Schímatos: a SHACL-based Web-Form Generator for Knowledge Graph Editing

Jesse Wright[1[0000-0002-5771-988X]], Sergio J. Rodríguez Méndez[1[0000-0001-7203-8399]], Armin Haller[1[0000-0003-3425-0780]], Kerry Taylor[1[0000-0003-2447-1088]], and Pouya G. Omran[1[0000-0002-4473-3877]]

Australian National University, Canberra ACT 2601, AU
{firstname.lastname}@anu.edu.au
https://cecs.anu.edu.au/

Abstract. Knowledge graph creation and maintenance is difficult for naïve users. One barrier is the paucity of user friendly publishing tools that separate schema modeling from instance data creation. The Shapes Constraint Language (SHACL) [12], a W3C standard for validating RDF based knowledge graphs, can help. SHACL enables domain relevant structure, expressed as a set of shapes, to constrain knowledge graphs. This paper introduces *Schímatos*, a form based Web application with which users can create and edit RDF data constrained and validated by shapes. Forms themselves are generated from, and stored as, shapes. In addition, *Schímatos*, can be used to edit shapes, and hence forms. Thus, *Schímatos* enables end-users to create and edit complex graphs abstracted in an easy-to-use custom graphical user interface with validation procedures to mitigate the risk of errors. This paper presents the architecture of *Schímatos*, defines the algorithm that builds Web forms from shape graphs, and details the workflows for SHACL creation and data-entry. *Schímatos* is illustrated by application to Wikidata.

Keywords: Knowledge Graph · SHACL · MVC-based Web-Form Generator · RDF Editing Tool · Linked Data Platform

1 Introduction

The rapid growth of Knowledge Graphs (KGs) impels the Semantic Web vision [6] of a ubiquitous network of machine readable resources. Popular KGs include the community-driven Wikidata [29], and Google's KG [15] which is largely populated through `schema.org` annotations on websites. An enduring barrier to the development of the machine-readable Web, however, is the lack of tools for authoring semantic annotations [14, 3]. While ontology editors such as *Protégé* [16] and Diagrammatic Ontology Patterns [24] are favoured when creating quality assured (TBox) axioms, their primary purpose is to build and validate the model: the schema of an ontology. While some of these editors *can* also create instances, that is, individuals assigned to classes and data values with property relations (ABox), the process is cumbersome and requires detailed knowledge of the RDF(S) and OWL languages. There is limited support to guide users to the correct classes to which an entity can belong and the permissible relationships between entities. Further, as ontology editors do not clearly distinguish schema editing from data editing, data editing users may inadvertently alter the schema.

To address this, some Web publishing tools on top of wikis, microblogs or content management systems have been developed that allow a user to create semantic annotations, that is, instance assertions (e.g. the work discussed in [28, 13, 8, 17, 4]). This work has, for example, been incorporated into the semantic MediaWiki software [28] on which Wikidata is based [29]. However, even within this software, and so in Wikidata, there is limited user support for instance assertions, mostly through text auto-completion. Consequently, in order to add instances, users must have a strong understanding of the RDF data model, underlying semantics of the Wikidata ontology, and the typical structure of other instances of the same class.

The Shapes Constraint Language (SHACL) [12] is a recent W3C recommendation developed to express conditions, as shape graphs, for validating RDF based KGs. Thus domain relevant structure can be enforced. For example, Wikidata is an early adopter of shape graphs to define constraints on classes [25]. While Wikidata has chosen to use ShEx [20] to express these constraints[1], *Schímatos* uses the SHACL standard.

At the time of writing, 218 schemas exist under the Wikidata schema entity prefix, but none are enforced within the graph. For instance, the shape available at `https://www.wikidata.org/wiki/EntitySchema:E10` [31] defines constraints on entities of type *Human*, but instances of this type (e.g. `http://www.wikidata.org/entity/Q88056610` - Sergio José Rodríguez Méndez) are not validated against this shape. Moreover, the authoring tool underlying Wikidata does not use these shapes to guide the user when creating similar entities [25]. However, shapes *could* be used in authoring tools to guide semantic annotations.

This paper proposes *Schímatos*, a SHACL-based Web-Form Generator for Knowledge Graph Editing. Beyond its primary purpose of enabling naïve users to create and edit instance assertions, it also provides means to create and edit shapes. The software is being developed in the Australian Government Records Interoperability Framework (AGRIF) project and has been applied to basic use cases within several Australian Government departments. As these use cases are confidential, the authors cannot report on the knowledge graphs maintained in these cases, so we demonstrate a possible application of *Schímatos* on Wikidata.

The remainder of this paper is structured as follows. First, related systems are presented in section 2. Section 3 describes a motivating use case for this work using an example entity from Wikidata. Then, *Schímatos* and its architecture are presented in section 4, followed by the form display logic in section 5. Section 6 defines formally the execution behaviour of the system with respect to the data entry and the shape creation process. The paper concludes in section 7 with an outlook on the schedule for future work.

[1] See `https://www.wikidata.org/wiki/Wikidata:WikiProject_Schemas`

2 Related Work

Many mature ontology editors such as Protégé [16], the Neon toolkit [9] or the commercial TopBraid Composer[2], offer ways to create entities based on one or more ontologies. Some of these editors have a Web-based version that can allow ordinary Web users to hand craft ontology instances. However, users must be able to recognise and correctly encode the expected property relationships for each new class instance according to RDF(S)/OWL semantics.

Web-publishing tools on top of wikis, microblogs or content management systems [28, 13, 8, 17, 4] allow naïve users to create semantic annotations, that is, instance assertions. However, the user interface of these tools is either fixed to a particular ontology or it has to be manually created based on the desired mapping to a TBox schema.

ActiveRaUL [10, 11, 7] is closest in functionality to *Schímatos*. It allows the automatic rendering[3] of Web-forms from arbitrary ontologies. The resulting forms can then be used to create instances compliant to the original ontology. The forms themselves become instances of a UI ontology, called RaUL. The major difference to *Schímatos* is that ActiveRaUL predated the introduction of SHACL and, as such, the resulting forms are generated by interpreting ontology assertions as rules. *Schímatos* is using SHACL shapes and therefore does not need to violate the Open World assumption of an underlying ontology, but truthfully renders a form, based on the constraints expressed.

