SULO-Compliant Schema Builder: a web-based tool bridging domain schemas with ontologies using SULO

Remzi Çelebi[1,*,†], Catalina Martínez Costa[2], Stefan Schulz[3,4] and Michel Dumontier[1]

[1]Institute of Data Science, Department of Advanced Computing Sciences, Maastricht University, Netherlands
[2]Department of Informatics and Systems, University of Murcia, Spain
[3]Institute for Medical Informatics, Statistics and Documentation, Medical University of Graz, Austria
[4]Averbis GmbH, Germany

Abstract

Experts who design biomedical data schemas such as HL7 FHIR, OMOP CDM, or relational databases, and ontologists who formalize domain semantics in OWL operate with fundamentally different representational commitments. Existing tools address one community or the other but not both simultaneously, creating a persistent gap between schemas and formally grounded ontologies.

We present the SULO-Compliant Schema Builder, an open-source web application that enables domain experts to define classes and properties, align them interactively to the Simplified Upper-Level Ontology (SULO), and automatically generate four artefacts from a single schema model: RDFS Turtle, OWL with SULO-compliant equivalence axioms and property restrictions, SHACL shapes and Mermaid UML diagrams. A declarative mapping-pattern mechanism, analogous to SPARQL triple templates, compiles domain relations into SULO relations without requiring the user to author OWL directly. A clinical health record case study demonstrates the practical viability of the approach.

The SULO-Compliant Schema Builder integrates schema authoring with ontology-driven semantic modeling through guided upper-level alignment and reusable mapping patterns. The resulting workflow enables users to generate logically enriched OWL artefacts while continuing to work within a schema-oriented design paradigm. The tool is freely available at https://github.com/MaastrichtU-IDS/sulo-schema-builder.

Keywords

ontology alignment, upper-level ontology, SULO, SHACL, OWL DL, RDF schema, mapping patterns, FHIR, OMOP, schema design

CEUR-WS.org/Vol-4342/paper8.pdf

1. Introduction

The biomedical informatics landscape is populated by well-established data standards, HL7 FHIR [1], OMOP CDM [2], openEHR [3], designed by experts who think in terms of tables, resources, and attributes rather than description-logic axioms. In parallel, the ontology community has invested decades in constructing upper-level frameworks such as the Basic Formal Ontology (BFO) [4], DOLCE [5], and the Simplified Upper-Level Ontology (SULO) [6], that provide principled, interoperable grounding for domain entities by upper-level categories. The gap between these two worlds is well-documented [7, 8]: schema designers regard ontologies as unnecessarily complex, while ontologists find flat schemas semantically impoverished and ambiguous. Similarly, schema designers tailor data structures for specific use cases without committing to the ontological characteristics to which these structures refer, while the latter is the main concern of ontologists who, in turn, tend to be indifferent regarding data structures referring to domain entities.

Several tools have attempted to narrow this gap from different angles. LinkML [9] provides a YAML-based schema language that compiles to OWL, SHACL, JSON Schema, and SQL from a single source, but requires command-line tooling and produces only basic OWL class mappings rather than

Proceedings of FOIS 2026 Satellite events co-located with the 16th International Conference on Formal Ontology in Information Systems (FOIS 2026), September 21–25, 2026, Vitória, Brazil

*Corresponding author: Remzi Celebi, remzi.celebi@maastrichtuniversity.nl
†These authors contributed equally.
0000-0001-7769-4272 (R. Çelebi); 0000-0003-1857-1744 (C. M. Costa); 0000-0001-7222-3287 (S. Schulz); 0000-0003-4727-9435 (M. Dumontier)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-27

full property restriction chains. Chowlk [10] converts UML diagrams drawn in diagrams.net into OWL Turtle, but is a one-directional converter rather than an interactive designer. model2owl [11] transforms UML XMI models into OWL and SHACL for institutional publishing frameworks. Astrea [12] automatically generates SHACL shapes from existing OWL ontologies. Protégé [13], the leading desktop ontology editor, supports full OWL DL authoring but requires ontology expertise and provides no integrated schema-design workflow or live upper-ontology alignment. None of these tools provides an integrated, browser-based environment that combines (a) interactive schema design and export in multiple standard formats (RDFS, SHACL, and UML), (b) live upper-ontology class and relation mapping with autocomplete, (c) OWL DL ontology with formal axioms derived from class and property mapping patterns, and (d) consistency checking, all from a single coherent model.

Critically, no existing tool couples upper-ontology alignment to executable OWL restriction generation: LinkML aligns classes to external vocabularies but does not compile property-level mapping patterns into restriction chains, so the semantic gain is limited to class-level typing; Chowlk and model2owl convert existing diagrams or UML models into OWL but do not support live, in-browser alignment during schema design, so any upper-ontology error is only detectable after the fact, outside the design loop; and Astrea generates SHACL from OWL rather than the reverse, so it presupposes that an ontology already exists rather than helping a schema-first user build one incrementally. The Schema Builder is therefore the first tool, to our knowledge, in which upper-ontology alignment and OWL restriction generation happen inside the same authoring step that a schema designer already performs, rather than as a separate downstream translation.

Here we present the SULO-Compliant Schema Builder, an open-source web application that addresses this gap. The tool is motivated by three design objectives: (i) to support the design of (RDF) schemas and complementary formal OWL ontologies through upper-ontology alignment; (ii) to enable declarative mapping patterns for interoperability across clinical standards; and (iii) to serve as a pedagogical instrument that makes the expressive advantages of formal ontologies tangible to practitioners fluent in schemas but unfamiliar with OWL and/or principles of ontology-driven domain analysis independent of data structures and use case requirements.

