Towards Declarative Linked Data Backends.

Generating the RELEVEN Graph API from RDF Path Expressions.

Lukas Plank[1], Kevin Stadler[1]

[1] *Austrian Center for Digital Humanities and Cultural Heritage*

Proceedings of the Posters and Demos Track, co-located with SEMANTiCS'25: International Conference on Semantic Systems, September 3–5, 2025, Vienna, Austria.

Abstract

The RELEVEN project implements a heavily reified CIDOC-CRM-based knowledge graph to represent historical claims with full contextual provenance. Exposing such complex semantic data to consuming applications presents significant technical challenges. Our demonstration showcases a model-driven approach for building REST APIs on top of SPARQL endpoints by combining RDFProxy—a Python library that maps SPARQL query results to Pydantic models—with WissKAS, a command-line tool that automatically generates RDFProxy endpoints from a declarative RDF path expression language.

Keywords

RDF Graph, REST API, Python, Pydantic, SPARQL

1. Introduction

The RELEVEN project [1] is methodologically based on the premise that digital representations of historical data should take the form of structured, contextualized assertions rather than isolated factual statements. Semantic technologies are well suited to provide the expressiveness needed for such *explicitly assertional assertions* and allow the specification of formal-ontological models to capture the essential contextuality of historical claims. RELEVEN implements this approach through the Structured Assertion Record (STAR) model, a CIDOC-CRM-based [2] semantic pattern that *reifies* historical claims in order to explicitely capture their provenance, authority and evidentiary grounding. The indirection required for reification is achieved by utilizing CIDOC CRM's Attribute Assignment facilities to project a given assertion onto a data entity (i.e. another assertion) representing that assertion. This technique allows a STAR graph to encode not only what is being claimed, but also who makes the claim and on what basis (and ergo *that* a claim is being made), as well as multiple, even conflicting perspectives within a coherent semantic framework. [3]

The inherent complexity of heavily reified knowledge graphs, however, creates significant technical implementation challenges, which, in RELEVEN, are addressed through a layered architecture essentially comprised of three core componentes:

- GraphDB as the triplestore persistence layer,
- WissKI [4], a LOD-focused virtual research environment, that serves as the CMS and data management interface and also provides a declarative path expression language for defining semantic data shapes,
- RDFProxy [5], a Python library developed within the RELEVEN project for mapping SPARQL result sets to Pydantic models [6], enabling a FastAPI [7] powered REST layer.

Although WissKI path expressions do not enforce and validate graph constraints like e.g. SHACL, they facilitate a concise means for formally declaring semantic shapes and serve as the structural modelling framework for the core RELEVEN knowledge graph implementation.

Building on this, another RELEVEN-developed tool, WissKAS (the *WissKI Adapter Serializer*) [8], enables the automatic generation of RDFProxy-compliant SPARQL queries and corresponding Pydantic models directly from WissKI path definitions. This allows for the seamless, declarative construction of of the entire RELEVEN REST API, fully aligned with the underlying data model. The proposed demo will showcase this pipeline in action, highlighting the integration of semantic modelling, data access, and declarative API generation within the RELEVEN graph architecture.

2. RDFProxy

While Knowledge Graphs and CRM-based ontologies provide the semantic expressiveness required for modeling contextualized historical data, the RDF technology stack offers only limited support for delivering graph data to consuming applications. SPARQL, the W3C-standard query language and specification for retrieving data from RDF graphs, is primarily designed for pattern-based querying and returns result sets of whatever binding projections satisfy a given graph pattern in flat, basically tabular structures. Although effective for graph navigation and data extraction, SPARQL provides no inherent means of declarative result shape transformation and does not natively support the retrieval of potentially deeply nested data structures, subset aggregation, result pagination, or schema validation. As a result, the expressive power of RDF models and reified graphs in particular stands in contrast to the essentially row-based structure of SPARQL result sets, and integrating RDF-based data into client applications typically requires extensive and endpoint-specific post-processing logic to convert SPARQL results into consumable data shapes.