ShEx-form [19] is a recent tool that creates forms from a shape graph. It appears to be inspired by Tim Berners-Lee's draft article, *Linked Data Shapes, Forms and Footprints* [26], which proposes the joint use of Shape Languages, Form Languages (such as the User Interface Ontology [5]), and Footprint Languages which describe where the data from a form is to be stored. To the best of our knowledge, this tool does not enable users to interact directly with external KGs and also does not perform any validation of user input. Furthermore, some features may not suit users unfamiliar with RDF concepts; requiring that they interact directly with the underlying shape serialization in order to generate a form.

3 Motivating Example

Within Wikidata [29] there are many instances of the class *Human*, most of which have missing attributes that are required according to the Wikidata *Human* shape. Of the existing instances of type human, there are only 8,117,293[4] entities which constitute about 0.103% of the current world population [27, Figure A.1.] and 0.007% [18, Table 2] of the total population over time. Whilst there have

[2] See `https://www.topquadrant.com/products/topbraid-composer/`

[3] The term 'rendering' is used to refer to the generation of a Document Object Model (DOM). This is the same terminology used in the ReactJS framework.

[4] As of 2020-08-19 using the SPARQL query `SELECT (COUNT(?item) AS ?count) WHERE {?item wdt:P31 wd:Q5}`

been attempts to scrape data on existing entities [32], much of the information is either not available online, or not in a machine-readable format. Thus, to effectively complete instances of the class *Human*, widely-accessible tools that enforce logical constraints on the class (i.e. *Human* in this case) are required, whilst also prompting users to enter requisite data for the system. An example is the entity `wd:Q88056610`[5] which appears to have been automatically generated from the ORCID of the researcher [32]. Currently, no other information about this entity is available on Wikidata.

This paper uses the shape graph for *Human* [31] as a running example, and shows how *Schímatos* can be used to enforce constraints on instances of this class. Specifically, the paper demonstrates how the tool can be used to fill out missing information; including gender, birthplace and date of birth; for the entity `wd:Q88056610`[6].

4 The *Schímatos* System

Schímatos is an application that automatically generates Web-forms from SHACL shapes. Data from completed forms, including the class and datatype annotations of inputs, can be submitted to a KG over a SPARQL endpoint registered with *Schímatos*. Web-forms may also be edited within the tool and their SHACL definition updated via SPARQL updates. All operations occur in the client so the tool can be packaged as a stand-alone desktop application or served from a website.

Schímatos is built in the ReactJS Framework[7] which compiles to W3C compliant HTML+CSS/JavaScript with cross-browser compatibility. All SPARQL requests are compliant with the LDP standard [23], so that it can read/write data in SPARQL compliant triplestores. The tool also accepts data in the proposed SPARQL 1.1 Subscribe Language [1] so as to receive live updates from SEPA clients [2].

Schímatos is available to download as an HTML+CSS/JavaScript package[8] and it can also be run online[9]. Both of these resources work by default with a local instance of Wikidata and re-use existing ShEx files [30] translated to SHACL files for *Schímatos*. Consider the entity `wd:Q88056610`, which appears to have been automatically generated from the ORCID of the researcher [32]. Currently, no other information about this entity is available on Wikidata. One can apply the SHACL-translated *Human* constraint [31] to this entity in order to create a form that prompts users to fill out the missing information including gender, birthplace and date of birth. Many additional examples are available

[5] This is the identifier for researcher Dr. Sergio José Rodríguez Méndez.

[6] The repo for *Schímatos* at `http://schimatos.github.io` includes a translation of the *Human* ShEx shape into a SHACL shape which can be obtained by automated means with tools such as RDFShape [21].

[7] See `http://reactjs.org`

[8] http://schimatos.org

[9] http://schimatos.github.io

```
@prefix ex: <http://example.com/ns#> .
@prefix sh: <http://www.w3.org/ns/SHACL#> .
@prefix wd: <http://www.wikidata.org/entity/> .
@prefix wdt: <http://www.wikidata.org/prop/direct/> .
@prefix tp: <http://www.shacl.kg/types/> .


ex:humanWikidataShape
  a sh:NodeShape ;
  sh:targetClass wd:Q5 ;
  rdfs:label "human shape" ;
  sh:property [
    sh:path wdt:P21 ;
    sh:name "gender" ;
    sh:in (
      wd:Q6581097 # male
      wd:Q6581072 # female
      wd:Q1097630 # intersex
      wd:Q1052281 # transgender female (MTF)
      wd:Q2449503 # transgender male (FTM)
      wd:Q48270 # non-binary
    );
    sh:maxCount 1 ;
  ] ;
  sh:property [
    sh:path wdt:P19 ;
    sh:name "place of birth" ;
    sh:maxCount 1 ;
    sh:property [
      sh:path wdt:P17 ;
      sh:name "country" ;
      sh:maxCount 1 ;
    ] ;
  ] ;
  sh:property [
    sh:path wdt:P569 ;
    sh:name "date of birth" ;
    sh:maxCount 1 ;
    sh:pattern "^[0-9]{2}\/[0-9]{2}\/[0-9]{4}$" ;
  ] ;
  sh:property [
    sh:path wdt:P735 ;
    sh:name "given name" ;
    sh:minCount 0 ;
    sh:datatype tp:name
  ] ;
  sh:property [
    sh:path wdt:P734;
    sh:name "family name" ;
    sh:minCount 0 ;
  ] ;
  sh:property [
    sh:path wdt:P106;
    sh:name "occupation" ;
    sh:minCount 0 ;
  ] ;
  sh:property [
    sh:path wdt:P27;
    sh:name "country of citizenship" ;
    sh:minCount 0 ;
  ] ;
  sh:property [
    sh:path wdt:P22 ;
    sh:name "father" ;
    sh:minCount 0 ;
    sh:class wd:Q5 ;
    sh:node ex:humanWikidataShape ;
  ] ;
  sh:property [
    sh:path wdt:P25 ;
    sh:name "mother" ;
    sh:minCount 0 ;
    sh:class wd:Q5 ;
    sh:node ex:humanWikidataShape ;
  ] ;
  sh:property [
    sh:path wdt:P3373 ;
    sh:name "sibling" ;
    sh:minCount 0 ;
    sh:class wd:Q5 ;
    sh:node ex:humanWikidataShape ;
  ] ;
  sh:property [
     sh:path wdt:P26 ;
     sh:name "husband|wife" ;
     sh:minCount 0 ;
     sh:node ex:humanWikidataShape ;
  ] ;
  sh:property [
    sh:path wdt:P40;
    sh:name "children" ;
    sh:minCount 0 ;
    sh:class wd:Q5 ;
  ] ;
  sh:property [
    sh:path [
      sh:alternativePath (
        wdt:P1038 [ # relatives
          sh:oneOrMorePath [
            sh:alternativePath (
              wdt:P22
              wdt:P25
              wdt:P3373
              wdt:P26
            )
          ] ;
        ] ;
      )
    ] ;
    sh:name "relatives" ;
    sh:minCount 0 ;
    sh:class wd:Q5 ;
    sh:node ex:humanWikidataShape ;
  ] ;
  sh:property [
    sh:path wdt:P103 ;
    sh:name "native language" ;
    sh:minCount 0 ;
  ] ;
  sh:property [
    sh:path wdt:P1412 ;
    sh:name "written/spoken language(s)"
    sh:minCount 0 ;
  ] ;
  sh:property [
    sh:path wdt:P6886 ;
    sh:name "publishing language(s)" ;
    sh:minCount 0 ;
  ] .
```