The tool enables researchers and knowledge engineers to define domain-specific classes and properties and align them with corresponding SULO concepts and relations while preserving existing instance-level data (known in OWL as the ABox — the set of individual assertions, as opposed to the TBox, which defines class and property structure). Users begin by specifying the classes and properties of their custom schema. Classes are mapped to relevant SULO ontological upper-level classes, while properties, which in standard database schemas typically carry only domain and range constraints, are enriched into fuller semantic structures. This mapping process is guided by a pattern-based mechanism analogous to SPARQL triple patterns. This entails that rather than representing a relationship as a simple subject–property–object triple, the tool "unfolds" each relation into a more descriptive subgraph using reserved placeholders (`?this` and `?value`) to denote the original subject and object. For instance, a property such as *has healthcare provider* can be expressed as a SULO-compliant graph pattern in which a clinical visit is modeled as a process involving a participant with the provider role. Such a subgraph would then not require any other relations than the SULO object properties `has-participant` and `has-feature`.

From a single schema configuration, the tool produces two complementary outputs. First, a standard schema in various formats (RDFS, SHACL and UML) is generated, capturing basic class hierarchies and property domain and range relationships. Second, a SULO-aligned OWL ontology is produced, incorporating complex class expressions and property restrictions that enable formal logical reasoning and systematic error detection. Both outputs are exportable as standard ontology files, compatible with editors such as Protégé, allowing for further inspection and validation of class hierarchies and restrictions.

The SULO-Compliant Schema Builder is openly accessible on GitHub, where collaborators can download the source code, define their custom schemas, and consult usage documentation. This ensures reproducibility and supports broader adoption across research communities working with semantic data integration.

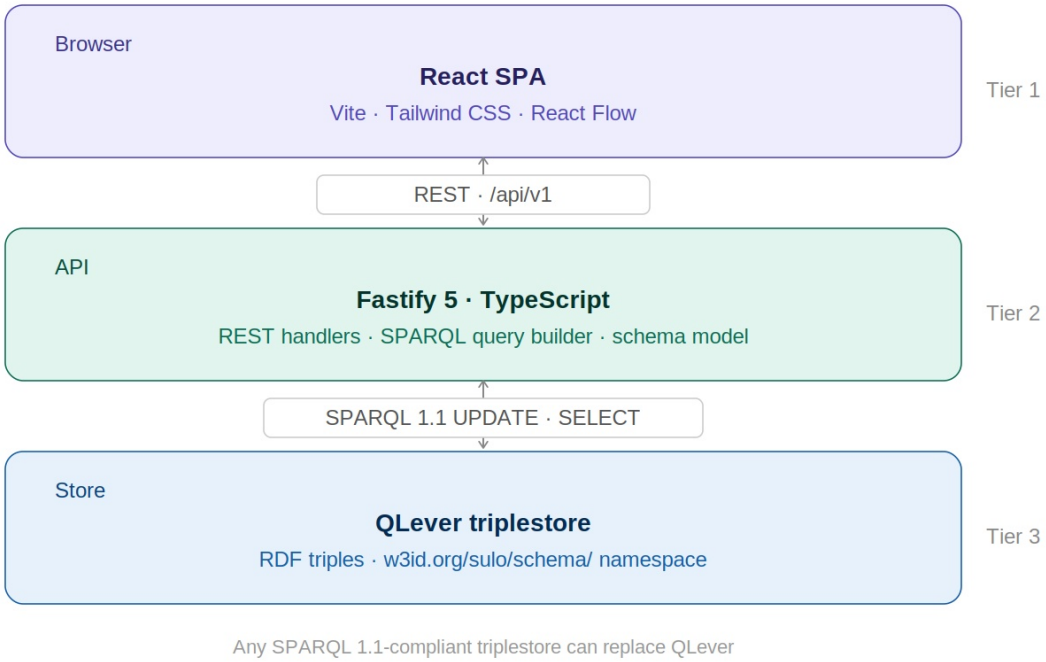

Figure 1: Three-tier architecture of the SULO-Compliant Schema Builder. The browser-based React SPA communicates with a Fastify REST API, which in turn issues SPARQL queries to a QLever triplestore storing all schema artefacts as RDF.

2. Methods

2.1. System architecture

The Schema Builder is a three-tier web application (Fig. 1). The frontend is a React single-page application (Vite / Tailwind CSS / React Flow) communicating with a Fastify 5 / TypeScript REST API via `/api/v1` endpoints. The API issues SPARQL 1.1 UPDATE and SELECT queries to a QLever [14] triplestore. All schema data are stored as RDF triples under the namespace https://w3id.org/sulo/schema/. Because the storage layer uses only standard SPARQL 1.1, any compliant triplestore can serve as the backend.

2.2. Data model

A Schema contains an ordered set of Classes and Properties. Each Class carries: (i) a local name used as the IRI fragment (e.g. `:ClinicalVisit`); (ii) a human-readable label and description; (iii) an optional `mapsToConceptIri` pointing to the aligned upper-ontology concept (e.g. https://w3id.org/sulo/Process); and (iv) an optional `superClassId` for intra-schema inheritance expressed as `rdfs:subClassOf`. Each Property carries: a name, label, and type (object or datatype); a `domainClass` and `rangeClass` (or XSD datatype for datatype properties); an `isRequired` flag; and a mapping pattern, a list of `TripleTemplate` objects, each comprising a subject variable, a predicate IRI, and an object (either a variable or an IRI constant).

2.3. Alignment with SULO via mapping

The mapping process aligns a custom data schema with the Simplified Upper-Level Ontology without requiring changes to the underlying instance-level data (ABox). It proceeds as follows:

1. **Class Mapping:** Domain-specific classes (e.g., clinical visit, person, device) are mapped to their corresponding upper level superclasses within the SULO hierarchy.