The RDFProxy Python library [5] addresses these limitations by implementing a mapping mechanism that allows the projection of SPARQL result sets onto Pydantic models, enabling the structured, type-enforced representation of RDF graph data based on Python's type annotation system. This integration allows RDFProxy to leverage the powerful capabilities of Pydantic and, by extension, FastAPI for working with RDF datasets, including model pre- and post-validation hooks, along with custom data serializers, native support for asynchronous API calls and automatic generation of detailed API documentation according to the OpenAPI standard.

Internally, RDFProxy dynamically modifies the incoming SPARQL query - for instance, by injecting certain clauses and subqueries to implement purely SPARQL-based pagination - and utilizes a dataframe abstraction for highly efficient grouping and aggregation operations over SPARQL result sets before mapping the processed data onto a given Pydantic model definition.

The primary code interface for defining API routes with RDFProxy is the *rdfproxy.SPARQLModelAdapter* class, which is designed to establish and handle a connection between a triplestore, a SPARQL query and a Pydantic model on a per-route basis.

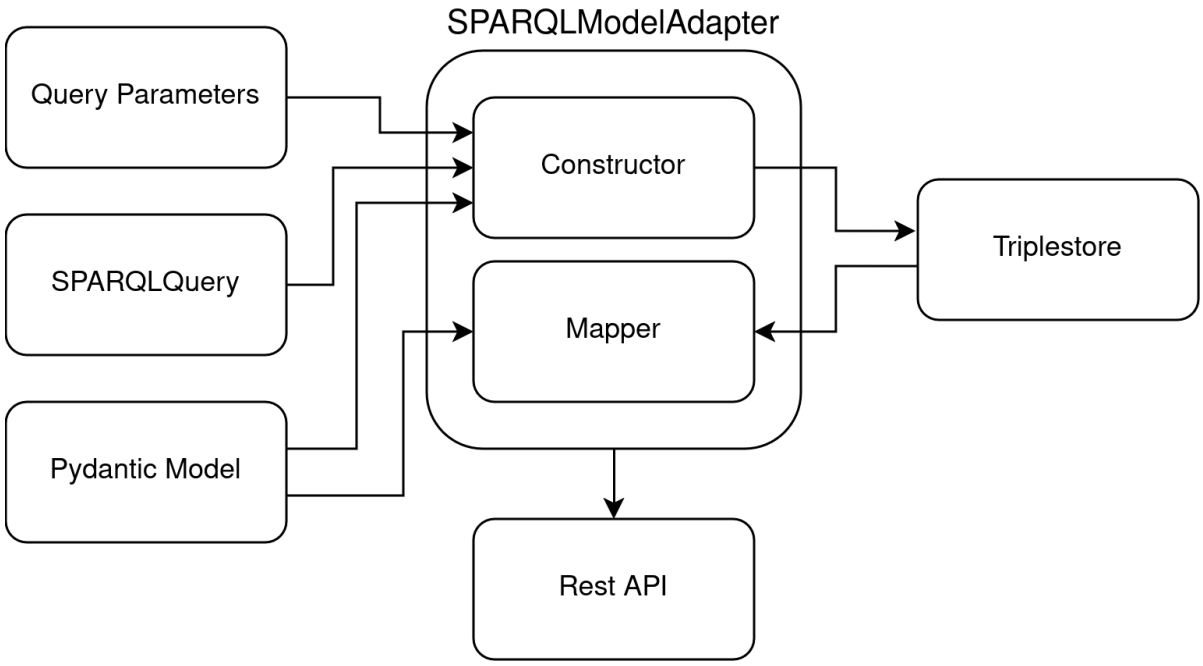

Figure 1: SPARQLModelAdapter Architecture: Dyncamic Query Construction and Result Mapping in RDFProxy.

To summarize, the RDFProxy Python library aims to provide a generic solution for building modern REST APIs on top of SPARQL endpoints, allowing backend implementers to leverage the powerful model-driven abstractions of Pydantic and enabling performant transformations of SPARQL query results into structured, type-safe API responses suitable for production workloads.