Fig. 1: A SHACL shape for the class of human (`wd:Q5`) entities described in the Wikidata ontology.

as translations of the pre-defined ShEx constraints publicly available on the Wikidata platform[10].

4.1 Architecture

At its core, *Schímatos*' design follows the Model-View-Controller (MVC) pattern [22] as depicted in Figure 2.

Model The model layer consists of three logical named graphs[11]: (1) SHACL store (shapes), (2) Type store[12], and (3) the KG (data). The SHACL store is a repository of shapes which, when loaded, are translated to form structures by the

[10] See `https://www.wikidata.org/wiki/Wikidata:WikiProject_Schemas`

[11] There is no requirement that these graphs share the same SPARQL endpoint or host.

[12] The current namespace prefix for this graph is `http://www.shacl.kg/types/` (`tp:`)

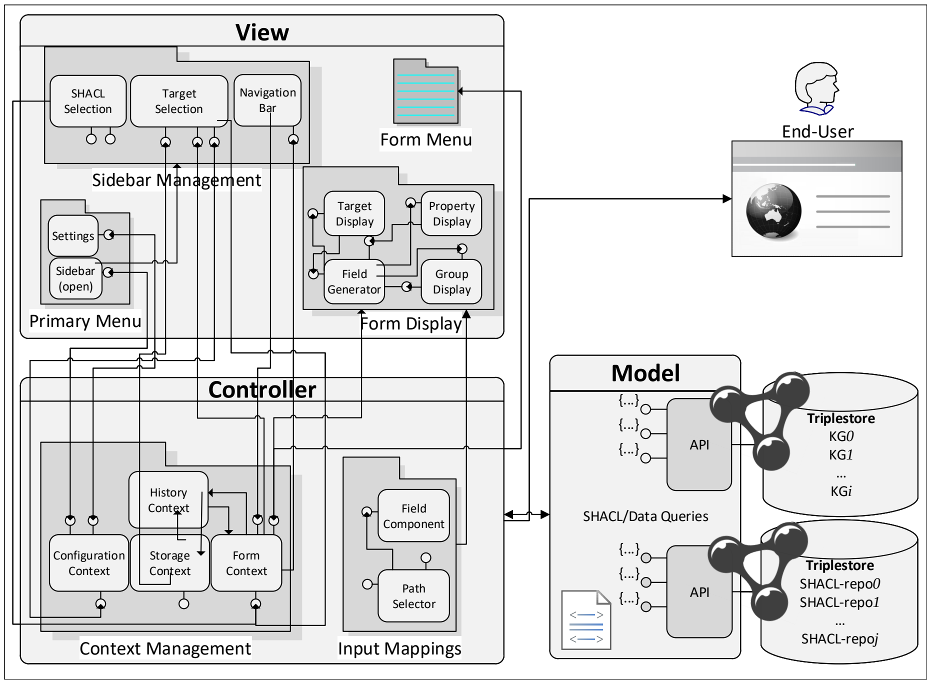


Fig. 2: Model-View-Controller software architectural design pattern for *Schímatos*

View. The Type store contains constraints for custom class and datatype definitions, such as regular expressions that all literals of a given datatype must match. Examples include: phone numbers, email addresses, and social security numbers. These definitions provide validation constraints for any `sh:PropertyShape` with a defined class or datatype. The KG, of course, has the data, which is used to pre-fill the form and is updated whenever a user submits new data.

Schímatos interacts with the model via RESTful API requests. In the current version of the tool, these requests are SPARQL queries and commands made over the SPARQL endpoint of the relevant triplestores.

View One of the main parts of the view layer in Figure 2 is the *Form Display* module; which has the components that handle the Web-Form generation from the SHACL structures. This module is explained in detail in section 5. The Sidebar Management module wraps several key components, including the SHACL and Target Selectors, the Navigation Bar, and the SHACL Uploader.

SHACL Selection Within this component, users can choose one or more constraints they wish to apply to given entities. Shapes can be searched for by properties including `rdfs:label`, `sh:targetClass`, and `sh:targetNode` constructors. Experienced users have the option to write custom SPARQL queries with which they can search for SHACL constraints. Users also have the option to customise many of their selections, including the 'severity' of the constraints they wish to apply and whether they wish to apply several constraints to a single entity. There is also functionality to 'automatically apply' constraints where the target `class/entity/subjectsOf/objectsOf` attributes of SHACL constraints are used to determine whether the constraint applies to the entity a user is targeting.

Target Selection Within this panel, users can search for entities that they wish to enter into the *Schímatos* form. Similarly to the *SHACL Selection* Component, shapes can be selected using a custom SPARQL query or by querying for properties such as `rdf:type` and `rdfs:label`. This part of the GUI also displays the datatype and object properties of the currently selected entity so that users can verify that they have selected the correct IRI.

Navigation Bar Within the *Navigation Bar*, users may choose to navigate to, and focus on different components of the form. This is primarily of use when entering data about complex entities for which the form structure may be too overwhelming to view in a single display.