2. **Property Mapping:** Domain relationships (e.g., `hasPatient` or `hasHealthcareProvider`) are transformed into detailed semantic structures using mapping patterns, analogous to SPARQL triple patterns. Rather than a simple subject–object link, each relation is "unfolded" into a descriptive subgraph using two placeholders: `?this` (the subject of the original property) and `?value` (the object or range of the original property).

Consider the following running example: a `ClinicalVisit` has a patient who is a `Person`. A schema designer would model this as a simple subject–property–object triple. An ontologist would insist that the patient participates via a SubjectOfCareRole, an intermediate node needed to distinguish the patient's role in one visit from their role in another. Bridging these two representations, automatically and without OWL expertise, is the central problem this tool addresses. It is important to be precise about what is and is not removed from the user's task. The tool eliminates the need to write OWL syntax: the triple-template form is authored entirely through predicate and class pickers in the browser, and the corresponding restriction axioms in Table 1 are generated automatically. It does not eliminate the need to understand SULO's categorial commitments, such as the distinction between a process and the information object that describes it, or between parthood and mere association. Choosing an inappropriate SULO relation for a given domain property remains possible and, as the modeling errors in §3.2 illustrate, is in practice the most common source of unsatisfiability once a schema has been aligned. The property `hasPatient` on `ClinicalVisit` can be expressed with the three-hop SULO role-bearer pattern:

```
?this  sulo:hasParticipant  ?role
?role  rdf:type             :SubjectOfCareRole
?role  sulo:isFeatureOf     ?value
```

The `buildOwlExpr` compiler performs a recursive descent over the pattern, treating each triple template as an axiom. Starting from `?this` (the domain class), it: (i) collects all triples with the current variable as subject; (ii) for `rdf:type` triples with a constant object, adds the type as an `owl:intersectionOf` member; (iii) for triples whose object is `?value`, emits an `owl:someValuesFrom` restriction pointing at the range class; and (iv) for triples whose object is another variable, recurses to build a nested restriction. The above pattern compiles to the expression shown in Table 1.

Table 1
OWL Turtle and Manchester Syntax for the `hasPatient` mapping pattern.

OWL Turtle

```
_:domain_hasPatient
  owl:equivalentClass [
    a owl:Class ;
    owl:intersectionOf (
      :ClinicalVisit
      [ a owl:Restriction ;
        owl:onProperty
          sulo:hasParticipant ;
        owl:someValuesFrom [
          a owl:Class ;
          owl:intersectionOf (
            :SubjectOfCareRole
            [ a owl:Restriction ;
              owl:onProperty
                sulo:isFeatureOf ;
              owl:someValuesFrom
                :Person ]
          ) ]])] .
```

Manchester Syntax

```
:domain_hasPatient
EquivalentTo:
 :ClinicalVisit
 and (sulo:hasParticipant
   some (
     :SubjectOfCareRole
     and (sulo:isFeatureOf
       some :Person)))
```

This restriction formally captures the SULO Process-Role-Object (PRO) design pattern in full description logic and defines the domain restriction for the `hasPatient` relation. Crucially, the user who produced this expression authored only a triple-template form in the browser; no OWL axiom was written by hand. The companion function `buildReverseOwlExpr` performs the same traversal in the inverse direction, emitting `owl:inverseOf` restrictions, enabling the same pattern list to support both forward and reverse SPARQL-based queries.

2.4. Export pipeline

From a single input schema configuration, the tool generates four complementary artefacts:

- **RDFS Schema.** A Standard RDF Schema with basic class hierarchies and property domain/range definitions. An RDFS vocabulary defining `owl:Class` hierarchies, `rdfs:label`, `rdfs:comment`, `rdfs:domain`, and `rdfs:range`. This is the schema view familiar to linked-data practitioners and readable without OWL knowledge.
- **OWL + SULO.** A SULO-aligned OWL Ontology (Fig. 3) incorporating the full restrictions and semantics defined in the mapping patterns. Extends the plain export with `owl:equivalentClass` axioms for every class with a `mapsToConceptIri` and with nested `owl:someValuesFrom` restrictions compiled from the mapping patterns. Union ranges generate `owl:unionOf` constructs.
- **SHACL.** One `sh:NodeShape` per class, using schema-native predicates as `sh:path` values. Union ranges generate `sh:or` blocks; required properties carry `sh:minCount 1`. Shapes are intentionally free of SULO predicates so that they validate instance data expressed in the schema's own vocabulary, not upper-ontology paths.
- **Mermaid UML.** A `classDiagram` in Mermaid syntax, suitable for embedding in GitHub Markdown or mermaid.live, with inheritance and association arrows rendered appropriately. (see Fig. 4)

The key advantage of this approach is that researchers can switch between a simple schema-based representation and a highly expressive SULO-based one, enabling controlled comparisons for tasks such as error detection and neural network performance evaluation, without manually converting any underlying database records.

2.5. User interface

The GUI integrates schema-editing and ontology-alignment tasks within a single workflow through three main functionalities:

- **Alignment with upper-level ontologies.** When adding or editing a property, the user constructs a mapping pattern/path that unfolds the relation by bridging the domain class (`?this`) to range class (`?value`) as shown in Fig. 2.
- **Export tabs.** The export modal displays the plain RDF Schema and OWL+SULO as well as SHACL and UML representations (Fig. 4), making the provenance of each formal axiom directly traceable to a specific schema design choice.
- **Consistency checking.** This runs a full OWL DL reasoner over the schema without leaving the editor. It submits the generated OWL + SULO artefact to a backend service, which merges it with the complete SULO ontology and invokes HermiT (via ROBOT [15]) to compute two verdicts: whether the merged ontology is `consistent`, and which classes are `unsatisfiable` with no instances even when the ontology is otherwise consistent. The result is returned as a list of clashes, each naming the offending class or individual together with the reasoner's explanation (the minimal set of axioms that entails the contradiction). Note that this in-browser check covers schema-level (TBox) reasoning; instance data must be added and exported for full ABox-level consistency checking.

