Description Schema across a Multi-Infrastructure Pipeline. Zenodo (2026). https://doi.org/10.5281/zenodo.19409795
8. Sichera, P., Marras, C., Pasini, E.: Orchestrating Research Infrastructure Services through OAI-PMH: The OPERAS-IT Approach within H2IOSC. Zenodo (2026). https://doi.org/10.5281/zenodo.19596981
9. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., et al.: The FAIR Guiding Principles for scientific data management and stewardship. Sci Data. 2016 Mar 15;3:160018. https://doi.org/10.1038/sdata.2016.18
10. Borgman, C.L.: Big Data, Little Data, No Data: Scholarship in the Networked World. MIT Press, Cambridge, MA (2015).
11. Dappert, A., Farquhar, A., Kotarski, R., Hewlett, K.: Connecting the Persistent Identifier Ecosystem: Building the Technical and Human Infrastructure for Open Research. Data Science Journal 16 (0): 28. (2017). https://doi.org/10.5334/dsj-2017-028
12. Tennant, J.P., Crane, H., Crick, T., Davila, J., Enkhbayar, A., Havemann, J., et al.: Ten Hot Topics around Scholarly Publishing. Publications. 7(2):34 (2019). https://doi.org/10.3390/publications7020034