4.2 Controller

Input Mappings All data entry in *Schímatos* is validated against the shape constraint used to generate the form. The current validation engine for individual inputs is built upon the React Hook Forms package[13]. For validating individual form fields, *Schímatos* uses `sh:PropertyShape` constructors such as `sh:pattern` and `sh:minValue` to validate entry of entities and literals. Each `sh:PropertyShape`, such as that for a human's given name, may have a datatype or class (such as `tp:name`) with an associated set of constraints. If not defined directly in the `sh:PropertyShape`, *Schímatos* can suggest the datatype using the `rdfs:domain` and `rdfs:range` of the predicate (such as `wdt:P735`). These constraints can be defined in a shape and loaded automatically from a designated Type store. For this purpose, SHACL definitions of all pre-defined XSD types are pre-loaded into the *Storage Context*. The controller does not update the *Form Context* with values that fail the validation procedure and alerts users to fix the entry. This means that when a completed form is submitted to the model, no 'invalid' entries are submitted to the KG.

Context Management *Schímatos* uses three ReactJS contexts[14] as described below, which control and manage the application's state at any given time.

1. The *Configuration Context* defines user settings for interacting with the tool such as the complexity of features available and the prefixes used within the

[13] See `https://react-hook-form.com/`
[14] See `https://reactjs.org/docs/context.html`

form. In addition, the *Configuration Context* defines the KG, SHACL store and Type store endpoints. If permitted by the user, this is also stored as a browser cookie to maintain settings across sessions.
2. The *Form Context* captures the information to construct the form, including a representation of the SHACL constraints currently in use. It additionally stores data for all the entities and literals currently entered into the form, the data of children nodes within the supported path length, and the display settings for any *form element*[15]. When multiple shape constraints are applied simultaneously, they are 'merged' within the *Form Context* by applying the most strict set of constraints on each property.
3. The *Storage Context* has a local copy of triples from the KG model which are relevant to the current form. This data can then be used to pre-fill the form, provide suggestions for possible user input, and enable the display of datatype and object properties. Additionally, the *Storage Context* contains datatype constraints which are automatically applied to any property fields that have 'datatype' information in the corresponding SHACL pattern.

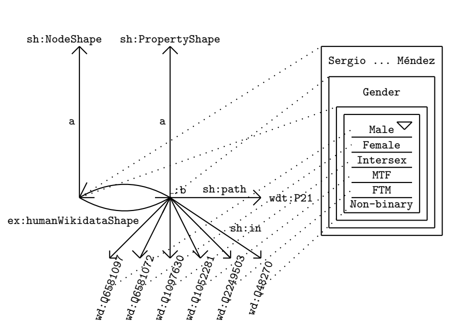
5 Form Display Logic

The SHACL standard defines two `sh:Shape` classes which are mapped in *Schímatos* to the DOM as follows: `sh:NodeShape`s are mapped to form elements, and `sh:PropertyShape`s are mapped to form fields that include HTML/JavaScript validators and a label. The following paragraphs provide more detail on this rendering logic.

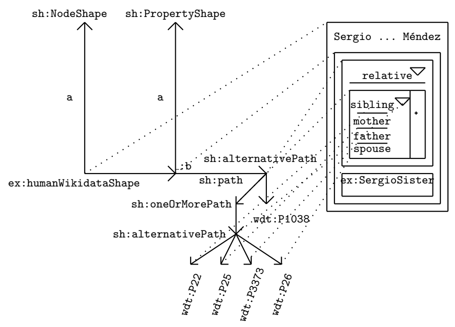
Rendering simple shapes A `sh:NodeShape` (e.g. `ex:humanWikidataShape`) is a set of `sh:PropertyShapes` (e.g. shapes for `wdt:P1038 -` *relatives*, or `wdt:P21 -` *gender*) used to generate a form for a chosen focus node (e.g. `wd:Q88056610` - Sergio José Rodríguez Méndez). When rendering a form, *Schímatos* uses the `sh:order` constructor of each `sh:PropertyShape` to determine the position in which it is displayed whilst `sh:PropertyShape`s with the same `sh:group` constructor are grouped in a pane.

Each `sh:PropertyShape` is rendered as a set of one or more HTML inputs that have the same validators, and the values of which follow the same property path to the focus node. A single label is used for each set of inputs. If defined, it is the value of the `sh:name` constructor. Otherwise, if the `sh:path` is a single predicate of length 1 (for instance, `wdt:P1038` rather than `wdt:P22/wdt:P1038`*) then the `rdfs:label` of that predicate is displayed. If `rdfs:label` is undefined, the property path is used as the label. Each set is displayed and validated uniquely depending on the constraints specified in the shape. If the `sh:nodeKind` constraint is `sh:IRI` (e.g. as for `wdt:P1038 -` *relatives*, in Figure 1), then the input must be a valid W3C IRI and the tool presents users with a customised IRI-field

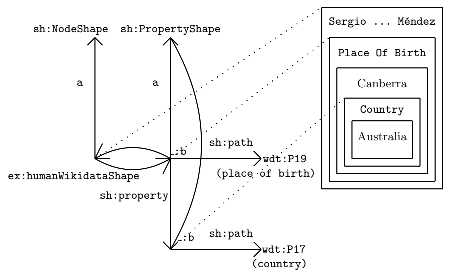
[15] see section 5



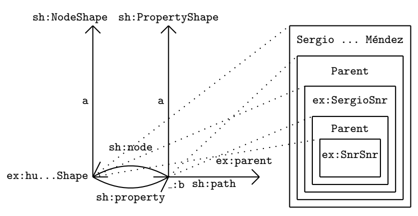
(a) Displaying `sh:in` property constraint