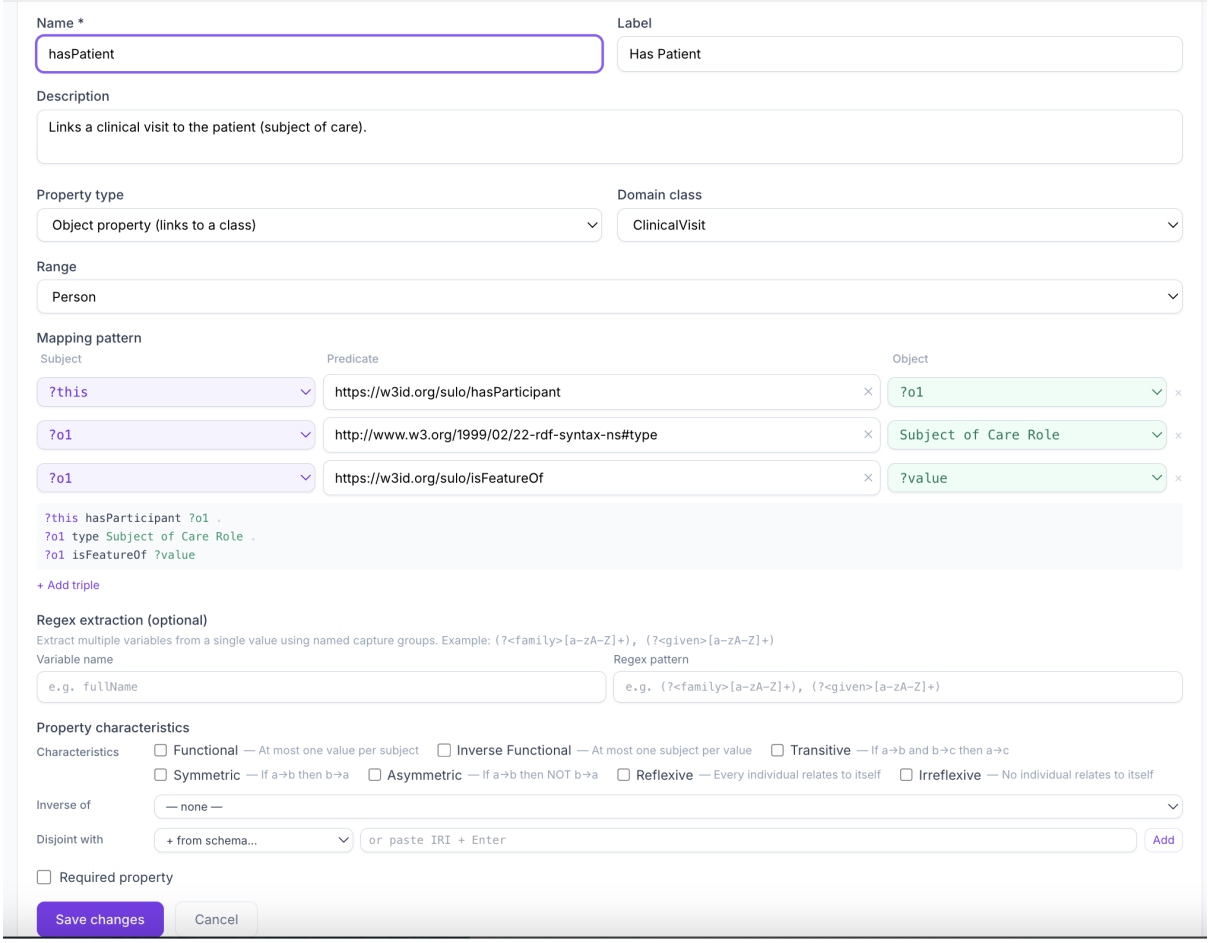

Figure 2: Properties panel — mapping patterns displayed inline; each row shows the mapping pattern with SULO predicates.

3. Clinical Case Study and Modelling Error Detection

3.1. Clinical health record case study

To evaluate the tool's practical viability, we constructed a Clinical Health Record Schema modelling a representative subset of the domain and inspired by the SPHN schema [16]. The schema, loadable via the *Load Example* button, comprises:

- 32 classes: 30 aligned to SULO concepts (Process, Role, Quality, SpatialObject, InformationObject, Quantity) and 2 aligned to SNOMED CT URIs (ObservableEntity: http://snomed.info/id/363787002; SCT_Procedure: http://snomed.info/id/71388002).
- 41 properties: 33 object properties linking schema classes and 8 datatype properties mapping to XSD literals.
- 5 subclass relationships.

3.2. Detecting Modelling Errors — Examples

One of the principal advantages of aligning a domain schema to SULO and generating OWL DL artefacts is that a description-logic reasoner can detect modelling errors that are invisible to schema validators and SHACL engines. SULO declares two-top-level categories, Object and Process, as disjoint. Under Object, it defines Capability, InformationObject, Quality, and Role as mutually disjoint via `owl:AllDisjointClasses`. Any instance data or class axiom that violates these constraints produces an unsatisfiable class or a direct contradiction, surfaced as a reasoner clash. The five examples below