3. WissKAS

Building an RDFProxy endpoint requires two main components:

- a SPARQL SELECT query for retrieving data from a triplestore
- a (potentially nested) Pydantic model that defines the desired API response shape and data validation rules.

Typically, RDFProxy-compliant SPARQL queries and their corresponding Pydantic models are implemented manually by backend implementers. However, given a sufficiently expressive structural model-to-RDF mapping, RDFProxy endpoints can also be derived automatically.

WissKI [4], a Drupal-based content management system that serves as the data management and CMS layer within the RELEVEN graph technology stack allows users to define hierarchical data models through a web interface, where each field specifies and maps a sequence (or "path") of RDF classes and predicates. A group of such paths originating from the same RDF class forms a model definition, which can be nested and include relational references to other model types.

These model-to-RDF path relations — called Pathbuilder Definitions in WissKI — are internally represented as XML structures, which makes them amenable to automated processing. The *WissKI Adapter Serializer* (WissKAS) [8] is a RELEVEN-developed Python command-line tool designed to generate complex RDFProxy endpoints directly from such Pathbuilder Definitions. Based on CLI options that allow users to specify the desired REST API response shape using a simple filter syntax, WissKAS can automatically derive RDFProxy endpoints that provide data views of configurable granularity, depending on how relational boundaries between model elements are traversed. This

flexibility allows dynamic generation of multiple views of the same data at different levels of detail, going beyond the fixed structures defined in the ontology.

Given a Pathbuilder Definition file and a set of endpoint filter specifications, the WissKAS command-line tool is able to construct:

- for each desired endpoint:
  - a Pydantic model (often composed of nested classes) with types inferred from the WissKI definitions.
  - a SPARQL query with a projection that includes all variables required by the model, reflecting the union of all RDF paths defined for the selected fields.

- a FastAPI entry point with Python function definitions for all derived routes, each of which instantiates an *rdfproxy.SPARQLModelAdapter* and calls its methods to execute queries against a Triplestore upon endpoint invocation.

By automating the generation of RDFProxy models and queries from RDF path declarations, WissKAS significantly reduces the effort required to create and maintain structured API endpoints. Within the RELEVEN project, WissKAS serves as a central integration facility in a broader semantic modeling workflow, linking domain-specific data modeling in WissKI to accessible, standards-compliant RESTful interfaces.

4. Demo Proposal

Our demonstration will showcase the RDFProxy-based REST layer of the RELEVEN graph implementation and automatic API generation from declarative RDF path expressions using the WissKAS backend serializer, highlighting the model-based transformation of complex RDF data into typed API endpoints.

Declaration on Generative AI

During the preparation of this work, the author(s) used GPT-4o for grammar and spelling checks.

References

[1] RELEVEN Project. University of Vienna. Available at: https://releven.univie.ac.at/ (visited on 2025-07-03).

[2] CIDOC Conceptual Reference Model (CIDOC CRM). International Committee for Documentation of the International Council of Museums (ICOM-CIDOC). Available at: https://cidoc-crm.org/ (visited on 2025-07-03).

[3] Andrews, T. et al. (2024). "Re-Evaluating the Eleventh Century through Linked Events and Entities". *Historical Studies on Central Europe*, 4(1), pp. 217–245. DOI: 10.47074/HSCE.2024-1.12.

[4] WissKI. Wissenschaftliche Kommunikationsinfrastruktur (Scientific Communication Infrastructur). Available at: https://wiss-ki.eu/de (visited on 2025-07-03).

[5] RDFProxy. ACDH-CH, Austrian Academy of Sciences. GitHub repository. Available at: https://github.com/acdh-oeaw/rdfproxy (visited on 2025-07-03).

[6] *Pydantic Documentation*. Available at: https://docs.pydantic.dev/latest/ (visited on 2025-07-03).

[7] *FastAPI Documentation*. Available at: https://fastapi.tiangolo.com/ (visited on 2025-07-03).

[8] WissKAS. ACDH-CH, Austrian Academy of Sciences. GitHub repository. Available at: https://github.com/acdh-oeaw/wisskas/ (visited on 2025-07-03).