(b) Displaying complex property paths

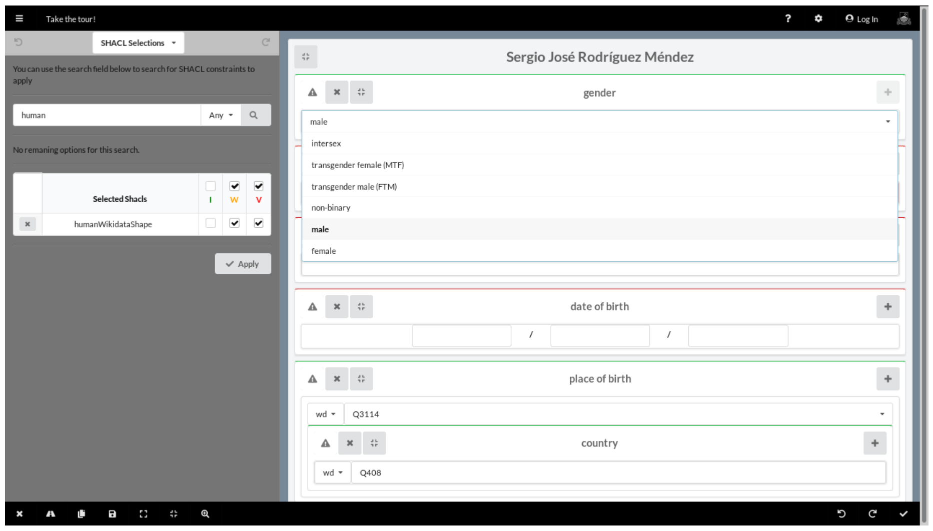


(c) Displaying nested-shapes



(d) Displaying recursive shapes

Fig. 3: Rendering of different property constraints in a generated form



You can use the search field below to search for SHACL constraints to apply

human

No remaning options for this search.

	Selected Shacls	I	W	V
×	humanWikidataShape	☐	✓	✓

Sergio José Rodríguez Méndez

gender

male

intersex
transgender female (MTF)
transgender male (FTM)
non-binary
male
female

date of birth

/ /

place of birth

wd Q3114

country

wd Q408

Fig. 4: Screenshot of the form generated by *Schímatos* from the *Human* shape

XML/XSD type	HTML/ReactJS input type
anyURI | string | hexBinary	text
decimal | double | float | gYear | duration	number
gDay | gMonth	drop-down
boolean	checkbox
date	date
dateTime	datetime-local

Table 1: XSD/XML to HTML/ReactJS mapping

where they can use a drop-down input to select the correct namespace prefix before entering the remaining IRI. Users are also presented with suggested inputs, such as known entities of the class, which they can select as the input.

If the `sh:in` constraint is present for a property (e.g. as for `wdt:P21` - *gender*) then the input value must lie within the set of values predefined in the SHACL constraint (e.g. male, female, transgender). The tool displays this set as an HTML drop-down list (cf. Figure 3a) from which a user can select the correct value.

When the `sh:pattern` constraint is present (e.g. as for `wdt:P569` - *date of birth*), the input must satisfy a regular expression. The expression is then broken into separate characters and capturing groups and the input is displayed as a series of inputs for each variable group. For instance, the input for a date of birth would be displayed as 3 separate numeric inputs with slashes in between them. All remaining inputs with a specified datatype constraint are mapped to standard HTML inputs as given in table 1. The number of inputs that are displayed to the user in the form field is initially determined by the `sh:minCount` constructor for the `sh:PropertyShape`. Users can add or remove such inputs so long as they remain within the `sh:minCount` and `sh:maxCount` constraint for a property with `sh:severity` of `sh:Violation`, but are by default unbounded for other levels of severity. These inputs are equipped with a set of JavaScript validators for each property constraint; the value is not saved into the form until it passes the validation criteria.

Rendering property shapes with complex paths Complex `sh:path` constructors in `sh:PropertyShape`s are displayed as a variable label for the set of inputs. Rather than displaying the `sh:name`, `rdfs:label` or property path IRI as the label, users are presented with a menu from which they can select the path they wish to use to add values to a given property constraint. For instance, the complex property path shape in Figure 3b shows the `ex:relatives` property of the `ex:humanWikidataShape`. For the label, there is a drop-down where users can select the option `wdt:P1038` (for a generic relative) and another option to create a custom path using the *sibling*, *mother*, *father*, and *spouse* properties. For instance, one could enter information about their *grandfather* using the path `wdt:P22/wdt:P22` (*father*/*father*). Values will be entered into the graph corresponding to the value of the drop-down label at the time of entry. This means, a user could enter her grandfather's name, and then change the path to `wdt:P22/wdt:P25` to enter information about her grandmother.

Rendering Nested Shapes Nested Shapes are rendered as nested form elements within the DOM. To do this, *Schímatos* first renders the form disregarding any `sh:node` constructors or nested properties present within a `sh:PropertyShape`. For each input generated by a given `sh:PropertyShape`, the nested `sh:NodeShape` is applied, treating the input as the focus node for the new form element. In our example, *Schímatos* will first generate a form beginning at `ex:humanWikidataShape` that has a form field for `wdt:P19` (*place of birth*). Once the user inputs the IRI denoting place of birth, it becomes the focus node of a nested `sh:NodeShape` which is a form element containing a single form field for `wdt:P17` (*country*). A sample rendering of this form is given in Figure 3c.