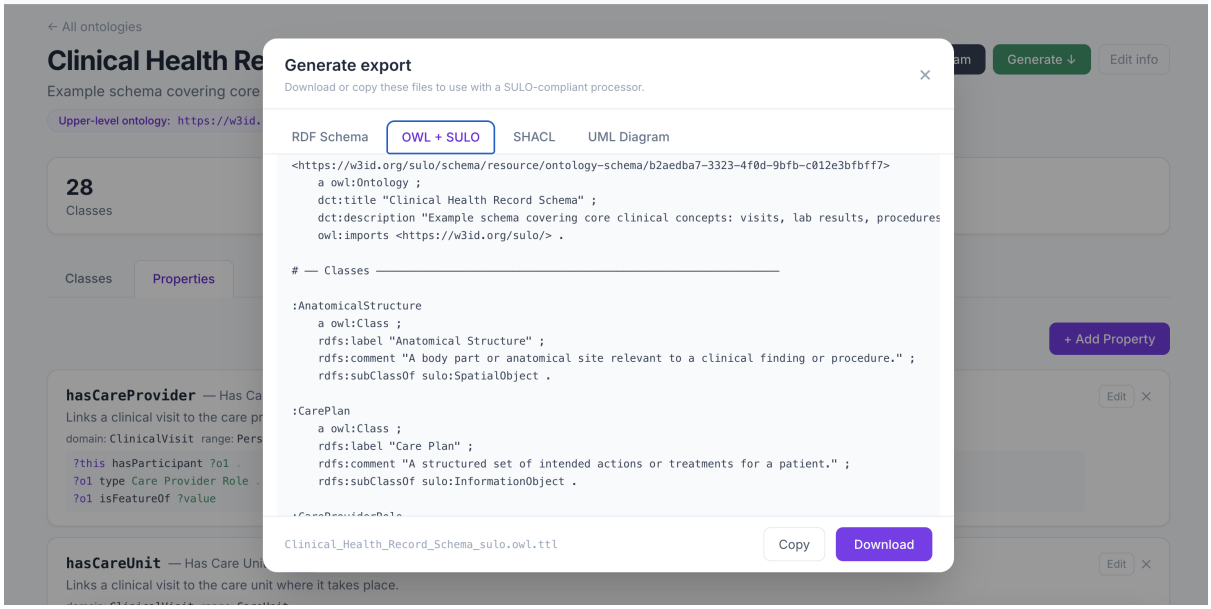

Figure 3: Export modal — OWL + SULO tab showing prefix declarations and `owl:equivalentClass` restrictions derived from mapping patterns.

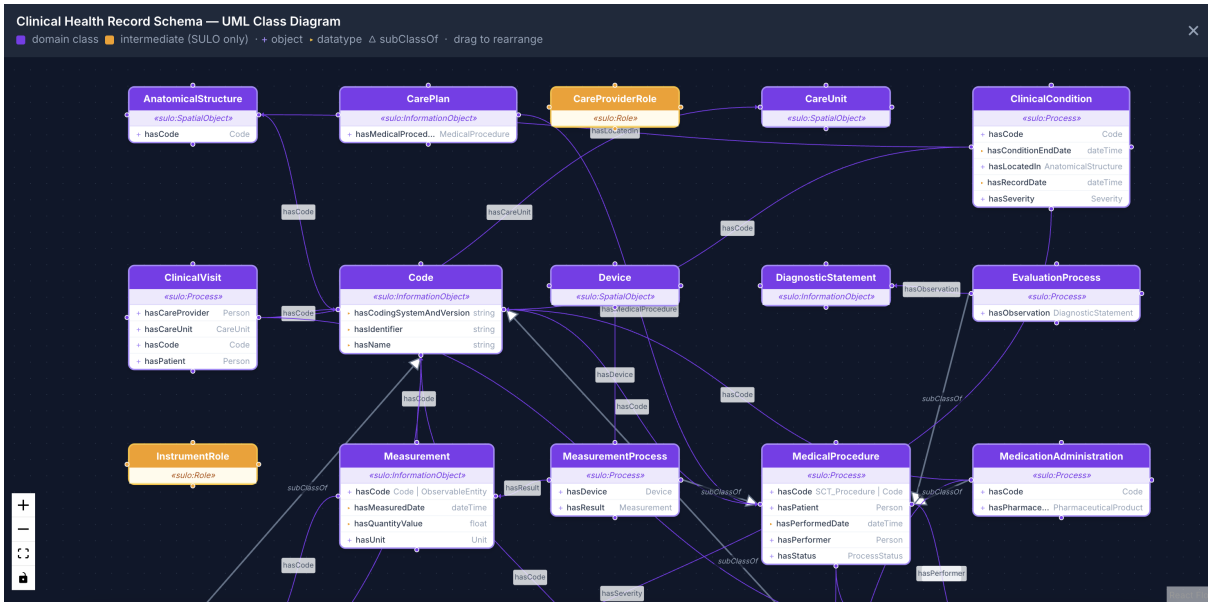

Figure 4: UML Diagram with interactive visualisation of schemas.

illustrate this in the clinical health record ontology, using realistic modelling mistakes when creating clinical data schemas and knowledge graphs. The four examples below cover three error patterns: (1) unsatisfiable class definitions from direct subclass conflicts, (2) instance-level contradictions where a valid-looking class accumulates incompatible inferred types, and (3) participant category errors where the wrong SULO relation is chosen

3.2.1. Example 1 — Unsatisfiable class definition via subclass conflict

A structurally common error in the clinical information models deployed is conflating a clinical condition with the diagnostic statement about that condition. A clinical condition is a pathological process in the patient whereas a diagnostic statement is an information object that refers to that process. While the

distinction is well understood in principle, it is routinely violated in the data structures of widely used terminologies and record models. The following class definition mixes them:

```
@prefix chr: <https://w3id.org/sulo/schema-builder/clinical-health-record/> .
chr:ClinicalFinding
    rdfs:subClassOf chr:DiagnosticStatement ; # -> sulo:InformationObject
chr:ClinicalFinding
    rdfs:subClassOf ex:ClinicalCondition . # -> sulo:Process
```

Any instance of `chr:ClinicalFinding` would simultaneously be a `sulo:Object` (via InformationObject) and a `sulo:Process`, two categories that SULO declares pairwise disjoint. The class is therefore unsatisfiable, and every individual typed as `chr:ClinicalFinding` immediately creates inconsistency. This example has particular educational value: it highlights the phenomenon of a representational use–mention confusion, exposing a category mistake at the schema level and making the distinction between referent and representation explicit and machine-verifiable.

3.2.2. Example 2 — instance-level conflict

A second class of error occurs when an automatically generated class accumulates properties whose domains map to disjoint SULO upper-level categories. Unlike the previous case, the class definition itself appears valid, and the contradiction emerges only through instance-level property usage.

```
ex:Observation
    rdfs:subClassOf sulo:InformationObject .
```

A representative instance:

```
ex:blood_test_1 a ex:Observation ;
    chr:hasPerformedDate "2021-06-15T10:00:00+02:00"^^xsd:dateTime ;
    chr:hasPerformer ex:nurse_jane ;
    chr:hasQuantityValue "5.4"^^xsd:float ;
    chr:hasUnit <http://unitsofmeasure.org/ucum#10*9/L> .
```

This models a single individual as both the blood-test procedure and its recorded result. Through the property mapping (`?this sulo:hasParticipantIn ?role. ?role a PerformerRole. ?role` and `sulo:isFeatureOf ?value`), `chr:hasPerformer` is inferred to have a domain of `sulo:Process`, implying that `ex:blood_test_1` should be a `sulo:Process`. At the same time, because `ex:Observation` is a subclass of `sulo:InformationObject`, the individual is also inferred to be a `sulo:Object`.

Because `sulo:Object owl:disjointWith sulo:Process` is asserted directly in SULO, OWL DL reasoner materializes the contradiction. The reasoner therefore classifies `ex:blood_test_1` as both a `sulo:Object` and a `sulo:Process`, making the ontology inconsistent. This is precisely the kind of modelling error that remains invisible to SHACL but is exposed through SULO-based OWL reasoning.

3.2.3. Example 3 — an information object given a participant

A measurement result, an information object recording a value, is assigned a patient through the participant pattern:

```
ex:measurement_7 a chr:Measurement ;
    chr:hasSubject ex:patient_3 .
```

`chr:Measurement` maps to `sulo:InformationObject`, and `chr:hasSubject` compiles to the pattern:

```
?this sulo:hasParticipant [
    a chr:SubjectRole ;
    sulo:isFeatureOf ?value ] .
```

SULO declares `sulo:hasParticipant rdfs:domain sulo:Process`: only a process has participants. The result is therefore forced to be a `Process`, contradicting its `sulo:Object` membership. The mistake is a category error between the measuring *process*, which legitimately has the patient as a

participant, and the measurement *result*, an information object that merely records it. The fix either attaches the patient to the underlying process, or states that the result `sulo:refersTo` the patient.

3.2.4. Example 4 — quantifying a quality (age)

It is tempting to model a patient's age as an intrinsic *quality* of the person, and then to record the number directly on it:

```
chr:Age rdfs:subClassOf sulo:Quality .

ex:age_1 a chr:Age ;
    chr:hasValue "45"^^xsd:decimal .
```

`chr:Age` maps to `sulo:Quality`, which is fine on its own. But the moment a value is attached via `chr:hasValue`, which maps to `sulo:hasValue`, and `sulo:hasValue rdfs:domain sulo:InformationObject`, so `ex:age_1` — being the subject of a `hasValue` assertion — is inferred to be an `InformationObject`. SULO declares `Quality` and `InformationObject` to be disjoint kinds of `Feature`, so `ex:age_1` is simultaneously a `Quality` and an `InformationObject`: a contradiction the reasoner reports as an inconsistency.

Age should be modelled as a `sulo:Quantity` related to the person, where the value and its unit properly belong; the same correction applies to body temperature, blood pressure, and any other quantified observation that feels like a property but is in fact recorded information.

3.3. Mapping patterns as pivot bridges for cross-standard transformation

A further capability the mapping-pattern architecture is designed to support, though not yet evaluated end-to-end in the current tool, is cross-standard data transformation. Consider a FHIR Observation resource (`code: CodeableConcept`, `valueQuantity: Quantity`) and the corresponding OMOP MEASUREMENT table (`measurement_concept_id`, `value_as_number`). Both encode the same clinical observable but in structurally incompatible forms. A transformation pipeline FHIR → SULO → OMOP would require (a) a mapping from FHIR predicates to SULO path expressions, and (b) a mapping from SULO path expressions to OMOP predicates. If both mappings are expressed as Schema Builder mapping patterns pointing at the same SULO predicates, SULO can act as a pivot model, and the composition of the two pattern sets would in principle yield the cross-standard transformation as a SPARQL CONSTRUCT query.

This pivot approach is enabled by the mapping pattern compiler. The `buildReverseOwlExpr` function traverses the same triple-template list in the inverse direction, emitting `owl:inverseOf` restrictions. Given the SULO representation of a patient, the reverse function expresses "an individual is a SubjectOfCareRole if it is the isRoleOf-inverse-participant of a ClinicalVisit", precisely what is needed to query all visits associated with a given patient IRI.

3.4. Educational application

The Schema Builder also serves as a pedagogical instrument for demonstrating what formal ontologies add over lightweight schemas. Three comparison points are particularly effective:

- **Structural vs. semantic subsumption.** In the RDFS output, `:MeasurementProcess rdfs:subClassOf :MedicalProcedure` is a structural assertion. In the OWL+SULO export, both classes carry `owl:equivalentClass sulo:Process`, making their shared nature machine-inferrable. An OWL reasoner can deduce that a `MeasurementProcess` individual is a `sulo:Process` even without knowledge of the local class hierarchy.
- **Role indirection.** A schema designer's first instinct is `:ClinicalVisit :hasPatient :Person`. The OWL+SULO export adds the intermediate `SubjectOfCareRole` class, with the concrete motivation: if the patient was the primary responsible contact in visit A but merely an observer in visit B, an intermediate participation node is needed to represent each role distinctly.