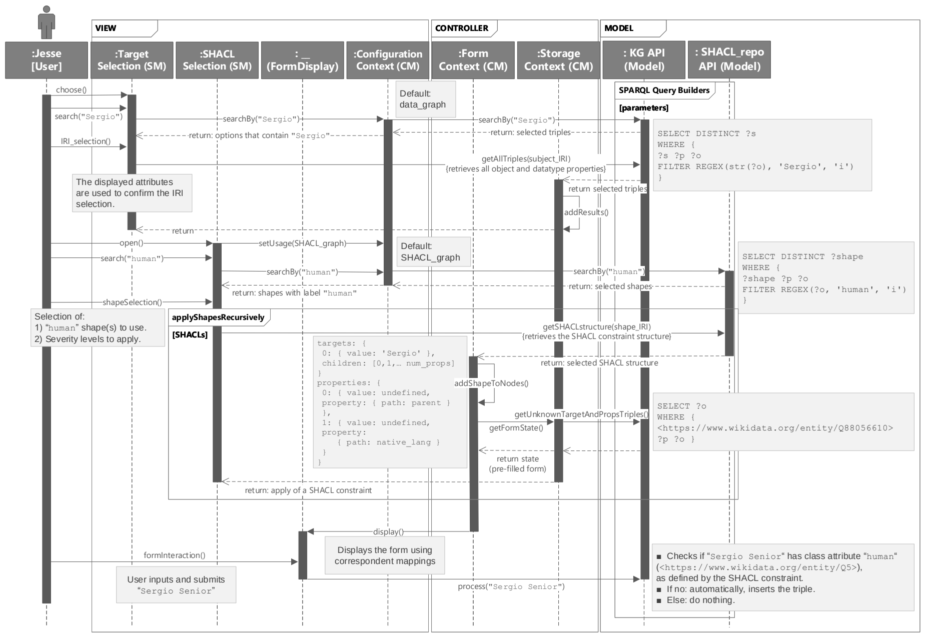
Rendering Recursive Shapes The SHACL standard specifies that 'validation with recursive shapes is not defined in SHACL and is left to SHACL processor implementations' [12]. In *Schímatos*, this is represented as a nested form structure that responds dynamically to user entry. Recursively defined shapes cannot be implemented in the same manner as nested shapes as doing so would cause a non-terminating loop within the application. Consequently, *Schímatos* loads and stores recursively defined data in the *Storage Context*. In our example, *Schímatos* first loads the `ex:humanWikidataShape` and renders all form fields within the form element. Since the `sh:PropertyShape` generating the form field `wdt:P22` (*father*) contains a recursive reference to the `ex:humanWikidataShape`, the rendering of this element terminates and a copy of `ex:humanWikidataShape` is saved to the *Storage Context*. Once a user submits a value in the form field for `wdt:P22` (e.g. `wd:SergioSenior`), or when the value is pre-filled using data from the KG, *Schímatos* validates the node `wd:SergioSenior` against the `ex:humanWikidataShape`. If the validation passes then the form remains unchanged, otherwise, a new `ex:humanWikidataShape` form element is rendered with `wd:SergioSenior` as the focus node. The process repeats, as the form element for `wd:SergioSenior` contains a `wdt:P22` form field which references the `ex:humanWikidataShape`. Users may enter the IRI of `wd:Q88056610`'s *grandfather* (`ex:SnrSnr`) in this field (cf. Figure 3d). Users can terminate this process at any point by closing any form element they do not wish to complete.

6 Execution Behavior

The execution behaviour of *Schímatos* can be grouped into two processes, i.e. a data entry process (cf. Figure 5), and a shape creation process (cf. Figure 6), that may be executed in an interleaving manner. However, as the intended user of each of these processes is typically distinct, i.e. domain experts that create data and information architects that can change shape constraints, respectively, this paper distinguishes these two processes in the following sections.

6.1 Data Entry

Users may enter data about either a new or an existing entity. In both cases, a user names the entity and selects the shape they wish to use for the process which in turn generates a form for the user to fill in. If there is existing data relating to the named entity, the form will be automatically initialised with information retrieved from the KG. Figure 5 presents a UML sequence diagram for the case where a user wishes to add new data about an existing entity `wd:Q88056610` - Sergio José Rodríguez Méndez. Figure 4 depicts a screen capture of this process.



```
SELECT DISTINCT ?s
WHERE {
?s ?p ?o
FILTER REGEX(str(?o), 'Sergio', 'i')
}
```

```
SELECT DISTINCT ?shape
WHERE {
?shape ?p ?o
FILTER REGEX(?o, 'human', 'i')
}
```

```
targets: {
 0: { value: 'Sergio' },
 children: [0,1,… num_props]
}
properties: {
 0: { value: undefined,
 property: { path: parent }
 },
 1: { value: undefined,
 property:
    { path: native_lang }
 }
}
```

```
SELECT ?o
WHERE {
<https://www.wikidata.org/entity/Q88056610>
?p ?o }
```

- Checks if "`Sergio Senior`" has class attribute "`human`" (`<https://www.wikidata.org/entity/Q5>`), as defined by the SHACL constraint.
- If no: automatically, inserts the triple.
- Else: do nothing.

Fig. 5: UML sequence diagram view of *Schímatos* for data creation

To do so, a user would first navigate to the *Target Selection* panel within the sidebar. There is a search bar within the panel where users can search by any datatype (literal) constructor relating to an entity. Since the `rdfs:label` constructor for the entity is "Sergio José Rodríguez Méndez", the entity will appear in the set of results when a user searches for "Sergio". Once a user selects an entity from the drop-down menu, it is entered into the empty field on the screen. Next users can select which shape (form) they wish to apply to the entity. To do this, they can first navigate to the *SHACL Selection* panel within the sidebar, since the *Human* shape which is to be applied has an `rdfs:label` "human shape for wikidata instances" (cf. Figure 1), it will be in the set of results for the search term 'human'. Once a user applies the shape to the entity, a form will be generated via the procedure outlined in section 5. Once the form is rendered, users can choose to add or remove repetitive form components as long as it is allowed by the SHACL constraint underlying that form component. For example, an individual can have between 0 and 2 living biological parents, so the user may choose to add another field under *parent* and enter the names of Sergio's mother *and* Sergio's father. Each user entry is subject to a validation process (based on regular expression patterns, value constraints, etc. as discussed in section 4.2). Users are not able to submit their data for this entry if the validation process fails: for instance, if a user attempts to enter a date of birth, without including the year. When this occurs, a popup will prompt the user to correct the entry. Once the user has 'completed' the form, they can choose to perform a 'final submission' which performs additional validations (including validating certain relationships between nodes, and cardinality of elements). Users will be guided to fix any errors if this validation process fails. If the validation process passes, all changes will be submitted to the KG.