Table 2
Feature comparison of schema and ontology design tools across five dimensions. Comparison criteria are defined as follows: Visual GUI — browser-based interactive editing without external drawing tools; ULO Alignment — ability to map schema elements to an upper-level ontology; OWL Restrictions — generation of owl:someValuesFrom-style property restriction chains, not merely class-level owl:equivalentClass assertions; SHACL — automatic shape generation from the same source model; Union Ranges — support for disjunctive (owl:unionOf/sh:or) range constraints. ✓ = fully supported; × = not supported.

Tool	Visual GUI	ULO Alignment	OWL Restrictions	SHACL Union Ranges
OWLGrEd [17]	✓	×	✓	×
LinkML [9]	× (YAML)	✓ (one-to-one)	✓ (basic)	beta
Chowlk [10]	✓ (draw.io)	×	✓	×
Schema Builder	✓	✓ (class & patterns)	✓ (chains)	✓

- **Constraint expressivity.** SHACL shapes validate that every Measurement has a `:hasCode` property pointing to an instance of `:Code` or `:ObservableEntity`. The OWL DL equivalence axioms additionally assert, via the `sulo:hasFeature` path, that any individual reached via that path from a `sulo:InformationObject` is a Code instance. A reasoner materializing the SULO graph can detect inter-dataset type inconsistencies invisible to SHACL.

A pre-built Clinical Health Record Schema (32 classes, 41 properties) is available via the *Load Example* button, enabling instructors to demonstrate the full export pipeline without students constructing a schema from scratch. A second *Load OMOP Example* option loads a parallel schema covering the same clinical domain in a different structural style, enabling side-by-side comparison.

3.5. Comparison with related tools

Table 2 compares Schema Builder with the most relevant related tools across five dimensions.

4. Limitations

4.1. Current maturity and scope

The Schema Builder is a research prototype rather than a production system: it has been exercised primarily on the clinical health record and OMOP example schemas described above, and has not yet been deployed or tested at the scale of a full institutional data model. The embedded HermiT-based consistency check operates at the TBox level only; instance-level (ABox) consistency checking requires exporting to Protégé or an equivalent external reasoner. Alignment discovery relies on the user knowing, or locating via autocomplete, the applicable SULO concept; no automated or LLM-assisted suggestion is currently provided, though this is planned.

4.2. Limitations of the mapping-pattern language

The triple-template UI supports only nested existential (i.e., 'some') restrictions; class expressions with boolean operators, and universal or cardinality restrictions are not supported. Only graph patterns are representable — SPARQL OPTIONAL, FILTER, and aggregation constructs are not. Concretely, a mapping cannot currently express a conditional pattern such as "use the role-bearer pattern only if the target individual is of a given subtype" or a cardinality-dependent branch such as "generate a different SULO path if the property is multi-valued versus single-valued." Such cases must currently be modelled as separate properties with distinct mapping patterns, rather than as a single conditional pattern, which can lead to a proliferation of near-duplicate properties in larger schemas.

5. Future Work

Although demonstrated here on a clinical schema, the tool's architecture is domain-independent: SULO's upper-level categories and the mapping-pattern compiler operate on schema structure rather than clinical vocabulary, and we expect the modeling-error patterns identified to recur in any domain where records are conflated with the entities they describe. Validating this generality on a non-clinical schema, alongside the planned extensions below, is left as future work. Planned extensions include a cross-schema mapping view for side-by-side FHIR–OMOP–SULO comparison, LLM-assisted concept alignment, and a library of reusable mapping pattern templates for common SULO design patterns (role bearer, quality bearer, information content entity).

6. Conclusion

The SULO-Compliant Schema Builder is an open-source web application that provides an integrated environment in which schema specifications, ontology alignments, validation artifacts, and OWL axioms are derived from a shared model. By embedding upper-ontology alignment and class- and property-level mapping patterns into the schema authoring workflow, the tool enables users to incrementally produce well-founded ontology artifacts through structured design decisions, rather than requiring manual OWL editing.

The tool makes three concrete contributions. First, the two-layer export architecture produces both a plain RDFS vocabulary and an OWL DL ontology with full `owl:equivalentClass` restrictions from the same schema model, with no OWL authoring required. Second, the mapping pattern compiler (`buildOwlExpr` / `buildReverseOwlExpr`) enables SULO to act as a pivot model across healthcare schemas, and other standards through shared pattern semantics. Third, the mutli-tab export view and pre-built example schemas provide a concrete pedagogical platform for demonstrating the expressive advantages that OWL DL provides over SHACL and RDFS alone.

Abbreviations

ABox: Assertion Box; BFO: Basic Formal Ontology; CDM: Common Data Model; FHIR: Fast Healthcare Interoperability Resources; IRI: Internationalized Resource Identifier; OMOP: Observational Medical Outcomes Partnership; OWL: Web Ontology Language; RDF: Resource Description Framework; RDFS: RDF Schema; SHACL: Shapes Constraint Language; SNOMED CT: Systematized Nomenclature of Medicine, Clinical Terms; SPARQL: SPARQL Protocol and RDF Query Language; SPHN: Swiss Personalized Health Network; SULO: Simplified Upper-Level Ontology; ULO: Upper-Level Ontology; UML: Unified Modeling Language.

Acknowledgments

The authors thank the AIDAVA project team for discussions on the SULO patterns, and the SPHN team for testing the tool.

Funding: This work was supported by the grant RYC2020-030190-I, in part by the Horizon Europe HORIZON-HLTH-2021-TOOL-06-03 under Grant 101057603, and in part by Horizon-HLTH-2022-Tool-12-Two-Stage under Grant 101080875 and the Horizon Europe Framework Program under Grant Agreement No. 101057062 (AIDAVA). The funders had no role in study design, data collection, or the decision to submit for publication.

Competing Interests: The authors declare no competing interests.

Declaration on Generative AI

Claude was used to assist in the development of the tool presented in this work and in the preparation of the manuscript. All AI-generated text and visuals were subsequently reviewed by the authors, who verified their accuracy and originality and take full responsibility for the content of this publication.

References

[1] HL7 International, FHIR: Fast healthcare interoperability resources, https://hl7.org/fhir/, 2019. Release 4.

[2] G. Hripcsak, J. D. Duke, N. H. Shah, C. G. Reich, V. Huser, M. J. Schuemie, M. A. Suchard, R. W. Park, I. C. K. Wong, P. R. Rijnbeek, J. van der Lei, N. Pratt, G. N. Norén, Y.-C. Li, P. E. Stang, D. Madigan, P. B. Ryan, Observational health data sciences and informatics (OHDSI): Opportunities for observational researchers, Studies in Health Technology and Informatics 216 (2015) 574–578. PMC4815923.

[3] F. Frexia, C. Mascia, L. Lianas, G. Delussu, A. Sulis, V. Meloni, M. Del Rio, G. Zanetti, openehr is fair-enabling by design, in: Public Health and Informatics, IOS Press, 2021, pp. 113–117.

[4] R. Arp, B. Smith, A. D. Spear, Building Ontologies with Basic Formal Ontology, MIT Press, 2015.

[5] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, L. Schneider, Sweetening ontologies with DOLCE, in: Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web. 13th International Conference, EKAW 2002, Sigüenza, Spain, October 1–4, 2002, Proceedings, volume 2473 of *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 2002, pp. 166–181. doi:10.1007/3-540-45810-7_18.

[6] M. Dumontier, R. Çelebi, K. Gilani, I. de Zegher, K. Serafimova, C. Martínez Costa, S. Schulz, SULO — a simplified upper-level ontology, in: Proceedings of the Joint Ontology Workshops (JOWO) — 15th International Conference on Formal Ontology in Information Systems (FOIS 2025), Catania, Italy, 2025.

[7] J. Euzenat, P. Shvaiko, Ontology Matching, 2 ed., Springer-Verlag, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-38721-0.

[8] C. Martínez-Costa, S. Schulz, Bridging the gap between ontologies and relational databases, Applied Ontology 11 (2016) 93–126.

[9] S. A. T. Moxon, H. Solbrig, N. L. Harris, P. Kalita, M. A. Miller, S. Patil, K. Schaper, C. Bizon, J. H. Caufield, S. Cirujano Cuesta, C. Cox, F. Dekervel, D. M. Dooley, W. D. Duncan, T. Fliss, S. Gehrke, A. S. L. Graefe, H. Hegde, A. J. Ireland, J. O. B. Jacobsen, M. Krishnamurthy, C. Kroll, D. Linke, R. Ly, N. Matentzoglu, J. A. Overton, J. L. Saunders, D. R. Unni, G. Vaidya, W.-M. A. M. Vierdag, LinkML Community Contributors, O. Ruebel, C. G. Chute, M. H. Brush, M. A. Haendel, C. J. Mungall, LinkML: an open data modeling framework, GigaScience 15 (2026) giaf152. doi:10.1093/gigascience/giaf152, published 21 January 2026. PMID: 41384800.

[10] S. Chávez-Feria, R. García-Castro, M. Poveda-Villalón, Chowlk: from UML-based ontology conceptualizations to OWL, in: The Semantic Web. 19th International Conference, ESWC 2022, Hersonissos, Crete, Greece, May 29 – June 2, 2022, Proceedings, volume 13261 of *Lecture Notes in Computer Science*, Springer, Cham, 2022, pp. 338–352. doi:10.1007/978-3-031-06981-9_20.

[11] OP-TED, model2owl: Transform UML XMI to formal OWL ontology and SHACL shapes, https://github.com/OP-TED/model2owl, 2023.

[12] A. Cimmino, A. Fernández-Izquierdo, R. García-Castro, Astrea: Automatic generation of SHACL shapes from ontologies, in: The Semantic Web. 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31 – June 4, 2020, Proceedings, volume 12123 of *Lecture Notes in Computer Science*, Springer, Cham, 2020, pp. 497–513. doi:10.1007/978-3-030-49461-2_29.

[13] M. A. Musen, The Protégé project: A look back and a look forward, AI Matters 1 (2015) 4–12. doi:10.1145/2757001.2757003.

[14] H. Bast, B. Buchhold, QLever: A query engine for efficient SPARQL+text search, in: Proceedings of

the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 6–10, 2017, ACM, 2017, pp. 647–656. doi:10.1145/3132847.3132921.

[15] R. C. Jackson, J. P. Balhoff, E. Douglass, N. L. Harris, C. J. Mungall, J. A. Overton, Robot: a tool for automating ontology workflows, BMC bioinformatics 20 (2019) 407.

[16] V. Touré, P. Krauss, K. Gnodtke, J. Buchhorn, D. Unni, P. Horki, J. L. Raisaro, K. Kalt, D. Teixeira, K. Crameri, S. Österle, FAIRification of health-related data using semantic web technologies in the swiss personalized health network, Scientific Data 10 (2023) 127. doi:10.1038/s41597-023-02028-y.

[17] J. Barzdins, G. Barzdins, K. Cerans, R. Liepins, A. Sprogis, OWLGrEd: a UML style graphical notation and editor for OWL 2, in: Proceedings of the 7th International Workshop on OWL: Experiences and Directions (OWLED 2010), 2010.