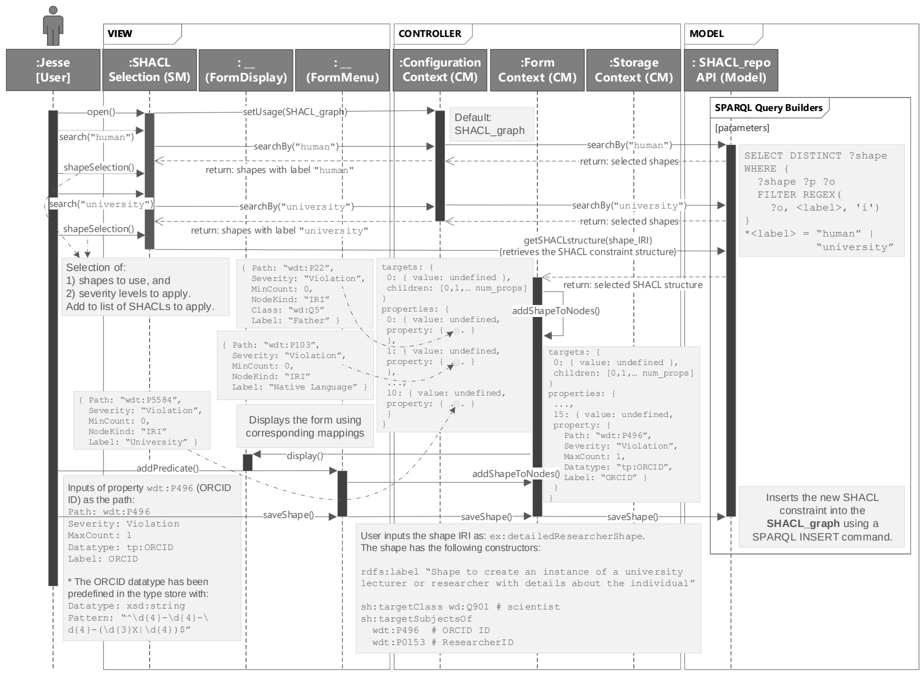
6.2 Shape Creation

Schímatos provides the capacity for information architects to construct SHACL shapes within a form building UI. Figure 6 presents the UML sequence diagram for creating a new *detailed researcher shape*[16]. In this example, there is an existing *Human* shape and a *University professor* shape defined for Wikidata instances. These shapes contain many of the attributes that the user wishes to include in a new shape, say *detailed researcher*. The user can search for the *Human* shape and the *University professor* shape before opting to apply both shapes simultaneously. The user has the option to apply all constraints, or only those with a defined level of violation severity. A severity may be `sh:Info`, `sh:Warning` or `sh:Violation` with an undefined severity defaulting to a `sh:Violation`. These shapes will then be merged within the *Form Context* by the processes outlined in section 4.2. This is displayed to the user as a single form which contains all of the constructors and constraints from both shapes. The user may then manually edit/create form fields using the form building tools. In this use case, the user will add the `wdt:P496` constructor (ORCID). The `sh:maxCount` constraint along with the pattern of the datatype are used to validate that the user entry follows the correct pattern for at most one ORCID. Once the user submits this change, the *Form Context* is updated accordingly. When a user chooses to 'save' a shape back to the SHACL graph, the internal structures storing the shape in the *Form Context* is first serialized to Turtle and then the data is submitted to the SHACL triplestore over the SPARQL endpoint via an INSERT command.

7 Conclusion and Future Work

Schímatos is the first interactive SHACL informed knowledge graph editor. It can be used for knowledge graph completion by domain experts without expertise in

[16] This example is described in further detail at `https://github.com/schimatos/schimatos.org`



```
SELECT DISTINCT ?shape
WHERE {
  ?shape ?p ?o
  FILTER REGEX(
    ?o, <label>, 'i')
}
*<label> = "human" |
           "university"
```

Selection of:
1) shapes to use, and
2) severity levels to apply.
Add to list of SHACLs to apply.

```
{ Path: "wdt:P22",
  Severity: "Violation",
  MinCount: 0,
  NodeKind: "IRI"
  Class: "wd:Q5"
  Label: "Father" }
```

```
{ Path: "wdt:P103",
  Severity: "Violation",
  MinCount: 0,
  NodeKind: "IRI"
  Label: "Native Language" }
```

```
{ Path: "wdt:P5584",
  Severity: "Violation",
  MinCount: 0,
  NodeKind: "IRI"
  Label: "University" }
```

```
targets: {
 0: { value: undefined },
 children: [0,1,… num_props]
}
properties: {
 0: { value: undefined,
 property: { ... }
 },
 1: { value: undefined,
 property: { ... }
 },
 ...,
 10: { value: undefined,
 property: { ... }
 }
}
```

Displays the form using corresponding mappings

```
targets: {
 0: { value: undefined },
 children: [0,1,… num_props]
}
properties: {
 ...,
 15: { value: undefined,
 property: {
   Path: "wdt:P496",
   Severity: "Violation",
   MaxCount: 1,
   Datatype: "tp:ORCID",
   Label: "ORCID" }
 }
}
```

Inputs of property `wdt:P496` (ORCID ID) as the path:

```
Path: wdt:P496
Severity: Violation
MaxCount: 1
Datatype: tp:ORCID
Label: ORCID
```

* The ORCID datatype has been predefined in the type store with:

```
Datatype: xsd:string
Pattern: "^\d{4}-\d{4}-\
d{4}-(\d{3}X|\d{4})$"
```

User inputs the shape IRI as: `ex:detailedResearcherShape`.
The shape has the following constructors:

```
rdfs:label "Shape to create an instance of a university
lecturer or researcher with details about the individual"

sh:targetClass wd:Q901 # scientist
sh:targetSubjectsOf
  wdt:P496  # ORCID ID
  wdt:P0153 # ResearcherID
```

Inserts the new SHACL constraint into the **SHACL_graph** using a SPARQL INSERT command.

Fig. 6: UML Sequence view of *Schímatos* for SHACL creation

RDF(S)/OWL as well as for the development of SHACL shapes by information architects. This paper has shown how *Schímatos* can dynamically transform SHACL shapes to HTML data-entry forms with built-in data validation.

Schímatos is available under the MIT license[17] for download at `http://schimatos.org` and for Web use at `http://schimatos.github.io`. The first public release of the software is also available under the DOI `https://doi.org/10.5281/zenodo.3988748`.

The authors expect that the software will be maintained in the long term by the Australian Government Linked Data Working Group[18] and the Open Source community at large. We would like to invite the Wikidata project to gauge the potential to use the tool in the continuous creation of this public knowledge graph.

In future work, the authors have planned formal user trials in the Australian Government. Currently, *Schímatos* is being improved in the following ways: (1) multi-user support; (2) a Model RESTful service that handles different graph versions (with DELETE operations); (3) support of SHACL generation from TBox axioms; and (4) RDF* support.

Schímatos should be cited as follows:

[17] See `https://opensource.org/licenses/MIT`
[18] See `http://linked.data.gov.au`

Wright, J., Rodríguez Méndez, S.J., Haller, A., Taylor, K., Omran, P.G.: *Schímatos* - A SHACL-based Web-Form Generator (2020), DOI `https://doi.org/10.5281/zenodo.3988748`.

References

1. Aguzzi, C., Antoniazzi, F., Roffia, L., Viola, F.: SPARQL 1.1 subscribe language. Unofficial Draft 12 October, w3c (2018), `http://mml.arces.unibo.it/TR/sparql11-subscribe.html`
2. Aguzzi, C., Antoniazzi, F., Roffia, L., Viola, F.: SPARQL event processing architecture (SEPA). Unofficial Draft, W3C (Oct 2018), `http://mml.arces.unibo.it/TR/sepa.html`
3. Barbosa, A., Bittencourt, I.I., Siqueira, S.W., Silva, R.D., Calado, I.: The use of software tools in linked data publication and consumption: A systematic literature review. IJSWIS **13**, 68–88 (2017)
4. Baumeister, J., Reutelshoefer, J., Puppe, F.: KnowWE: a Semantic Wiki for knowledge engineering. Applied Intelligence **35**, 323–344 (2011)
5. Berners-Lee, T.: User interface ontology. Tech. rep., w3c (aug 2010), `https://www.w3.org/ns/ui`
6. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
7. Butt, A.S., Haller, A., Liu, S., Xie, L.: ActiveRaUL: A web form-based user interface to create and maintain RDF data. In: Proceedings of the ISWC 2013 Posters & Demonstrations Track. vol. 1035, pp. 117–120. CEUR-WS.org (2013)
8. Corlosquet, S., Delbru, R., Clark, T., Polleres, A., Decker, S.: Produce and consume linked data with Drupal! In: Proceedings of the International Semantic Web Conference (ISWC 2009). pp. 763–778. Springer-Verlag (2009)
9. Haase, P., Lewen, H., Studer, R., Tran, D.T., Erdmann, M., d'Aquin, M., Motta, E.: The NeOn ontology engineering toolkit. In: Proceedings of the WWW 2008 Developers Track (2008)
10. Haller, A.: Activeraul: A model-view-controller approach for semantic web applications. In: Proceedings of the International Conference on Service-Oriented Computing and Applications (SOCA). pp. 1–8. IEEE (2010)
11. Haller, A., Groza, T., Rosenberg, F.: Interacting with linked data via semantically annotated widgets. In: Proceedings of the Joint International Semantic Technology Conference, JIST. LNCS, vol. 7185, pp. 300–317. Springer (2011)
12. Knublauch, H., Kontokostas, D.: Shapes constraint language (SHACL). W3C Recommendation, w3c (July 2017), `https://www.w3.org/TR/shacl`
13. Kuhn, T.: AceWiki: Collaborative Ontology Management in Controlled Natural Language. In: Proceedings of the 3rd Semantic Wiki Workshop (SemWiki 2008), in conjunction with ESWC 2008 (2008)
14. Liao, X., Zhao, Z.: Unsupervised approaches for textual semantic annotation, a survey. ACM Comput. Surv. **52**(4) (Aug 2019)
15. Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., Taylor, J.: Industry-scale knowledge graphs: Lessons and challenges. ACM Queue **17**(2) (2019)
16. Noy, N., Sintek, M., Decker, S., Crubezy, M., Fergerson, R., Musen, M.: Creating Semantic Web contents with Protege-2000. Intelligent Systems, IEEE **16**(2), 60–71 (2001)

17. Passant, A., Breslin, J.G., Decker, S.: Open, distributed and semantic microblogging with SMOB. In: Proceedings of the 10th International Conference on Web Engineering (ICWE 2010). pp. 494–497. Springer-Verlag (2010)
18. Population Reference Bureau: How many people have ever lived on earth? (Aug 2018), `https://www.prb.org/howmanypeoplehaveeverlivedonearth/`
19. Prud'hommeaux, E.: Play with ShEx-generated forms. `https://github.com/ericprud/shex-form` (2020)
20. Prud'hommeaux, E., Boneva, I., Gayo, J.E.L., Kellogg, G.: Shape expressions language 2.1. Final community group report 8 october 2019, W3C Community Group (2019), `http://shex.io/shex-semantics/`
21. RDFShape: Parse and convert schema, `http://rdfshape.herokuapp.com/schemaConversions`
22. Reenskaug, T.: The original MVC reports. Tech. rep., Dept. of Informatics, University of Oslo (February 2007), `http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf`
23. Speicher, S., Arwe, J., Malhotra, A.: Linked data platform 1.0. W3C Recommendation, w3c (feb 2015), `https://www.w3.org/TR/ldp/`
24. Stapleton, G., Howse, J., Taylor, K., Delaney, A., Burton, J., Chapman, P.: Towards diagrammatic ontology patterns. In: Workshop on Ontology and Semantic Web Patterns (WOP 2013). vol. 1188. CEUR proceedings (October 2013), `http://ceur-ws.org/Vol-1188/paper_4.pdf`
25. Thornton, K., Solbrig, H., Stupp, G.S., Labra Gayo, J.E., Mietchen, D., Prud'hommeaux, E., Waagmeester, A.: Using shape expressions (ShEx) to share RDF data models and to guide curation with rigorous validation. In: The Semantic Web. pp. 606–620. Springer, Cham (2019)
26. Tim Berners-Lee: Linked data shapes, forms and footprints. Tech. rep., w3c (December 2019), `https://www.w3.org/DesignIssues/Footprints.html`
27. United Nations: World population prospects 2019: Data booklet (2019), `https://population.un.org/wpp/Publications/Files/WPP2019_DataBooklet.pdf`, retrieved April 2020
28. Völkel, M., Krötzsch, M., Vrandecic, D., Haller, H., Studer, R.: Semantic wikipedia. In: Proceedings of the 15th International Conference on World Wide Web. p. 585–594. WWW '06, ACM, New York, NY, USA (2006)
29. Vrandeĉić, D., Krötzsch, M.: Wikidata: A free collaborative knowledgebase. Communications of the ACM **57**(10), 78–85 (2014)
30. Wikiproject schemas, `https://www.wikidata.org/wiki/Wikidata:WikiProject_Schemas`
31. Wikidata: human (E10) (2020), `https://www.wikidata.org/wiki/EntitySchema:E10`
32. Wikiproject source metadata, `https://www.wikidata.org/wiki/Wikidata:WikiProject_Source_MetaData`